Guide du développeur, version 1

AWS IoT Greengrass

AWS IoT Greengrass: Guide du développeur, version 1

Table of Contents

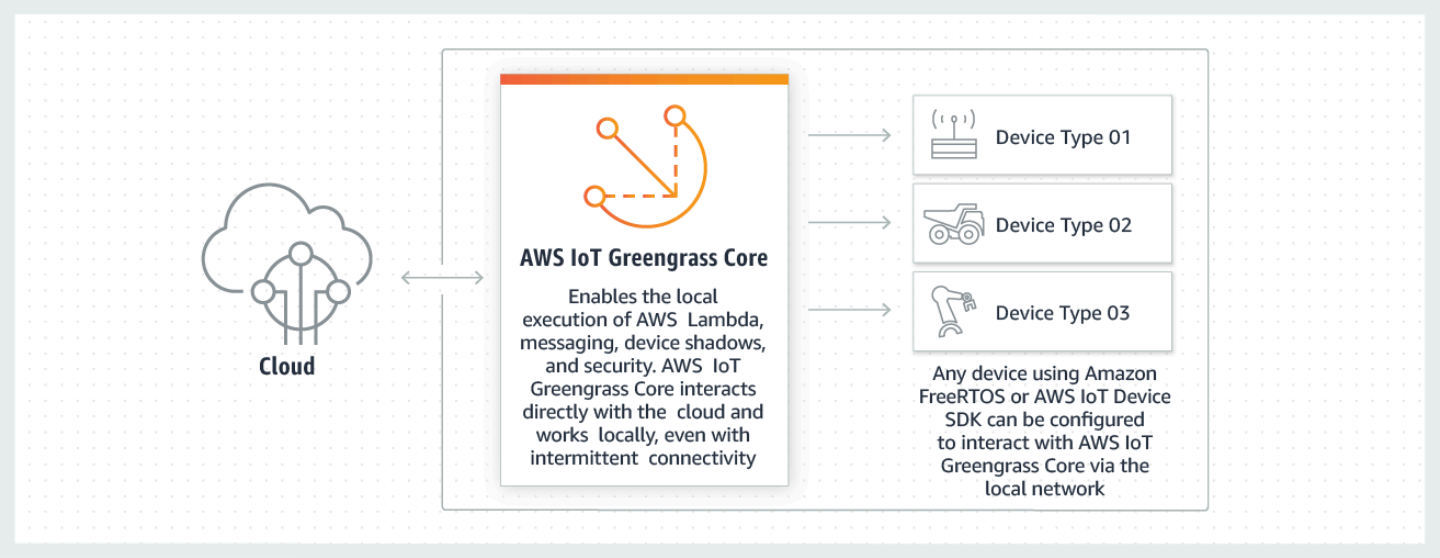
AWS IoT Greengrass Version 1 est entré dans la phase de durée de vie prolongée le 30 juin 2023. Pour plus d'informations, consultez la politique de AWS IoT Greengrass V1 maintenance. Après cette date, AWS IoT Greengrass V1 ne publiera pas de mises à jour fournissant des fonctionnalités, des améliorations, des corrections de bogues ou des correctifs de sécurité. Les appareils qui fonctionnent AWS IoT Greengrass V1 sous tension ne seront pas perturbés et continueront à fonctionner et à se connecter au cloud. Nous vous recommandons vivement de migrer vers AWS IoT Greengrass Version 2, qui ajoute de nouvelles fonctionnalités importantes et prend en charge des plateformes supplémentaires.

Les traductions sont fournies par des outils de traduction automatique. En cas de conflit entre le contenu d'une traduction et celui de la version originale en anglais, la version anglaise prévaudra.

Qu'est-ce qu'AWS IoT Greengrass ?

AWS IoT Greengrass est un logiciel qui étend les fonctionnalités du cloud aux appareils locaux. Il permet aux appareils de collecter et d'analyser les données plus près de la source des informations, de réagir de manière autonome aux événements locaux et de communiquer en toute sécurité sur les réseaux locaux. Les appareils locaux peuvent également communiquer en toute sécurité avec AWS IoT Core et exporter des données IoT vers leAWS Cloud. AWS IoT Greengrassles développeurs peuvent utiliser AWS Lambda des fonctions et des connecteurs prédéfinis pour créer des applications sans serveur déployées sur des appareils pour une exécution locale.

Le schéma suivant illustre l'architecture de base de AWS IoT Greengrass.



AWS IoT Greengrass permet aux clients de construire des appareils IoT et une logique d'application. Plus précisément, AWS IoT Greengrass permet la gestion basée sur le cloud de logique d'application qui s'exécute sur les appareils. Les fonctions et connecteurs Lambda déployés localement sont déclenchés par des événements locaux, des messages provenant du cloud ou d'autres sources.

Dans AWS IoT Greengrass, les appareils communiquent en toute sécurité sur un réseau local et échangent des messages sans avoir à se connecter au cloud. AWS IoT Greengrass fournit un gestionnaire de messages pub/sub local qui peut mettre les messages en mémoire tampon de façon intelligente en cas de perte de connexion. Les messages entrants et sortants échangés avec le cloud sont ainsi préservés.

AWS IoT Greengrass protège les données utilisateur :

- par le biais de l'authentification sécurisée et de l'autorisation des appareils ;

- grâce à la connectivité sécurisée dans le réseau local ;
- entre les appareils locaux et le cloud.

Les informations d'identification de sécurité d'un appareil fonctionnent dans un groupe jusqu'à leur révocation et ce, même si la connectivité au cloud est interrompue, afin que les appareils puissent continuer à communiquer de manière sécurisée en local.

AWS IoT Greengrassfournit des over-the-air mises à jour sécurisées des fonctions Lambda.

AWS IoT Greengrass comprend :

- Distributions logicielles
  - Logiciel AWS IoT Greengrass Core
  - Kit SDK AWS IoT Greengrass Core
- Le services cloud
  - API AWS IoT Greengrass
- Fonctionnalités
  - Le fichier d’exécution Lambda
  - Implémentation des shadows
  - Le gestionnaire de messages
  - La gestion des groupes
  - Le service de découverte
  - Agent de ver-the-air mise à jour O
  - Gestionnaire de flux
  - Accès aux ressources locales
  - Inférence d'apprentissage automatique locale
  - Secrets Manager local
  - Connecteurs avec intégration prédéfinie avec des services, protocoles et logiciels

Rubriques

- Kits SDK
- Exigences et plateformes prises en charge
- Téléchargements AWS IoT Greengrass
- Nous voulons entendre parler de vous
- Installer le logiciel AWS IoT Greengrass Core
- Configuration de AWS IoT Greengrass Core

Logiciel AWS IoT Greengrass Core

Le logiciel AWS IoT Greengrass Core fournit les fonctionnalités suivantes :

- Déploiement et exécution locale de connecteurs et de fonctions Lambda.
- Traitez les flux de données localement avec des exportations automatiques vers leAWS Cloud.
- Messagerie MQTT sur le réseau local entre les appareils, les connecteurs et les fonctions Lambda à l'aide d'abonnements gérés.
- Messagerie MQTT entre appareils, connecteurs AWS IoT et fonctions Lambda à l'aide d'abonnements gérés.
- Connexions sécurisées entre les appareils et AWS Cloud utilisation de l'authentification et de l'autorisation des appareils.
- Synchronisation cachée locale des appareils. Les ombres peuvent être configurées pour être synchronisées avecAWS Cloud.
- Accès contrôlé à l'appareil local et aux ressources de volume.
- Déploiement de modèles d'apprentissage automatique formés dans le cloud pour exécution de l'inférence locale.
- Détection automatique d'adresse IP qui permet aux appareils de détecter votre appareil Greengrass principal.
- Déploiement central de nouvelles configurations de groupe ou mises à jour. Une fois les données de configuration téléchargée, l'appareil principal redémarre automatiquement.
- Mises à jour logicielles sécurisées over-the-air (OTA) des fonctions Lambda définies par l'utilisateur.
- Stockage sécurisé et crypté des secrets locaux et accès contrôlé par des connecteurs et des fonctions Lambda.

AWS IoT Greengrassles instances principales sont configurées via AWS IoT Greengrass des API qui créent et mettent à jour les définitions de AWS IoT Greengrass groupes stockées dans le cloud.

Versions logicielles AWS IoT Greengrass Core

AWS IoT Greengrass fournit plusieurs options pour installer le logiciel AWS IoT Greengrass Core, y compris les fichiers de téléchargement targ.gz, un script de démarrage rapide et des installations de `apt` sur les plateformes Debian prises en charge. Pour plus d'informations, consultez the section called “Installer le logiciel AWS IoT Greengrass Core”.

Les onglets suivants décrivent les nouveautés et les modifications dans les versions du logiciel AWS IoT Greengrass Core.

GGC v1.11

1,11.6

Correctifs de bogues et améliorations :

- Résilience améliorée en cas de coupure de courant soudaine lors d'un déploiement.
- Correction d'un problème en raison duquel la corruption des données du gestionnaire de flux pouvait empêcher le démarrage du logiciel AWS IoT Greengrass Core.
- Correction d'un problème qui empêchait les nouveaux appareils clients de se connecter au cœur dans certains scénarios.
- Correction d'un problème en raison duquel les noms de flux du gestionnaire de flux ne pouvaient pas contenir `.log`.

1.11.5

Correctifs de bogues et améliorations :

- Améliorations des performances générales et correctifs de bogues.

1.11.4

Correctifs de bogues et améliorations :

- Correction d'un problème lié au gestionnaire de flux qui empêchait les mises à niveau vers le logiciel AWS IoT Greengrass Core v1.11.3. Si vous utilisez le gestionnaire de flux pour exporter des données vers le cloud, vous pouvez désormais utiliser une mise à jour OTA pour mettre à niveau une version v1.x antérieure du logiciel AWS IoT Greengrass Core vers la version v1.11.4.
- Améliorations des performances générales et correctifs de bogues.

1.11.3

Correctifs de bogues et améliorations :

- Correction d'un problème en raison duquel le logiciel AWS IoT Greengrass Core s'exécutant en un clin d'œil sur un appareil Ubuntu ne répondait plus après une coupure de courant soudaine de l'appareil.
- Correction d'un problème qui retardait la livraison des messages MQTT aux fonctions Lambda à longue durée de vie.
- Correction d'un problème en raison duquel les messages MQTT n'étaient pas envoyés correctement lorsque la `maxWorkItemCount` valeur était définie sur une valeur supérieure `1024` à.
- Correction d'un problème en raison duquel l'agent de mise à jour OTA ignorait la `KeepAlive` période MQTT spécifiée dans la `keepAlive` propriété dans `config.json`.
- Améliorations des performances générales et correctifs de bogues.

> **Important**
>
> Si vous utilisez le gestionnaire de flux pour exporter des données vers le cloud, n'effectuez pas de mise à niveau vers le logiciel AWS IoT Greengrass Core v1.11.3 à partir d'une version v1.x antérieure. Si vous activez le gestionnaire de flux pour la première fois, nous vous recommandons vivement d'installer d'abord la dernière version du logiciel AWS IoT Greengrass Core.

1.11.1

Correctifs de bogues et améliorations :

- Correction d'un problème qui entraînait une augmentation de l'utilisation de la mémoire par le gestionnaire de flux.
- Correction d'un problème en raison duquel le gestionnaire de flux réinitialisait le numéro de séquence du flux `0` si le périphérique principal de Greengrass était éteint pendant une période supérieure à la période spécifiée time-to-live (TTL) pour les données du flux.
- Correction d'un problème qui empêchait le gestionnaire de flux d'arrêter correctement les tentatives d'exportation de données vers leAWS Cloud.

1.11.0

Nouvelles fonctions :

- Un agent de télémétrie situé sur le cœur de Greengrass collecte les données de télémétrie locales et les publie sur. AWS Cloud Pour récupérer les données de télémétrie afin de les traiter ultérieurement, les clients peuvent créer une EventBridge règle Amazon et s'abonner à une cible. Pour plus d'informations, consultez la section Collecte des données de télémétrie relatives à l'état du système à partir des AWS IoT Greengrass principaux appareils.
- Une API HTTP locale renvoie un instantané de l'état actuel des processus de travail locaux lancés parAWS IoT Greengrass. Pour plus d'informations, consultez la section Appel de l'API de vérification de santé locale.
- Un gestionnaire de flux exporte automatiquement les données vers Amazon S3 etAWS IoT SiteWise.

  Les nouveaux paramètres du gestionnaire de flux vous permettent de mettre à jour les flux existants et de suspendre ou de reprendre l'exportation de données.
- Support pour l'exécution des fonctions Lambda de Python 3.8.x sur le noyau.
- Une nouvelle `ggDaemonPort` propriété utilisée pour configurer le numéro de port IPC principal de Greengrass. `config.json` Le numéro de port par défaut est 8000.

  Une nouvelle `systemComponentAuthTimeout` propriété `config.json`que vous utilisez pour configurer le délai d'expiration pour l'authentification IPC principale de Greengrass. Le délai d'expiration par défaut est de 5 000 millisecondes.
- Le nombre maximum d'AWS IoTappareils par AWS IoT Greengrass groupe a été augmenté de 200 à 2 500.

  Le nombre maximum d'abonnements par groupe a été augmenté de 1 000 à 10 000.

  Pour de plus amples informations, consultez AWS IoT Greengrass Points de terminaison et quotas.

Correctifs de bogues et améliorations :

- Optimisation générale permettant de réduire l'utilisation de la mémoire par les processus du service Greengrass.
- Un nouveau paramètre de configuration d'exécution (`mountAllBlockDevices`) permet à Greengrass d'utiliser des montages par liaison pour monter tous les périphériques en mode bloc dans un conteneur après avoir configuré OverlayFS. Cette fonctionnalité a résolu un problème qui provoquait l'échec du déploiement de Greengrass s'il ne se `/usr` trouvait pas dans la `/` hiérarchie.

- Correction d'un problème qui provoquait une défaillance AWS IoT Greengrass de base s'il `/tmp` s'agissait d'un lien symbolique.
- Correction d'un problème qui permettait à l'agent de déploiement Greengrass de supprimer du dossier les artefacts de modèles d'apprentissage automatique non utilisés. `mlmodel_public`
- Améliorations des performances générales et correctifs de bogues.

Extended life versions

1,1,5

Correctifs de bogues et améliorations :

- Améliorations des performances générales et correctifs de bogues.

1.10.4

Correctifs de bogues et améliorations :

- Correction d'un problème en raison duquel le logiciel AWS IoT Greengrass Core s'exécutant en un clin d'œil sur un appareil Ubuntu ne répondait plus après une coupure de courant soudaine de l'appareil.
- Correction d'un problème qui retardait la livraison des messages MQTT aux fonctions Lambda à longue durée de vie.
- Correction d'un problème en raison duquel les messages MQTT n'étaient pas envoyés correctement lorsque la `maxWorkItemCount` valeur était définie sur une valeur supérieure `1024` à.
- Correction d'un problème en raison duquel l'agent de mise à jour OTA ignorait la `KeepAlive` période MQTT spécifiée dans la `keepAlive` propriété dans `config.json`.
- Améliorations des performances générales et correctifs de bogues.

1.10.3

Correctifs de bogues et améliorations :

- Une nouvelle `systemComponentAuthTimeout` propriété `config.json`que vous utilisez pour configurer le délai d'expiration pour l'authentification IPC principale de Greengrass. Le délai d'expiration par défaut est de 5 000 millisecondes.
- Correction d'un problème qui entraînait une augmentation de l'utilisation de la mémoire par le gestionnaire de flux.

1.10.2

Correctifs de bogues et améliorations :

- Nouvelle `mqttOperationTimeout` propriété du fichier config.json que vous utilisez pour définir le délai d'expiration des opérations de publication, d'abonnement et de désabonnement dans les connexions MQTT avec. AWS IoT Core
- Améliorations des performances générales et correctifs de bogues.

1.10.1

Correctifs de bogues et améliorations :

- Le gestionnaire de flux est plus résilient à la corruption des données de fichier.
- Correction d'un problème provoquant une défaillance du montage sysfs sur les périphériques utilisant le noyau Linux 5.1 et versions ultérieures.
- Améliorations des performances générales et correctifs de bogues.

1.10.0

Nouvelles fonctions :

- Un gestionnaire de flux qui traite les flux de données localement et les exporte AWS Cloud automatiquement vers le. Cette fonctionnalité nécessite Java 8 sur l'appareil central Greengrass. Pour plus d’informations, consultez Gestion des flux de données.
- Nouveau connecteur de déploiement d'application Greengrass Docker qui exécute une application Docker sur un appareil principal (noyau). Pour plus d’informations, consultez the section called “Déploiement d'applications Docker”.
- Un nouveau SiteWise connecteur IoT qui envoie les données des appareils industriels depuis les serveurs OPC-UA vers les propriétés des actifs. AWS IoT SiteWise Pour plus d’informations, consultez the section called “IoT SiteWise”.
- Les fonctions Lambda qui s'exécutent sans conteneurisation peuvent accéder aux ressources d'apprentissage automatique du groupe Greengrass. Pour plus d’informations, consultez the section called “Accès aux ressources de machine learning”.
- Prise en charge des sessions persistantes MQTT avec AWS IoT. Pour plus d’informations, consultez the section called “Sessions persistantes MQTT avec AWS IoT Core”.
- Le trafic MQTT local peut circuler sur un port autre que le port par défaut 8883. Pour plus d’informations, consultez the section called “Port MQTT pour la messagerie locale”.
- Nouvelles `queueFullPolicy` options du SDK AWS IoT Greengrass principal pour une publication fiable des messages à partir des fonctions Lambda.

- Support pour l'exécution des fonctions Lambda de Node.js 12.x sur le noyau.
- Les mises à jour O ver-the-air (OTA) avec intégration de la sécurité matérielle peuvent être configurées avec OpenSSL 1.1.
- Améliorations des performances générales et correctifs de bogues.

1.9.4

Correctifs de bogues et améliorations :

- Améliorations des performances générales et correctifs de bogues.

1.9.3

Nouvelles fonctions :

- Support pour ARMv6L. AWS IoT Greengrass Le logiciel de base v1.9.3 ou version ultérieure peut être installé sur les distributions Raspbian sur les architectures ARMv6L (par exemple, sur les appareils Raspberry Pi Zero).
- Mises à jour OTA sur le port 443 avec ALPN. Les cœurs Greengrass qui utilisent le port 443 pour le trafic MQTT prennent désormais en charge les mises à jour over-the-air logicielles (OTA). AWS IoT Greengrassutilise l'extension TLS ALPN (Application Layer Protocol Network) pour activer ces connexions. Pour plus d'informations, consultez Mises à jour OTA du logiciel AWS IoT Greengrass Core et the section called "Connexion au port 443 ou via un proxy réseau".

Correctifs de bogues et améliorations :

- Corrige un bogue introduit dans la version v1.9.0 qui empêchait les fonctions Lambda de Python 2.7 d'envoyer des charges utiles binaires à d'autres fonctions Lambda.
- Améliorations des performances générales et correctifs de bogues.

1.9.2

Nouvelles fonctions :

- Support pour OpenWrt. AWS IoT Greengrass Le logiciel de base v1.9.2 ou version ultérieure peut être installé sur les OpenWrt distributions avec les architectures Armv8 (AArch64) et ARMv7L. Actuellement, OpenWrt ne prend pas en charge l'inférence ML.

1.9.1

Correctifs de bogues et améliorations :

- Correctif d'un bogue introduit dans la version 1.9.0 qui supprime les messages `cloud` contenant des caractères génériques dans le sujet.

1.9.0

Nouvelles fonctions :

- Support pour les environnements d'exécution Lambda de Python 3.7 et Node.js 8.10. Les fonctions Lambda qui utilisent les environnements d'exécution Python 3.7 et Node.js 8.10 peuvent désormais être exécutées sur un cœur. AWS IoT Greengrass (AWS IoT Greengrasscontinue de prendre en charge les environnements d'exécution Python 2.7 et Node.js 6.10.)
- Connexions MQTT optimisées. Le noyau Greengrass établit moins de connexions avec le noyau AWS IoT Core. Cette modification peut réduire les coûts d'exploitation pour les frais qui sont calculés en fonction du nombre de connexions.
- Clé à courbe elliptique (EC) pour le serveur MQTT local. Le serveur MQTT local prend en charge les clés EC, en plus des clés RSA. (Le certificat du serveur MQTT présente une signature RSA SHA-256, quel que soit le type de clé.) Pour plus d'informations, consultez the section called “Mandataires de sécurité”.

Correctifs de bogues et améliorations :

- Améliorations des performances générales et correctifs de bogues.

1.8.4

Résolution d'un problème lié à la synchronisation des shadows et à la reconnexion du gestionnaire de certificats d'appareil.

Améliorations des performances générales et correctifs de bogues.

1.8.3

Améliorations des performances générales et correctifs de bogues.

1.8.2

Améliorations des performances générales et correctifs de bogues.

1.8.1

Améliorations des performances générales et correctifs de bogues.

1.8.0

Nouvelles fonctions :

- Identité d'accès par défaut configurable pour les fonctions Lambda du groupe. Ce paramètre au niveau du groupe détermine les autorisations par défaut utilisées pour exécuter les fonctions Lambda. Vous pouvez définir l'ID d'utilisateur, l'ID de groupe ou les deux. Les fonctions Lambda individuelles peuvent remplacer l'identité d'accès par défaut de leur groupe. Pour plus d'informations, consultez the section called "Définition de l'identité d'accès par défaut pour les fonctions Lambda dans un groupe".
- Trafic HTTPS sur le port 443. La communication HTTPS peut être configurée pour acheminer le trafic sur le port 443 plutôt que sur le port 8443 par défaut. Cela complète la AWS IoT Greengrass prise en charge de l'extension TLS ALPN (Application Layer Protocol Network) et permet à tout le trafic de messagerie Greengrass (MQTT et HTTPS) d'utiliser le port 443. Pour plus d'informations, consultez the section called "Connexion au port 443 ou via un proxy réseau".
- ID de client nommés de manière prévisible pour les connexions AWS IoT. Cette modification permet la prise en charge des événements liés au AWS IoT cycle de vie AWS IoT Device Defender et vous permet de recevoir des notifications pour les événements de connexion, de déconnexion, d'abonnement et de désinscription. L'attribution de noms de manière prévisible facilite également la création d'une logique pour les ID de connexion (par exemple, pour créer des modèles de stratégie d'abonnement basés sur les attributs de certificat). Pour plus d'informations, consultez the section called "ID client pour les connexions MQTT avec AWS IoT".

Correctifs de bogues et améliorations :

- Résolution d'un problème lié à la synchronisation des shadows et à la reconnexion du gestionnaire de certificats d'appareil.
- Améliorations des performances générales et correctifs de bogues.

1.7.1

Nouvelles fonctions :

- Les connecteurs Greengrass fournissent une intégration intégrée à l'infrastructure locale, aux protocoles des appareils et à d'autres services cloud. AWS Pour plus d'informations, consultez Intégrer à des services et protocoles à l'aide de connecteurs.
- AWS IoT Greengrasss'étend AWS Secrets Manager aux appareils principaux, ce qui met vos mots de passe, jetons et autres secrets à la disposition des connecteurs et des fonctions Lambda. Les secrets sont chiffrés au repos et en transit. Pour plus d'informations, consultez Déployer des secrets sur Core.

- Prise en charge d'une racine matérielle d'une option de sécurité fiable. Pour plus d'informations, consultez the section called “Intégration de sécurité matérielle”.
- Paramètres d'isolation et d'autorisation qui permettent aux fonctions Lambda de s'exécuter sans les conteneurs Greengrass et d'utiliser les autorisations d'un utilisateur et d'un groupe spécifiés. Pour plus d'informations, consultez the section called “Contrôle de l'exécution de la fonction Greengrass Lambda”.
- Vous pouvez exécuter AWS IoT Greengrass dans un conteneur Docker (sur Windows, macOS ou Linux) en configurant le groupe Greengrass pour une exécution sans conteneurisation. Pour plus d'informations, consultez the section called “Exécuter AWS IoT Greengrass dans un conteneur Docker”.
- La messagerie MQTT sur le port 443 avec le protocole de négociation de la couche d'application (ALPN) ou la via un proxy réseau. Pour plus d'informations, consultez the section called “Connexion au port 443 ou via un proxy réseau”.
- Le moteur d'apprentissage profond SageMaker Neo, qui prend en charge les modèles d'apprentissage automatique optimisés par le compilateur d'apprentissage profond SageMaker Neo. Pour de plus amples informations sur l'exécution Neo deep learning, veuillez consulter the section called “Environnements d'exécution et bibliothèques pour l'inférence ML”.
- Prise en charge de Raspbian Stretch (2018-06-27) sur les principaux appareils Raspberry Pi.

Correctifs de bogues et améliorations :

- Améliorations des performances générales et correctifs de bogues.

En outre, les fonctions suivantes sont disponibles avec cette version :

- AWS IoT Device Tester pour AWS IoT Greengrass, que vous pouvez utiliser pour vérifier que votre architecture d'UC, la configuration du noyau et les pilotes fonctionnent avec AWS IoT Greengrass. Pour plus d'informations, consultez Utilisation de AWS IoT Device Tester pour AWS IoT Greengrass V1.
- Les packages AWS IoT Greengrass Core Software, AWS IoT Greengrass Core SDK et AWS IoT Greengrass Machine Learning SDK peuvent être téléchargés via Amazon. CloudFront Pour plus d'informations, consultez the section called “Téléchargements AWS IoT Greengrass”.

1.6.1

Nouvelles fonctions :

- Exécutables Lambda qui exécutent du code binaire sur le noyau de Greengrass. Utilisez le nouveau SDK AWS IoT Greengrass Core pour C pour écrire des exécutables Lambda en C et C++. Pour plus d'informations, consultez the section called "Exécutables Lambda".
- Mise en cache des messages de stockage local (facultative) qui peuvent être conservés lors des redémarrages. Vous pouvez configurer les paramètres de stockage pour les messages MQTT mis en file d'attente pour traitement. Pour plus d'informations, consultez the section called "File d'attente de messages MQTT".
- Intervalle maximal de tentative de reconnexion configurable lorsque l'appareil du noyau est déconnecté. Pour plus d'informations, consultez la propriété `mqttMaxConnectionRetryInterval` dans the section called "Fichier de configuration de AWS IoT Greengrass Core".
- Accès aux ressources locales du répertoire /proc de l'hôte. Pour plus d'informations, consultez Accès aux ressources locales.
- Répertoire en écriture configurable. Le logiciel AWS IoT Greengrass peut être déployé sur des emplacements en lecture seule et en lecture-écriture. Pour plus d'informations, consultez the section called "Configuration d'un répertoire en écriture".

Correctifs de bogues et améliorations :

- Amélioration des performances pour la publication de messages dans le noyau Greengrass, et entre les appareils et le noyau.
- Réduction des ressources de calcul requises pour traiter les journaux générés par les fonctions Lambda définies par l'utilisateur.

1.5.0

Nouvelles fonctions :

- L'inférence de Machine Learning AWS IoT Greengrass est généralement disponible. Vous pouvez exécuter l'inférence de Machine Learning localement sur des appareils AWS IoT Greengrass à l'aide de modèles créés et formés dans le cloud. Pour plus d'informations, consultez Exécuter l'inférence de Machine Learning.
- Les fonctions Lambda de Greengrass prennent désormais en charge les données binaires en tant que charge utile d'entrée, en plus du JSON. Pour utiliser cette fonctionnalité, vous devez passer à la version 1.1.0 du SDK AWS IoT Greengrass Core, que vous pouvez télécharger depuis la page de téléchargement du SDK AWS IoT Greengrass Core.

Correctifs de bogues et améliorations :

- Réduction de l'empreinte mémoire globale.
- Amélioration des performances lors de l'envoi de messages dans le cloud.
- Amélioration des performances et de la stabilité de l'agent de téléchargement, du gestionnaire de certificats de l'appareil et de l'agent de mise à jour OTA.
- Correctifs de bogues mineurs.

1.3.0

Nouvelles fonctions :

- Agent de mise à jour O ver-the-air (OTA) capable de gérer les tâches de mise à jour Greengrass déployées dans le cloud. L'agent se trouve dans le nouveau répertoire `/greengrass/ota`. Pour plus d'informations, consultez Mises à jour OTA du logiciel AWS IoT Greengrass Core.
- La fonction d'accès aux ressources locales permet aux fonctions Lambda Greengrass d'accéder aux ressources locales, telles que des périphériques et des volumes. Pour plus d'informations, consultez Accédez à des ressources locales avec des fonctions et des connecteurs Lambda.

1.1.0

Nouvelles fonctions :

- AWS IoT GreengrassLes groupes déployés peuvent être réinitialisés en supprimant les fonctions, les abonnements et les configurations Lambda. Pour plus d'informations, consultez the section called "Réinitialiser les déploiements".
- Support des environnements d'exécution Lambda de Node.js 6.10 et Java 8, en plus de Python 2.7.

Pour effectuer une migration depuis la version précédente du AWS IoT Greengrass noyau, procédez comme suit :

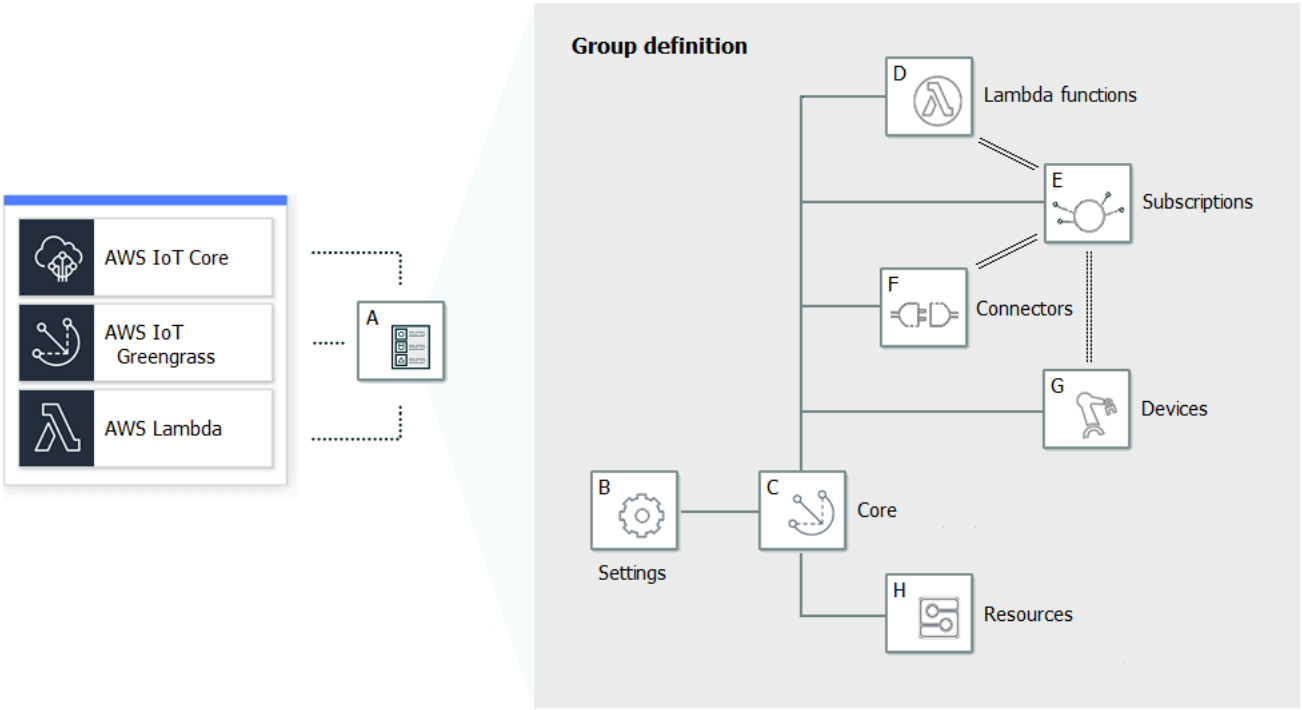
- Copiez les certificats du dossier `/greengrass/configuration/certs` dans `/greengrass/certs`.
- Copiez `/greengrass/configuration/config.json` dans `/greengrass/config/config.json`.
- Exécutez `/greengrass/ggc/core/greengrassd` au lieu de `/greengrass/greengrassd`.
- Déployez le groupe sur le nouveau noyau.

1.0.0

Première version

Groupes AWS IoT Greengrass

Un groupe Greengrass est un ensemble de paramètres et de composants, tels qu'un noyau Greengrass, des périphériques et des abonnements. Les groupes sont utilisés pour définir une portée d'interaction. Par exemple, un groupe peut représenter un étage d'un bâtiment, un camion ou l'ensemble d'un site minier. Le schéma suivant montre les composants qui constituent un groupe Greengrass.



Dans le schéma précédent :

A : définition du groupe Greengrass

Informations sur les paramètres et les composants du groupe.

B : paramètres du groupe Greengrass

Il s'agit des licences suivantes :

- Rôle du groupe Greengrass.

- Autorité de certification (CA) et configuration de la connexion locale
- Informations sur la connectivité du noyau Greengrass.
- Environnement d'exécution Lambda par défaut. Pour plus d'informations, consultez the section called "Configuration de la conteneurisation par défaut pour les fonctions Lambda dans un groupe".
- CloudWatch et configuration des journaux locaux. Pour plus d'informations, consultez the section called "Surveillance avec les journaux AWS IoT Greengrass".

C : noyau Greengrass

L'objet (périphérique) AWS IoT qui représente le noyau Greengrass. Pour plus d'informations, consultez the section called "Configuration de AWS IoT Greengrass Core".

D : Définition de la fonction Lambda

Liste des fonctions Lambda qui s'exécutent localement sur le cœur, avec les données de configuration associées. Pour plus d'informations, consultez Exécuter des fonctions Lambda locales.

E : Définition de l'abonnement

Liste des abonnements qui permettent de communiquer à l'aide des messages MQTT. Un abonnement définit :

- Une source et une cible de message. Il peut s'agir de périphériques clients, de fonctions Lambda, de connecteurs et du service parallèle local. AWS IoT Core
- Rubrique ou sujet utilisé pour filtrer les messages.

Pour plus d'informations, consultez the section called "Abonnements gérés dans le flux de travail de messagerie MQTT".

F : Définition du connecteur

Une liste des connecteurs qui s'exécutent localement sur le noyau avec les données de configuration associées. Pour plus d'informations, consultez Intégrer à des services et protocoles à l'aide de connecteurs.

G : Définition de l'appareil

Liste des AWS IoT éléments (appelés appareils clients ou appareils) membres du groupe Greengrass, avec les données de configuration associées. Pour plus d'informations, consultez the section called "Appareils dans AWS IoT Greengrass".

H : Définition des ressources

Une liste des ressources locales, des ressources d'apprentissage automatique et des ressources de secret sur le noyau Greengrass avec les données de configuration associées. Pour plus d'informations, consultez Accès aux ressources locales, Exécuter l'inférence de Machine Learning et Déployer des secrets sur Core.

Lors du déploiement, la définition du groupe Greengrass, les fonctions Lambda, les connecteurs, les ressources et la table d'abonnement sont copiés sur le périphérique principal. Pour plus d'informations, consultez Déploiement de groupes AWS IoT Greengrass.

Appareils dans AWS IoT Greengrass

Un groupe Greengrass peut contenir deux types de périphériques AWS IoT :

Noyau Greengrass

Un noyau Greengrass est un appareil qui exécute le logiciel AWS IoT Greengrass Core, ce qui lui permet de communiquer directement avec AWS IoT Core et le service AWS IoT Greengrass. Un noyau possède son propre certificat d'appareil auprès d'AWS IoT Core. Il a un shadow d'appareil et une entrée dans le registre AWS IoT Core. Les cœurs Greengrass exécutent un environnement d'exécution Lambda local, un agent de déploiement et un outil de suivi d'adresses IP qui envoient des informations d'adresse IP au AWS IoT Greengrass service pour permettre aux appareils clients de découvrir automatiquement leurs informations de connexion de groupe et de base. Pour plus d'informations, consultez the section called "Configuration de AWS IoT Greengrass Core".

> **Note**
>
> Un groupe Greengrass doit contenir exactement un noyau.

Appareil client

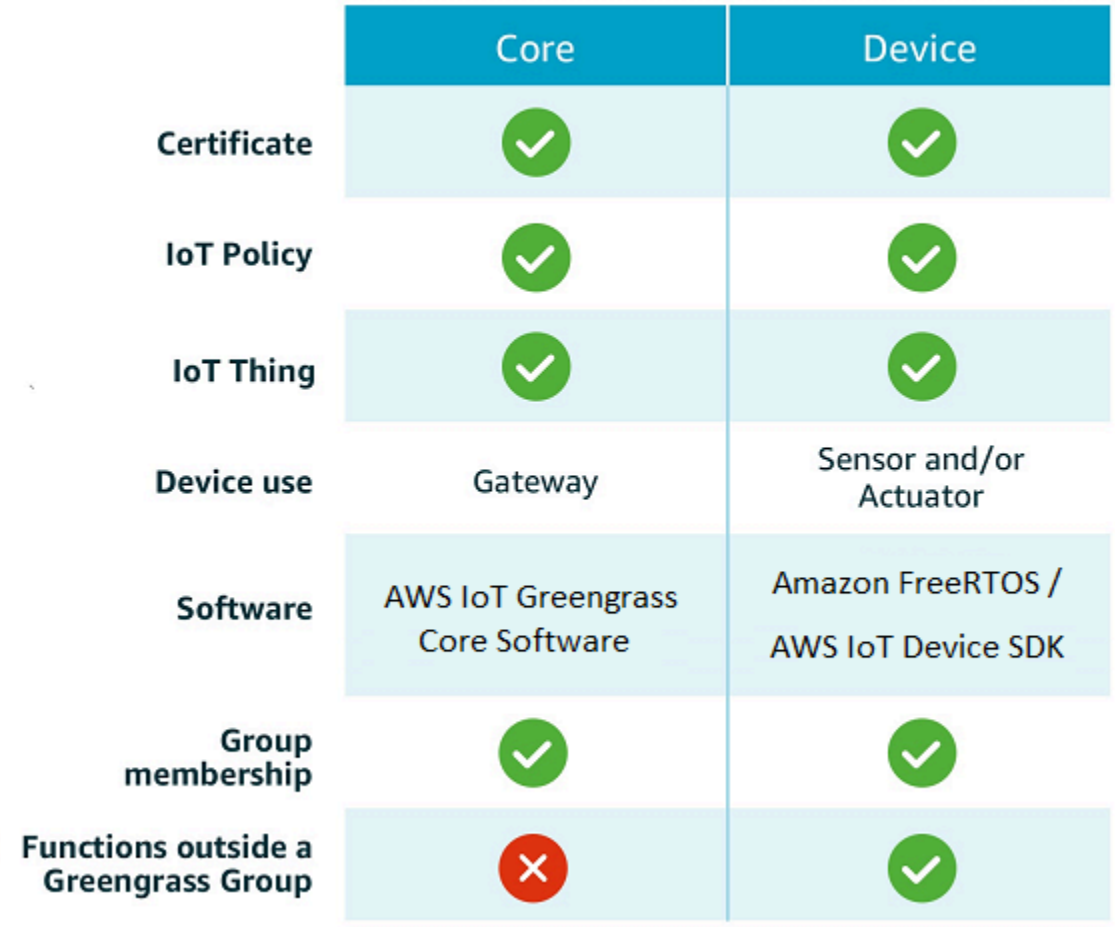
Les appareils clients (également appelés appareils connectés, appareils Greengrass ou appareils) sont des appareils qui se connectent à un cœur Greengrass via MQTT. Ils disposent de leur propre certificat d'appareil pour AWS IoT Core l'authentification, d'un shadow d'appareil et d'une entrée dans le AWS IoT Core registre. Les appareils clients peuvent exécuter FreeRTOS ou utiliser AWS IoTle Device SDK AWS IoT Greengrass ou l'API Discovery pour obtenir les

informations de découverte utilisées pour se connecter et s'authentifier auprès du noyau d'un même groupe Greengrass. Pour savoir comment utiliser la AWS IoT console pour créer et configurer un appareil client pourAWS IoT Greengrass, voirthe section called “Module 4 : Interaction avec les appareils clients dans unAWS IoT Greengrassgroupe”. Ou, pour des exemples qui vous montrent comment utiliser le AWS CLI pour créer et configurer un appareil client pourAWS IoT Greengrass, reportez-vous create-device-definitionà la référence des AWS CLI commandes.

Dans un groupe Greengrass, vous pouvez créer des abonnements qui permettent aux appareils clients de communiquer via MQTT avec les fonctions Lambda, les connecteurs et les autres appareils clients du groupe, ainsi qu'avec AWS IoT Core ou avec le service fantôme local. Les messages MQTT sont acheminés via le noyau. Si le périphérique principal perd sa connectivité au cloud, les appareils clients peuvent continuer à communiquer via le réseau local. La taille des appareils clients peut varier, qu'il s'agisse de petits appareils basés sur un microcontrôleur ou de gros appareils. À l'heure actuelle, un groupe Greengrass peut contenir jusqu'à 2 500 appareils clients. Un appareil client peut appartenir à un maximum de 10 groupes.

> **Note**
>
> OPC-UA est une norme d'échange d'informations pour la communication industrielle. Pour implémenter la prise en charge de l'OPC-UA sur le cœur de Greengrass, vous pouvez utiliser le connecteur IoT. SiteWise Le connecteur envoie les données d'appareil industrielles provenant de serveurs OPC-UA aux propriétés des ressources dans AWS IoT SiteWise.

Le tableau suivant montre comment ces types d'appareils sont associés.



L'appareil AWS IoT Greengrass principal stocke les certificats à deux emplacements :

- Certificat de l'appareil noyau dans `/greengrass-root/certs`. En règle générale, le certificat de l'appareil noyau est nommé *hash*`.cert.pem` (par exemple, `86c84488a5.cert.pem`). Ce certificat est utilisé par le client AWS IoT pour l'authentification mutuelle lorsque le noyau se connecte aux services AWS IoT Core et AWS IoT Greengrass.
- Certificat de serveur MQTT dans `/greengrass-root/ggc/var/state/server`. Le certificat de serveur MQTT est nommé `server.crt`. Ce certificat est utilisé pour l'authentification mutuelle entre le serveur MQTT local (sur le noyau Greengrass) et les appareils Greengrass.

> **Note**
>
> *`greengrass-root`* indique le chemin d'installation du logiciel AWS IoT Greengrass Core sur votre appareil. Généralement, il s'agit du répertoire `/greengrass`.

Kits SDK

Les SDK AWS fournis ci-dessous sont utilisés pour travailler avec : AWS IoT Greengrass

Kit SDK AWS

Utilisez le AWS SDK pour créer des applications qui interagissent avec n'importe quel AWS service, notamment Amazon S3, Amazon AWS IoT DynamoDBAWS IoT Greengrass,,, et bien d'autres encore. Dans le contexte deAWS IoT Greengrass, vous pouvez utiliser le AWS SDK dans les fonctions Lambda déployées pour effectuer des appels directs vers n'importe quel service. AWS Pour plus d'informations, consultez Kits SDK AWS.

> **Note**
>
> Les opérations spécifiques à Greengrass disponibles dans les AWS SDK sont également disponibles dans l'AWS IoT GreengrassAPI et. AWS CLI

SDK pour les appareils AWS IoT

Le AWS IoT SDK pour appareils aide les appareils à se connecter à AWS IoT Core etAWS IoT Greengrass. Pour plus d'informations, consultez la section SDK pour AWS IoT appareils dans le guide du AWS IoT développeur.

Les appareils clients peuvent utiliser n'importe laquelle des plateformes AWS IoT Device SDK v2 pour découvrir les informations de connectivité d'un noyau Greengrass. Les informations de connectivité incluent :

- Les identifiants des groupes Greengrass auxquels appartient l'appareil client.
- Les adresses IP du noyau Greengrass de chaque groupe. Ils sont également appelés points de terminaison principaux.
- Le certificat CA du groupe, que les appareils utilisent pour l'authentification mutuelle avec le noyau. Pour plus d'informations, consultez the section called "Flux de connexion des appareils".

> **Note**
>
> Dans la version 1 des SDK pour AWS IoT appareils, seules les plateformes C++ et Python fournissent un support de découverte intégré.

Kit SDK AWS IoT Greengrass Core

Le SDK AWS IoT Greengrass principal permet aux fonctions Lambda d'interagir avec le cœur de Greengrass, de publier des messagesAWS IoT, d'interagir avec le service parallèle local, d'invoquer d'autres fonctions Lambda déployées et d'accéder à des ressources secrètes. Ce SDK est utilisé par les fonctions Lambda qui s'exécutent sur AWS IoT Greengrass un cœur. Pour plus d'informations, consultez Kits SDK AWS IoT Greengrass Core.

AWS IoT GreengrassKit de développement logiciel pour le Machine Learning

Le SDK AWS IoT Greengrass Machine Learning permet aux fonctions Lambda d'utiliser des modèles d'apprentissage automatique déployés sur le cœur de Greengrass en tant que ressources d'apprentissage automatique. Ce SDK est utilisé par les fonctions Lambda qui s'exécutent sur AWS IoT Greengrass un cœur et interagissent avec un service d'inférence local. Pour plus d'informations, consultez AWS IoT GreengrassKit de développement logiciel (SDK) pour le Machine Learning.

Exigences et plateformes prises en charge

Les onglets suivants répertorient les plateformes prises en charge et les exigences pour le logiciel AWS IoT Greengrass Core.

> **Note**
>
> Vous pouvez télécharger le logiciel AWS IoT Greengrass principal à partir des téléchargements du logiciel AWS IoT Greengrass principal.

GGC v1.11

Plateformes prises en charge:

- Architecture : Armv7l
  - Système d'exploitation : Linux
  - Système d'exploitation : Linux (OpenWrt)
- Architecture : Armv8 (AArch64)
  - Système d'exploitation : Linux
  - Système d'exploitation : Linux (OpenWrt)

- Architecture : Armv6l
  - Système d'exploitation : Linux
- Architecture : x86_64
  - Système d'exploitation : Linux
- Les plateformes Windows, macOS et Linux peuvent exécuter AWS IoT Greengrass dans un conteneur Docker. Pour plus d'informations, consultez the section called "Exécuter AWS IoT Greengrass dans un conteneur Docker".

Prérequis:

- 128 Mo d'espace disque minimum disponible pour le logiciel AWS IoT Greengrass Core. Si vous utilisez l'agent de mise à jour OTA, le minimum est de 400 Mo.
- Minimum de 128 Mo de RAM alloués au logiciel AWS IoT Greengrass Core. Si le gestionnaire de flux est activé, le minimum passe à 198 Mo de RAM.

  > **Note**
  >
  > Le gestionnaire de flux est activé par défaut si vous utilisez l'option de création de groupe par défaut sur la AWS IoT console pour créer votre groupe Greengrass.

- Version du noyau Linux :
  - Le noyau Linux version 4.4 ou ultérieure est nécessaire pour prendre en charge l'exécution de AWS IoT Greengrass avec des conteneurs.
  - La version 3.17 ou ultérieure du noyau Linux est requise pour prendre en charge l'exécution AWS IoT Greengrass sans conteneurs. Dans cette configuration, la conteneurisation de la fonction Lambda par défaut pour le groupe Greengrass doit être définie sur Aucun conteneur. Pour obtenir des instructions, veuillez consulter the section called "Configuration de la conteneurisation par défaut pour les fonctions Lambda dans un groupe".
- Bibliothèque GNU C (glibc) version 2.14 ou ultérieure. OpenWrt les distributions nécessitent la version 1.1.16 ou ultérieure de musl C Library.
- Le répertoire `/var/run` doit être présent sur l'appareil.
- Les fichiers `/dev/stdin`, `/dev/stdout` et `/dev/stderr` doivent être disponibles.
- Les protections hardlink et softlink doivent être activées sur l'appareil. Sinon, AWS IoT Greengrass ne peut être exécuté qu'en mode non sécurisé, en utilisant l'indicateur `-i`.
- Les configurations suivantes du noyau Linux doivent être activées sur l'appareil :

- Espace de noms :
  - CONFIG_IPC_NS
  - CONFIG_UTS_NS
  - CONFIG_USER_NS
  - CONFIG_PID_NS
- Cgroups :
  - CONFIG_CGROUP_DEVICE
  - CONFIG_CGROUPS
  - CONFIG_MEMCG

  Le noyau doit prendre en charge cgroups. Les conditions requises suivantes s'appliquent lors de l'exécution d'AWS IoT Greengrass avec des conteneurs :
  - Le cgroup de mémoire doit être activé et monté pour permettre de AWS IoT Greengrass définir la limite de mémoire pour les fonctions Lambda.
  - Le cgroup des appareils doit être activé et monté si les fonctions Lambda avec accès aux ressources locales sont utilisées pour ouvrir des fichiers sur AWS IoT Greengrass le périphérique principal.
- Autres :
  - CONFIG_POSIX_MQUEUE
  - CONFIG_OVERLAY_FS
  - CONFIG_HAVE_ARCH_SECCOMP_FILTER
  - CONFIG_SECCOMP_FILTER
  - CONFIG_KEYS
  - CONFIG_SECCOMP
  - CONFIG_SHMEM
- Le certificat racine d'Amazon S3 AWS IoT doit être présent dans le magasin de confiance du système.
- Le gestionnaire de flux nécessite l'environnement d'exécution Java 8 et un minimum de 70 Mo de RAM en plus de la mémoire logicielle AWS IoT Greengrass Core de base requise. Le gestionnaire de flux est activé par défaut lorsque vous utilisez l'option de création de groupe par défaut sur la AWS IoT console. Le gestionnaire de flux n'est pas pris en charge sur OpenWrt les distributions.

- Bibliothèques prenant en charge le temps AWS Lambdad'exécution requis par les fonctions Lambda que vous souhaitez exécuter localement. Les bibliothèques requises doivent être installées sur le noyau et ajoutées à la variable d'environnement PATH. Plusieurs bibliothèques peuvent être installées sur le même noyau.
  - Python version 3.8 pour les fonctions qui utilisent le runtime Python 3.8.
  - Python version 3.7 pour les fonctions qui utilisent l'environnement d'exécution Python 3.7.
  - Python version 2.7 pour les fonctions qui utilisent l'environnement d'exécution Python 2.7.
  - Node.js version 12.x pour les fonctions qui utilisent l'environnement d'exécution Node.js 12.x.
  - Java version 8 ou ultérieure pour les fonctions qui utilisent l'environnement d'exécution Java 8.

    > **Note**
    >
    > L'exécution de Java sur une OpenWrt distribution n'est pas officiellement prise en charge. Toutefois, si votre OpenWrt version prend en charge Java, vous pourrez peut-être exécuter des fonctions Lambda créées en Java sur vos appareils. OpenWrt

    Pour plus d'informations sur la AWS IoT Greengrass prise en charge des environnements d'exécution Lambda, consultez. Exécuter des fonctions Lambda locales
- Les commandes shell suivantes (et non les BusyBox variantes) sont requises par l'agent de mise à jour 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

Plateformes prises en charge:

- Architecture : Armv7l
  - Système d'exploitation : Linux
  - Système d'exploitation : Linux (OpenWrt)
- Architecture : Armv8 (AArch64)
  - Système d'exploitation : Linux
  - Système d'exploitation : Linux (OpenWrt)
- Architecture : Armv6l
  - Système d'exploitation : Linux
- Architecture : x86_64
  - Système d'exploitation : Linux
- Les plateformes Windows, macOS et Linux peuvent exécuter AWS IoT Greengrass dans un conteneur Docker. Pour plus d'informations, consultez the section called "Exécuter AWS IoT Greengrass dans un conteneur Docker".

Prérequis:

- 128 Mo d'espace disque minimum disponible pour le logiciel AWS IoT Greengrass Core. Si vous utilisez l'agent de mise à jour OTA, le minimum est de 400 Mo.
- Minimum de 128 Mo de RAM alloués au logiciel AWS IoT Greengrass Core. Si le gestionnaire de flux est activé, le minimum passe à 198 Mo de RAM.

> **Note**
>
> Le gestionnaire de flux est activé par défaut si vous utilisez l'option de création de groupe par défaut sur la AWS IoT console pour créer votre groupe Greengrass.

- Version du noyau Linux :
  - Le noyau Linux version 4.4 ou ultérieure est nécessaire pour prendre en charge l'exécution de AWS IoT Greengrass avec des conteneurs.
  - La version 3.17 ou ultérieure du noyau Linux est requise pour prendre en charge l'exécution AWS IoT Greengrass sans conteneurs. Dans cette configuration, la conteneurisation de la fonction Lambda par défaut pour le groupe Greengrass doit être définie sur Aucun conteneur. Pour obtenir des instructions, veuillez consulter the section called “Configuration de la conteneurisation par défaut pour les fonctions Lambda dans un groupe”.
- Bibliothèque GNU C (glibc) version 2.14 ou ultérieure. OpenWrt les distributions nécessitent la version 1.1.16 ou ultérieure de musl C Library.
- Le répertoire `/var/run` doit être présent sur l'appareil.
- Les fichiers `/dev/stdin`, `/dev/stdout` et `/dev/stderr` doivent être disponibles.
- Les protections hardlink et softlink doivent être activées sur l'appareil. Sinon, AWS IoT Greengrass ne peut être exécuté qu'en mode non sécurisé, en utilisant l'indicateur `-i`.
- Les configurations suivantes du noyau Linux doivent être activées sur l'appareil :
  - Espace de noms :
    - CONFIG_IPC_NS
    - CONFIG_UTS_NS
    - CONFIG_USER_NS
    - CONFIG_PID_NS
  - Cgroups :
    - CONFIG_CGROUP_DEVICE
    - CONFIG_CGROUPS
    - CONFIG_MEMCG

    Le noyau doit prendre en charge cgroups. Les conditions requises suivantes s'appliquent lors de l'exécution d'AWS IoT Greengrass avec des conteneurs :

        - Le cgroup de mémoire doit être activé et monté pour permettre de AWS IoT Greengrass définir la limite de mémoire pour les fonctions Lambda.
        - Le cgroup des appareils doit être activé et monté si les fonctions Lambda avec accès aux ressources locales sont utilisées pour ouvrir des fichiers sur AWS IoT Greengrass le périphérique principal.
    - Autres :
        - CONFIG_POSIX_MQUEUE
        - CONFIG_OVERLAY_FS
        - CONFIG_HAVE_ARCH_SECCOMP_FILTER
        - CONFIG_SECCOMP_FILTER
        - CONFIG_KEYS
        - CONFIG_SECCOMP
        - CONFIG_SHMEM
- Le certificat racine d'Amazon S3 AWS IoT doit être présent dans le magasin de confiance du système.
- Le gestionnaire de flux nécessite l'environnement d'exécution Java 8 et un minimum de 70 Mo de RAM en plus de la mémoire logicielle AWS IoT Greengrass Core de base requise. Le gestionnaire de flux est activé par défaut lorsque vous utilisez l'option de création de groupe par défaut sur la AWS IoT console. Le gestionnaire de flux n'est pas pris en charge sur OpenWrt les distributions.
- Bibliothèques prenant en charge le temps AWS Lambdad'exécution requis par les fonctions Lambda que vous souhaitez exécuter localement. Les bibliothèques requises doivent être installées sur le noyau et ajoutées à la variable d'environnement `PATH`. Plusieurs bibliothèques peuvent être installées sur le même noyau.
    - Python version 3.7 pour les fonctions qui utilisent l'environnement d'exécution Python 3.7.
    - Python version 2.7 pour les fonctions qui utilisent l'environnement d'exécution Python 2.7.
    - Node.js version 12.x pour les fonctions qui utilisent l'environnement d'exécution Node.js 12.x.
    - Java version 8 ou ultérieure pour les fonctions qui utilisent l'environnement d'exécution Java 8.

> **Note**
>
> L'exécution de Java sur une OpenWrt distribution n'est pas officiellement prise en charge. Toutefois, si votre OpenWrt version prend en charge Java, vous pourrez peut-être exécuter des fonctions Lambda créées en Java sur vos appareils. OpenWrt

Pour plus d'informations sur la AWS IoT Greengrass prise en charge des environnements d'exécution Lambda, consultez. Exécuter des fonctions Lambda locales

- Les commandes shell suivantes (et non les BusyBox variantes) sont requises par l'agent de mise à jour 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

Plateformes prises en charge:

- Architecture : Armv7l
  - Système d'exploitation : Linux
  - Système d'exploitation : Linux (OpenWrt)
- Architecture : Armv8 (AArch64)
  - Système d'exploitation : Linux
  - Système d'exploitation : Linux (OpenWrt)
- Architecture : Armv6l
  - Système d'exploitation : Linux
- Architecture : x86_64
  - Système d'exploitation : Linux
- Les plateformes Windows, macOS et Linux peuvent exécuter AWS IoT Greengrass dans un conteneur Docker. Pour plus d'informations, consultez the section called "Exécuter AWS IoT Greengrass dans un conteneur Docker".

Prérequis:

- 128 Mo d'espace disque minimum disponible pour le logiciel AWS IoT Greengrass Core. Si vous utilisez l'agent de mise à jour OTA, le minimum est de 400 Mo.
- Minimum de 128 Mo de RAM alloués au logiciel AWS IoT Greengrass Core.
- Version du noyau Linux :
  - Le noyau Linux version 4.4 ou ultérieure est nécessaire pour prendre en charge l'exécution de AWS IoT Greengrass avec des conteneurs.
  - La version 3.17 ou ultérieure du noyau Linux est requise pour prendre en charge l'exécution AWS IoT Greengrass sans conteneurs. Dans cette configuration, la conteneurisation de la fonction Lambda par défaut pour le groupe Greengrass doit être définie sur Aucun conteneur. Pour obtenir des instructions, veuillez consulter the section called "Configuration de la conteneurisation par défaut pour les fonctions Lambda dans un groupe".
- Bibliothèque GNU C (glibc) version 2.14 ou ultérieure. OpenWrt les distributions nécessitent la version 1.1.16 ou ultérieure de musl C Library.
- Le répertoire `/var/run` doit être présent sur l'appareil.

- Les fichiers `/dev/stdin`, `/dev/stdout` et `/dev/stderr` doivent être disponibles.
- Les protections hardlink et softlink doivent être activées sur l'appareil. Sinon, AWS IoT Greengrass ne peut être exécuté qu'en mode non sécurisé, en utilisant l'indicateur `-i`.
- Les configurations suivantes du noyau Linux doivent être activées sur l'appareil :
  - Espace de noms :
    - CONFIG_IPC_NS
    - CONFIG_UTS_NS
    - CONFIG_USER_NS
    - CONFIG_PID_NS
  - Cgroups :
    - CONFIG_CGROUP_DEVICE
    - CONFIG_CGROUPS
    - CONFIG_MEMCG

    Le noyau doit prendre en charge cgroups. Les conditions requises suivantes s'appliquent lors de l'exécution d'AWS IoT Greengrass avec des conteneurs :
    - Le cgroup de mémoire doit être activé et monté pour permettre de AWS IoT Greengrass définir la limite de mémoire pour les fonctions Lambda.
    - Le cgroup des appareils doit être activé et monté si les fonctions Lambda avec accès aux ressources locales sont utilisées pour ouvrir des fichiers sur AWS IoT Greengrass le périphérique principal.
  - Autres :
    - CONFIG_POSIX_MQUEUE
    - CONFIG_OVERLAY_FS
    - CONFIG_HAVE_ARCH_SECCOMP_FILTER
    - CONFIG_SECCOMP_FILTER
    - CONFIG_KEYS
    - CONFIG_SECCOMP
    - CONFIG_SHMEM
- Le certificat racine d'Amazon S3 AWS IoT doit être présent dans le magasin de confiance du système.

- Bibliothèques prenant en charge le temps AWS Lambdad'exécution requis par les fonctions Lambda que vous souhaitez exécuter localement. Les bibliothèques requises doivent être installées sur le noyau et ajoutées à la variable d'environnement PATH. Plusieurs bibliothèques peuvent être installées sur le même noyau.
  - Python version 2.7 pour les fonctions qui utilisent l'environnement d'exécution Python 2.7.
  - Python version 3.7 pour les fonctions qui utilisent l'environnement d'exécution Python 3.7.
  - Node.js version 6.10 ou ultérieure pour les fonctions qui utilisent l'environnement d'exécution Node.js 6.10.
  - Node.js version 8.10 ou ultérieure pour les fonctions qui utilisent l'environnement d'exécution Node.js 8.10.
  - Java version 8 ou ultérieure pour les fonctions qui utilisent l'environnement d'exécution Java 8.

    > **Note**
    >
    > L'exécution de Java sur une OpenWrt distribution n'est pas officiellement prise en charge. Toutefois, si votre OpenWrt version prend en charge Java, vous pourrez peut-être exécuter des fonctions Lambda créées en Java sur vos appareils. OpenWrt

    Pour plus d'informations sur la AWS IoT Greengrass prise en charge des environnements d'exécution Lambda, consultez. Exécuter des fonctions Lambda locales
- Les commandes shell suivantes (et non les BusyBox variantes) sont requises par l'agent de mise à jour over-the-air (OTA) :
  - `wget`
  - `realpath`
  - `tar`
  - `readlink`
  - `basename`
  - `dirname`
  - `pidof`
  - `df`
  - `grep`
  - `umount`

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

GGC v1.8

- Plateformes prises en charge:
  - Architecture : ARMv7L ; Système d'exploitation : Linux
  - Architecture : x86_64 ; Système d'exploitation : Linux
  - Architecture : Armv8 (AArch64) ; Système d'exploitation : Linux
  - Les plateformes Windows, macOS et Linux peuvent exécuter AWS IoT Greengrass dans un conteneur Docker. Pour plus d'informations, consultez the section called "Exécuter AWS IoT Greengrass dans un conteneur Docker".
  - Les plateformes Linux peuvent exécuter une version de AWS IoT Greengrass avec des fonctionnalités limitées à l'aide du composant logiciel enfichable, qui est disponible via Snapcraft. Pour plus d'informations, consultez the section called "Logiciel de composant enfichable AWS IoT Greengrass".
- Configuration minimale requise :
  - 128 Mo d'espace disque minimum disponible pour le logiciel AWS IoT Greengrass Core. Si vous utilisez l'agent de mise à jour OTA, le minimum est de 400 Mo.
  - Minimum de 128 Mo de RAM alloués au logiciel AWS IoT Greengrass Core.
  - Version du noyau Linux :
    - Le noyau Linux version 4.4 ou ultérieure est nécessaire pour prendre en charge l'exécution de AWS IoT Greengrass avec des conteneurs.
    - La version 3.17 ou ultérieure du noyau Linux est requise pour prendre en charge l'exécution AWS IoT Greengrass sans conteneurs. Dans cette configuration, la conteneurisation de la fonction Lambda par défaut pour le groupe Greengrass doit être définie sur Aucun conteneur. Pour obtenir des instructions, veuillez consulter the section

called “Configuration de la conteneurisation par défaut pour les fonctions Lambda dans un groupe”.

- Bibliothèque C GNU (glibc) version 2.14 ou ultérieure.
- Le répertoire `/var/run` doit être présent sur l'appareil.
- Les fichiers `/dev/stdin`, `/dev/stdout` et `/dev/stderr` doivent être disponibles.
- Les protections hardlink et softlink doivent être activées sur l'appareil. Sinon, AWS IoT Greengrass ne peut être exécuté qu'en mode non sécurisé, en utilisant l'indicateur `-i`.
- Les configurations suivantes du noyau Linux doivent être activées sur l'appareil :
  - Espace de noms :
    - CONFIG_IPC_NS
    - CONFIG_UTS_NS
    - CONFIG_USER_NS
    - CONFIG_PID_NS
  - Cgroups :
    - CONFIG_CGROUP_DEVICE
    - CONFIG_CGROUPS
    - CONFIG_MEMCG

    Le noyau doit prendre en charge cgroups. Les conditions requises suivantes s'appliquent lors de l'exécution d'AWS IoT Greengrass avec des conteneurs :

    - Le cgroup de mémoire doit être activé et monté pour permettre de AWS IoT Greengrass définir la limite de mémoire pour les fonctions Lambda.
    - Le cgroup des appareils doit être activé et monté si les fonctions Lambda avec accès aux ressources locales sont utilisées pour ouvrir des fichiers sur AWS IoT Greengrass le périphérique principal.
  - Autres :
    - CONFIG_POSIX_MQUEUE
    - CONFIG_OVERLAY_FS
    - CONFIG_HAVE_ARCH_SECCOMP_FILTER
    - CONFIG_SECCOMP_FILTER
    - CONFIG_KEYS
    - CONFIG_SECCOMP

        - CONFIG_SHMEM
    - Le certificat racine d'Amazon S3 AWS IoT doit être présent dans le magasin de confiance du système.
- Les éléments suivants sont soumis à condition :
    - Bibliothèques prenant en charge le temps AWS Lambdad'exécution requis par les fonctions Lambda que vous souhaitez exécuter localement. Les bibliothèques requises doivent être installées sur le noyau et ajoutées à la variable d'environnement `PATH`. Plusieurs bibliothèques peuvent être installées sur le même noyau.
        - Python version 2.7 pour les fonctions qui utilisent l'environnement d'exécution Python 2.7.
        - Node.js version 6.10 ou ultérieure pour les fonctions qui utilisent l'environnement d'exécution Node.js 6.10.
        - Java version 8 ou ultérieure pour les fonctions qui utilisent l'environnement d'exécution Java 8.
    - Les commandes shell suivantes (et non les BusyBox variantes) sont requises par l'agent de mise à jour over-the-air (OTA) :
        - `wget`
        - `realpath`
        - `tar`
        - `readlink`
        - `basename`
        - `dirname`
        - `pidof`
        - `df`
        - `grep`
        - `umount`
        - `mv`
        - `gzip`
        - `mkdir`
        - `rm`
        - `ln`
        - `cut`

- `cat`

Pour de plus amples informations sur les quotas (limites) AWS IoT Greengrass, veuillez consulter Quotas de service dans la Référence générale d'Amazon Web Services.

Pour en savoir plus sur la tarification, veuillez consulter Tarification AWS IoT Greengrass et Tarification AWS IoT Core.

Téléchargements AWS IoT Greengrass

Vous pouvez utiliser les informations suivantes pour rechercher et télécharger des logiciels pour une utilisation avec AWS IoT Greengrass.

Rubriques

- Logiciel AWS IoT Greengrass Core
- Logiciel de composant enfichable AWS IoT Greengrass
- Logiciel AWS IoT Greengrass Docker
- Kit SDK AWS IoT Greengrass Core
- Bibliothèques et environnements d'exécution de machine learning pris en charge
- Logiciel du kit SDK de machine learning AWS IoT Greengrass

Logiciel AWS IoT Greengrass Core

Le logiciel AWS IoT Greengrass Core étend les AWS fonctionnalités à un appareil AWS IoT Greengrass principal, ce qui permet aux appareils locaux d'agir localement sur les données qu'ils génèrent.

v1.11

1,11.6

Correctifs de bogues et améliorations :

- Résilience améliorée en cas de coupure de courant soudaine lors d'un déploiement.
- Correction d'un problème en raison duquel la corruption des données du gestionnaire de flux pouvait empêcher le démarrage du logiciel AWS IoT Greengrass Core.

- Correction d'un problème qui empêchait les nouveaux appareils clients de se connecter au cœur dans certains scénarios.
- Correction d'un problème en raison duquel les noms de flux du gestionnaire de flux ne pouvaient pas contenir `.log`.

1.11.5

Correctifs de bogues et améliorations :

- Améliorations des performances générales et correctifs de bogues.

1.11.4

Correctifs de bogues et améliorations :

- Correction d'un problème lié au gestionnaire de flux qui empêchait les mises à niveau vers le logiciel AWS IoT Greengrass Core v1.11.3. Si vous utilisez le gestionnaire de flux pour exporter des données vers le cloud, vous pouvez désormais utiliser une mise à jour OTA pour mettre à niveau une version v1.x antérieure du logiciel AWS IoT Greengrass Core vers la version v1.11.4.
- Améliorations des performances générales et correctifs de bogues.

1.11.3

Correctifs de bogues et améliorations :

- Correction d'un problème en raison duquel le logiciel AWS IoT Greengrass Core s'exécutant en un clin d'œil sur un appareil Ubuntu ne répondait plus après une coupure de courant soudaine de l'appareil.
- Correction d'un problème qui retardait la livraison des messages MQTT aux fonctions Lambda à longue durée de vie.
- Correction d'un problème en raison duquel les messages MQTT n'étaient pas envoyés correctement lorsque la `maxWorkItemCount` valeur était définie sur une valeur supérieure `1024` à.
- Correction d'un problème en raison duquel l'agent de mise à jour OTA ignorait la `KeepAlive` période MQTT spécifiée dans la `keepAlive` propriété dans `config.json`.
- Améliorations des performances générales et correctifs de bogues.

> **Important**
>
> Si vous utilisez le gestionnaire de flux pour exporter des données vers le cloud, n'effectuez pas de mise à niveau vers le logiciel AWS IoT Greengrass Core v1.11.3

à partir d'une version v1.x antérieure. Si vous activez le gestionnaire de flux pour la première fois, nous vous recommandons vivement d'installer d'abord la dernière version du logiciel AWS IoT Greengrass Core.

1.11.1

Correctifs de bogues et améliorations :

- Correction d'un problème qui entraînait une augmentation de l'utilisation de la mémoire par le gestionnaire de flux.
- Correction d'un problème en raison duquel le gestionnaire de flux réinitialisait le numéro de séquence du flux `0` si le périphérique principal de Greengrass était éteint pendant une période supérieure à la période spécifiée time-to-live (TTL) pour les données du flux.
- Correction d'un problème qui empêchait le gestionnaire de flux d'arrêter correctement les tentatives d'exportation de données vers leAWS Cloud.

1.11.0

Nouvelles fonctions :

- Un agent de télémétrie situé sur le cœur de Greengrass collecte les données de télémétrie locales et les publie sur. AWS Cloud Pour récupérer les données de télémétrie afin de les traiter ultérieurement, les clients peuvent créer une EventBridge règle Amazon et s'abonner à une cible. Pour plus d'informations, consultez la section Collecte des données de télémétrie relatives à l'état du système à partir des AWS IoT Greengrass principaux appareils.
- Une API HTTP locale renvoie un instantané de l'état actuel des processus de travail locaux lancés parAWS IoT Greengrass. Pour plus d'informations, consultez la section Appel de l'API de vérification de santé locale.
- Un gestionnaire de flux exporte automatiquement les données vers Amazon S3 etAWS IoT SiteWise.

  Les nouveaux paramètres du gestionnaire de flux vous permettent de mettre à jour les flux existants et de suspendre ou de reprendre l'exportation de données.
- Support pour l'exécution des fonctions Lambda de Python 3.8.x sur le noyau.
- Une nouvelle `ggDaemonPort` propriété utilisée pour configurer le numéro de port IPC principal de Greengrass. `config.json` Le numéro de port par défaut est 8000.

Une nouvelle `systemComponentAuthTimeout` propriété `config.json`que vous utilisez pour configurer le délai d'expiration pour l'authentification IPC principale de Greengrass. Le délai d'expiration par défaut est de 5 000 millisecondes.

- Le nombre maximum d'AWS IoTappareils par AWS IoT Greengrass groupe a été augmenté de 200 à 2 500.

  Le nombre maximum d'abonnements par groupe a été augmenté de 1 000 à 10 000.

  Pour de plus amples informations, consultez AWS IoT Greengrass Points de terminaison et quotas.

Correctifs de bogues et améliorations :

- Optimisation générale permettant de réduire l'utilisation de la mémoire par les processus du service Greengrass.
- Un nouveau paramètre de configuration d'exécution (`mountAllBlockDevices`) permet à Greengrass d'utiliser des montages par liaison pour monter tous les périphériques en mode bloc dans un conteneur après avoir configuré OverlayFS. Cette fonctionnalité a résolu un problème qui provoquait l'échec du déploiement de Greengrass s'il ne se `/usr` trouvait pas dans la `/` hiérarchie.
- Correction d'un problème qui provoquait une défaillance AWS IoT Greengrass de base s'il `/tmp` s'agissait d'un lien symbolique.
- Correction d'un problème qui permettait à l'agent de déploiement Greengrass de supprimer du dossier les artefacts de modèles d'apprentissage automatique non utilisés. `mlmodel_public`
- Améliorations des performances générales et correctifs de bogues.

Pour installer le logiciel AWS IoT Greengrass Core sur votre appareil principal, téléchargez le package correspondant à votre architecture et à votre système d'exploitation (OS), puis suivez les étapes du guide de démarrage.

> **Tip**
>
> AWS IoT Greengrass fournit également d'autres options pour installer le logiciel AWS IoT Greengrass Core. Par exemple, vous pouvez utiliser la configuration de l'appareil Greengrass pour configurer votre environnement et installer la dernière version du logiciel AWS IoT Greengrass Core. Ou, sur les plateformes Debian prises en charge,

vous pouvez utiliser le gestionnaire de paquets APT pour installer ou mettre à niveau le logiciel AWS IoT Greengrass Core. Pour plus d'informations, consultez the section called "Installer le logiciel AWS IoT Greengrass Core".

Architecture	Système d'exploitation	Lien
Armv8 (AArch64)	Linux	Download
Armv8 (AArch64)	Linux (OpenWrt)	Download
Armv7l	Linux	Download
Armv7l	Linux (OpenWrt)	Download
Armv6l	Linux	Download
x86_64	Linux	Download

Extended life versions

1,1,5

Nouvelles fonctions dans v1.10 :

- Un gestionnaire de flux qui traite les flux de données localement et les exporte AWS Cloud automatiquement vers le. Cette fonctionnalité nécessite Java 8 sur l'appareil central Greengrass. Pour plus d'informations, consultez Gestion des flux de données.
- Nouveau connecteur de déploiement d'application Greengrass Docker qui exécute une application Docker sur un appareil principal (noyau). Pour plus d'informations, consultez the section called "Déploiement d'applications Docker".
- Un nouveau SiteWise connecteur IoT qui envoie les données des appareils industriels depuis les serveurs OPC-UA vers les propriétés des actifs. AWS IoT SiteWise Pour plus d'informations, consultez the section called "IoT SiteWise".
- Les fonctions Lambda qui s'exécutent sans conteneurisation peuvent accéder aux ressources d'apprentissage automatique du groupe Greengrass. Pour plus d'informations, consultez the section called "Accès aux ressources de machine learning".

- Prise en charge des sessions persistantes MQTT avec AWS IoT. Pour plus d'informations, consultez the section called "Sessions persistantes MQTT avec AWS IoT Core".
- Le trafic MQTT local peut circuler sur un port autre que le port par défaut 8883. Pour plus d'informations, consultez the section called "Port MQTT pour la messagerie locale".
- Nouvelles `queueFullPolicy` options du SDK AWS IoT Greengrass principal pour une publication fiable des messages à partir des fonctions Lambda.
- Support pour l'exécution des fonctions Lambda de Node.js 12.x sur le noyau.

Correctifs de bogues et améliorations :

- Les mises à jour O ver-the-air (OTA) avec intégration de la sécurité matérielle peuvent être configurées avec OpenSSL 1.1.
- Le gestionnaire de flux est plus résilient à la corruption des données de fichier.
- Correction d'un problème provoquant une défaillance du montage sysfs sur les périphériques utilisant le noyau Linux 5.1 et versions ultérieures.
- Nouvelle `mqttOperationTimeout` propriété du fichier config.json que vous utilisez pour définir le délai d'expiration des opérations de publication, d'abonnement et de désabonnement dans les connexions MQTT avec. AWS IoT Core
- Correction d'un problème qui entraînait une augmentation de l'utilisation de la mémoire par le gestionnaire de flux.
- Une nouvelle `systemComponentAuthTimeout` propriété `config.json`que vous utilisez pour configurer le délai d'expiration pour l'authentification IPC principale de Greengrass. Le délai d'expiration par défaut est de 5 000 millisecondes.
- Correction d'un problème en raison duquel l'agent de mise à jour OTA ignorait la `KeepAlive` période MQTT spécifiée dans la `keepAlive` propriété dans `config.json`.
- Correction d'un problème en raison duquel les messages MQTT n'étaient pas envoyés correctement lorsque la `maxWorkItemCount` valeur était définie sur une valeur supérieure `1024` à.
- Correction d'un problème qui retardait la livraison des messages MQTT aux fonctions Lambda à longue durée de vie.
- Correction d'un problème en raison duquel le logiciel AWS IoT Greengrass Core s'exécutant en un clin d'œil sur un appareil Ubuntu ne répondait plus après une coupure de courant soudaine de l'appareil.
- Améliorations des performances générales et correctifs de bogues.

Pour installer le logiciel AWS IoT Greengrass Core sur votre appareil principal, téléchargez le package correspondant à votre architecture et à votre système d'exploitation (OS), puis suivez les étapes du guide de démarrage.

Architecture	Système d'exploitation	Lien
Armv8 (AArch64)	Linux	Download
Armv8 (AArch64)	Linux (OpenWrt)	Download
Armv7l	Linux	Download
Armv7l	Linux (OpenWrt)	Download
Armv6l	Linux	Download
x86_64	Linux	Download

1.9.4

Nouvelles fonctions dans v1.9 :

- Support pour les environnements d'exécution Lambda de Python 3.7 et Node.js 8.10. Les fonctions Lambda qui utilisent les environnements d'exécution Python 3.7 et Node.js 8.10 peuvent désormais être exécutées sur un cœur. AWS IoT Greengrass (AWS IoT Greengrasscontinue de prendre en charge les environnements d'exécution Python 2.7 et Node.js 6.10.)
- Connexions MQTT optimisées. Le noyau Greengrass établit moins de connexions avec le noyau AWS IoT Core. Cette modification peut réduire les coûts d'exploitation pour les frais qui sont calculés en fonction du nombre de connexions.
- Clé à courbe elliptique (EC) pour le serveur MQTT local. Le serveur MQTT local prend en charge les clés EC, en plus des clés RSA. (Le certificat du serveur MQTT présente une signature RSA SHA-256, quel que soit le type de clé.) Pour plus d'informations, consultez the section called “Mandataires de sécurité”.
- Support pour OpenWrt. AWS IoT Greengrass Le logiciel de base v1.9.2 ou version ultérieure peut être installé sur les OpenWrt distributions avec les architectures Armv8 (AArch64) et ARMv7L. Actuellement, OpenWrt ne prend pas en charge l'inférence ML.

- Support pour ARMv6L. AWS IoT Greengrass Le logiciel de base v1.9.3 ou version ultérieure peut être installé sur les distributions Raspbian sur les architectures ARMv6L (par exemple, sur les appareils Raspberry Pi Zero).
- Mises à jour OTA sur le port 443 avec ALPN. Les cœurs Greengrass qui utilisent le port 443 pour le trafic MQTT prennent désormais en charge les mises à jour over-the-air logicielles (OTA). AWS IoT Greengrassutilise l'extension TLS ALPN (Application Layer Protocol Network) pour activer ces connexions. Pour plus d'informations, consultez Mises à jour OTA du logiciel AWS IoT Greengrass Core et the section called "Connexion au port 443 ou via un proxy réseau".

Pour installer le logiciel AWS IoT Greengrass Core sur votre appareil principal, téléchargez le package correspondant à votre architecture et à votre système d'exploitation (OS), puis suivez les étapes du guide de démarrage.

Architecture	Système d'exploitation	Lien
Armv8 (AArch64)	Linux	Download
Armv8 (AArch64)	Linux (OpenWrt)	Download
Armv7l	Linux	Download
Armv7l	Linux (OpenWrt)	Download
Armv6l	Linux	Download
x86_64	Linux	Download

1.8.4

- Nouvelles fonctions :
  - Identité d'accès par défaut configurable pour les fonctions Lambda du groupe. Ce paramètre au niveau du groupe détermine les autorisations par défaut utilisées pour exécuter les fonctions Lambda. Vous pouvez définir l'ID d'utilisateur, l'ID de groupe ou les deux. Les fonctions Lambda individuelles peuvent remplacer l'identité d'accès par défaut de leur groupe. Pour plus d'informations, consultez the section called "Définition de l'identité d'accès par défaut pour les fonctions Lambda dans un groupe".

- Trafic HTTPS sur le port 443. La communication HTTPS peut être configurée pour acheminer le trafic sur le port 443 plutôt que sur le port 8443 par défaut. Cela complète la AWS IoT Greengrass prise en charge de l'extension TLS ALPN (Application Layer Protocol Network) et permet à tout le trafic de messagerie Greengrass (MQTT et HTTPS) d'utiliser le port 443. Pour plus d'informations, consultez the section called "Connexion au port 443 ou via un proxy réseau".
- ID de client nommés de manière prévisible pour les connexions AWS IoT. Cette modification permet la prise en charge des événements liés au AWS IoT cycle de vie AWS IoT Device Defender et vous permet de recevoir des notifications pour les événements de connexion, de déconnexion, d'abonnement et de désinscription. L'attribution de noms de manière prévisible facilite également la création d'une logique pour les ID de connexion (par exemple, pour créer des modèles de stratégie d'abonnement basés sur les attributs de certificat). Pour plus d'informations, consultez the section called "ID client pour les connexions MQTT avec AWS IoT".

Correctifs de bogues et améliorations :

- Résolution d'un problème lié à la synchronisation des shadows et à la reconnexion du gestionnaire de certificats d'appareil.
- Améliorations des performances générales et correctifs de bogues.

Pour installer le logiciel AWS IoT Greengrass Core sur votre appareil principal, téléchargez le package correspondant à votre architecture et à votre système d'exploitation (OS), puis suivez les étapes du guide de démarrage.

Architecture	Système d'exploitation	Lien
Armv8 (AArch64)	Linux	Download
Armv7l	Linux	Download
x86_64	Linux	Download

Si vous téléchargez ce logiciel, vous acceptez le contrat de licence du logiciel Greengrass Core.

Pour de plus amples informations sur les autres options d'installation du logiciel AWS IoT Greengrass Core sur votre appareil, veuillez consulter the section called "Installer le logiciel AWS IoT Greengrass Core".

Logiciel de composant enfichable AWS IoT Greengrass

AWS IoT Greengrasssnap 1.11.x vous permet d'exécuter une version limitée de AWS IoT Greengrass via des progiciels pratiques, ainsi que toutes les dépendances nécessaires, dans un environnement conteneurisé.

> **Note**
>
> Le AWS IoT Greengrass snap est disponible pour le logiciel AWS IoT Greengrass Core v1.11.x. AWS IoT Greengrassne fournit pas de snap pour la v1.10.x. Les versions non prises en charge ne reçoivent pas de corrections de bogues ni de mises à jour.
> Le AWS IoT Greengrass snap ne prend pas en charge les connecteurs et l'inférence par apprentissage automatique (ML).

Pour plus d’informations, consultez the section called “Exécuter AWS IoT Greengrass dans un snap”.

Logiciel AWS IoT Greengrass Docker

AWS fournit un Dockerfile et une image Docker qui simplifient l'exécution d’AWS IoT Greengrass dans un conteneur Docker.

Dockerfile

Dockerfiles contiennent le code source pour créer des images de conteneur AWS IoT Greengrass personnalisées. Les images peuvent être modifiées pour s'exécuter sur différentes architectures de plateformes ou pour réduire la taille de l'image. Pour obtenir des instructions détaillées, consultez le fichier README.

Téléchargez la version de votre logiciel AWS IoT Greengrass Core cible.

v1.11

- Dockerfile pour AWS IoT Greengrass v1.11.6.

Extended life versions

v1.10

Dockerfile pour v1.10.5. AWS IoT Greengrass

v1.9

Dockerfile pour AWS IoT Greengrass v1.9.4.

v1.8

Dockerfile pour AWS IoT Greengrass v1.8.1.

Image Docker

Les images Docker ont le logiciel AWS IoT Greengrass Core et les dépendances installés sur les images de base Amazon Linux 2 (x86_64) et Alpine Linux (x86_64, Armv7l ou AArch64). Vous pouvez utiliser des images prédéfinies pour commencer à utiliser AWS IoT Greengrass.

> **Important**
>
> Le 30 juin 2022, AWS IoT Greengrass fin de la maintenance des images Docker du logiciel AWS IoT Greengrass Core v1.x publiées sur Amazon Elastic Container Registry (Amazon ECR) et Docker Hub. Vous pouvez continuer à télécharger ces images Docker depuis Amazon ECR et Docker Hub jusqu'au 30 juin 2023, soit un an après la fin de la maintenance. Cependant, les images Docker du logiciel AWS IoT Greengrass Core v1.x ne reçoivent plus de correctifs de sécurité ni de corrections de bogues après la fin de la maintenance le 30 juin 2022. Si vous exécutez une charge de travail de production qui dépend de ces images Docker, nous vous recommandons de créer vos propres images Docker à l'aide des Dockerfiles fournis. AWS IoT Greengrass Pour plus d'informations, consultez AWS IoT Greengrass Version 1politique de maintenance.

Téléchargez une image prédéfinie depuis Docker Hub ou Amazon Elastic Container Registry (Amazon ECR).

- Pour Docker Hub, utilisez la balise de *`version`* pour télécharger une version spécifique de l'image Greengrass Docker. Pour trouver des balises pour toutes les images disponibles, consultez la page Tags (Balises) sur Docker Hub.
- Pour Amazon ECR, utilisez le `latest` tag pour télécharger la dernière version disponible de l'image Greengrass Docker. Pour plus d'informations sur la mise en vente des versions d'images disponibles et le téléchargement d'images depuis Amazon ECR, consultezExécution de AWS IoT Greengrass dans un conteneur Docker.

> **Warning**
>
> À partir de la version v1.11.6 du logiciel AWS IoT Greengrass Core, les images Greengrass Docker n'incluent plus Python 2.7, car Python 2.7 est arrivé end-of-life en 2020 et ne reçoit plus de mises à jour de sécurité. Si vous choisissez de mettre à jour ces images Docker, nous vous recommandons de vérifier que vos applications fonctionnent avec les nouvelles images Docker avant de déployer les mises à jour sur les appareils de production. Si vous avez besoin de Python 2.7 pour votre application qui utilise une image Greengrass Docker, vous pouvez modifier le Dockerfile Greengrass pour inclure Python 2.7 pour votre application.

AWS IoT Greengrassne fournit pas d'images Docker pour le logiciel AWS IoT Greengrass Core v1.11.1.

> **Note**
>
> Par défaut, les images `alpine-aarch64` et les `alpine-armv7l` ne peuvent s'exécuter que sur des hôtes ARM. Pour exécuter ces images sur un hôte x86, vous pouvez installer QEMU et monter les bibliothèques QEMU sur l'hôte. Par exemple :
>
> ```
> docker run --rm --privileged multiarch/qemu-user-static --reset -p yes
> ```

Kit SDK AWS IoT Greengrass Core

Les fonctions Lambda utilisent le SDK AWS IoT Greengrass Core pour interagir avec le AWS IoT Greengrass noyau localement. Cela permet aux fonctions Lambda déployées de :

- Échange de messages MQTT avec AWS IoT Core.
- Échangez des messages MQTT avec des connecteurs, des appareils clients et d'autres fonctions Lambda du groupe Greengrass.
- Interagissez avec le service shadow local.
- Appelez d'autres fonctions Lambda locales.
- Accédez aux ressources de secret.

- Interagissez avec le gestionnaire de flux.

Téléchargez le SDK AWS IoT Greengrass de base correspondant à votre langue ou à votre plateforme depuis GitHub.

- AWS IoT GreengrassSDK de base pour Java
- AWS IoT GreengrassSDK de base pour Node.js
- AWS IoT GreengrassSDK de base pour Python
- AWS IoT GreengrassSDK de base pour C

Pour plus d'informations, consultez Kits SDK AWS IoT Greengrass Core.

Bibliothèques et environnements d'exécution de machine learning pris en charge

Pour effectuer des inférences sur un noyau Greengrass, vous devez installer le moteur d'exécution ou la bibliothèque de machine learning pour votre type de modèle de machine learning.

AWS IoT Greengrass prend en charge les types de modèles de machine learning suivants. Utilisez ces liens pour trouver des informations sur l'installation du moteur d'exécution ou de la bibliothèque de votre type de modèle et de la plateforme de l'appareil.

- Deep Learning Runtime (DLR)
- MXNet
- TensorFlow

Exemples de machine learning

AWS IoT Greengrass fournit des exemples que vous pouvez utiliser avec les moteurs d'exécution et les bibliothèques de machine learning pris en charge. Ces exemples sont publiés sous le Contrat de licence du logiciel Greengrass Core.

Deep learning runtime (DLR)

Téléchargez l'exemple correspondant à la plateforme de votre appareil :

- Exemple DLR pour Raspberry Pi
- Exemple DLR pour NVIDIA Jetson TX2
- Exemple DLR pour Intel Atom

Pour obtenir un didacticiel qui utilise l'exemple DLR, veuillez consulter the section called “Configuration de l'inférence Machine Learning optimisée”.

MXNet

Téléchargez l'exemple correspondant à la plateforme de votre appareil :

- Exemple MXNet pour Raspberry Pi
- Exemple MXNet pour NVIDIA Jetson TX2
- Exemple MXNet pour Intel Atom

Pour obtenir un didacticiel qui utilise l'exemple MXNet, veuillez consulter the section called “Configuration de l'inférence Machine Learning”.

TensorFlow

Téléchargez l'exemple Tensorflow pour la plateforme de votre appareil. Cet exemple fonctionne avec Raspberry Pi, NVIDIA Jetson TX2 et Intel Atom.

Logiciel du kit SDK de machine learning AWS IoT Greengrass

AWS IoT GreengrassKit de développement logiciel (SDK) pour le Machine LearningCela permet aux fonctions Lambda que vous créez d'utiliser un modèle d'apprentissage automatique local et d'envoyer des données au connecteur ML Feedback pour le téléchargement et la publication.

v1.1.0

- Python 3.7.

v1.0.0

- Python 2.7.

Nous voulons entendre parler de vous

Nous apprécions vos commentaires. Pour nous contacter, rendez-vous sur AWSRe:post et utilisez le AWS IoT Greengrass tag.

Installer le logiciel AWS IoT Greengrass Core

Le logiciel AWS IoT Greengrass Core étend les AWS fonctionnalités à un appareil AWS IoT Greengrass principal, ce qui permet aux appareils locaux d'agir localement sur les données qu'ils génèrent.

AWS IoT Greengrass propose plusieurs options pour installer le logiciel AWS IoT Greengrass Core :

- Téléchargez et extrayez un fichier tar.gz.
- Exécutez le script de configuration de l'appareil Greengrass.
- Installez-le à partir d'un référentiel APT.

AWS IoT Greengrass fournit également des environnements conteneurisés qui exécutent le logiciel AWS IoT Greengrass Core.

- Exécutez AWS IoT Greengrass dans un conteneur Docker.
- Exécutez AWS IoT Greengrass dans un snap.

Télécharger et extraire le progiciel AWS IoT Greengrass Core

Choisissez le logiciel AWS IoT Greengrass Core pour votre plateforme, téléchargez-le sous forme de fichier tar.gz et procédez à son extraction sur votre appareil. Vous pouvez télécharger les versions récentes du logiciel. Pour plus d’informations, consultez the section called “Logiciel AWS IoT Greengrass Core”.

Exécuter le script de configuration de l'appareil Greengrass

Exécutez le programme de configuration de l'appareil Greengrass pour configurer votre appareil, installer la dernière version du logiciel AWS IoT Greengrass Core et déployer une fonction Hello

World Lambda en quelques minutes. Pour plus d'informations, consultez the section called "Démarrage rapide : Configuration de l'appareil Greengrass".

Installer le logiciel AWS IoT Greengrass Core à partir d'un référentiel APT

> **Important**
>
> Depuis le 11 février 2022, vous ne pouvez plus installer ou mettre à jour le logiciel AWS IoT Greengrass Core à partir d'un dépôt APT. Sur les appareils sur lesquels vous avez ajouté le AWS IoT Greengrass référentiel, vous devez le supprimer de la liste des sources. Les appareils qui exécutent le logiciel depuis le dépôt APT continueront à fonctionner normalement. Nous vous recommandons de mettre à jour le logiciel AWS IoT Greengrass Core à l'aide de fichiers tar.

Le référentiel APT fourni par AWS IoT Greengrass comprend les packages suivants :

- `aws-iot-greengrass-core`. Installe le logiciel AWS IoT Greengrass Core.
- `aws-iot-greengrass-keyring`. Installe les clés GnuPG (GPG) utilisées pour signer le dépôt de paquets. AWS IoT Greengrass

  Si vous téléchargez ce logiciel, vous acceptez le contrat de licence du logiciel Greengrass Core.

Rubriques

Utiliser des scripts systemd pour gérer le cycle de vie du démon Greengrass

Le package `aws-iot-greengrass-core` installe également des scripts `systemd` que vous pouvez utiliser pour gérer le cycle de vie du logiciel AWS IoT Greengrass Core (démon).

- Pour lancer le démon Greengrass pendant le démarrage :

```
systemctl enable greengrass.service
```

- Pour lancer le démon Greengrass :

```
systemctl start greengrass.service
```

- Pour arrêter le démon Greengrass.

```
systemctl stop greengrass.service
```

- Pour vérifier l'état du démon Greengrass :

```
systemctl status greengrass.service
```

Désinstallez le logiciel AWS IoT Greengrass principal à l'aide du référentiel APT

Lorsque vous désinstallez le logiciel AWS IoT Greengrass principal, vous pouvez choisir de conserver ou de supprimer les informations de configuration du logiciel AWS IoT Greengrass principal, telles que les certificats des appareils, les informations de groupe et les fichiers journaux.

Pour désinstaller le logiciel AWS IoT Greengrass principal et conserver les informations de configuration

- Exécutez la commande suivante pour supprimer les AWS IoT Greengrass principaux packages logiciels et conserver les informations de configuration dans le `/greengrass` dossier.

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

Pour désinstaller le logiciel AWS IoT Greengrass principal et supprimer les informations de configuration

1. Exécutez la commande suivante pour supprimer les AWS IoT Greengrass principaux packages logiciels et supprimer les informations de configuration du`/greengrass folder`.

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

2. Supprimez le référentiel logiciel AWS IoT Greengrass principal de votre liste de sources. Pour plus d'informations, consultez Supprimer les AWS IoT Greengrass principales sources du référentiel de logiciels.

Supprimer les AWS IoT Greengrass principales sources du référentiel de logiciels

Vous pouvez supprimer les sources du référentiel logiciel AWS IoT Greengrass principal lorsque vous n'avez plus besoin d'installer ou de mettre à jour le logiciel AWS IoT Greengrass principal depuis le référentiel APT. Après le 11 février 2022, vous devez supprimer le dépôt de votre liste de sources pour éviter toute erreur lors de l'exécution`apt update`.

Pour supprimer le dépôt APT de la liste des sources

- Exécutez les commandes suivantes pour supprimer le référentiel logiciel AWS IoT Greengrass principal de la liste des sources.

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

Exécuter AWS IoT Greengrass dans un conteneur Docker

AWS IoT Greengrass fournit un Dockerfile et des images Docker qui simplifient l'exécution du logiciel AWS IoT Greengrass Core dans un conteneur Docker. Pour plus d'informations, consultez the section called "Logiciel AWS IoT Greengrass Docker".

> **Note**
>
> Vous pouvez également exécuter une application Docker sur un appareil principal Greengrass. Pour ce faire, utilisez le connecteur de déploiement d'application Greengrass Docker.

Exécuter AWS IoT Greengrass dans un snap

AWS IoT Greengrasssnap 1.11.x vous permet d'exécuter une version limitée de AWS IoT Greengrass via des progiciels pratiques, ainsi que toutes les dépendances nécessaires, dans un environnement conteneurisé.

Le 31 décembre 2023, la maintenance de la version logicielle AWS IoT Greengrass principale 1.11.x Snap publiée sur snapcraft.io AWS IoT Greengrass prendra fin. Les appareils utilisant actuellement le Snap continueront de fonctionner jusqu'à nouvel ordre. Cependant, le Snap AWS IoT Greengrass principal ne recevra plus de correctifs de sécurité ni de corrections de bogues une fois la maintenance terminée.

Concepts Snap

Les concepts essentiels suivants vous aideront à comprendre comment utiliser le AWS IoT Greengrass snap :

Channel

Composant snap qui définit la version d'un snap qui est installée et dont les mises à jour sont suivies. Les snaps sont automatiquement mis à jour avec la dernière version de la chaîne actuelle.

Interface

Composant instantané qui autorise l'accès aux ressources, telles que les réseaux et les fichiers utilisateur.

Pour exécuter le AWS IoT Greengrass snap, les interfaces suivantes doivent être connectées. Notez que vous `greengrass-support-no-container` devez d'abord être connecté et ne jamais être déconnecté.

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

```
    - process-control
    - system-observe
```

Les autres interfaces sont facultatives. Si vos fonctions Lambda nécessitent l'accès à des ressources spécifiques, vous devrez peut-être vous connecter aux interfaces appropriées.

Actualiser

Les snaps sont automatiquement mis à jour. Le `snapd` daemon est le gestionnaire de paquets Snap qui vérifie les mises à jour quatre fois par jour par défaut. Chaque vérification de mise à jour est appelée actualisation. Lors d'une actualisation, le démon s'arrête, le snap est mis à jour, puis le démon redémarre.

Pour plus d'informations, consultez le site Web de Snapcraft.

Nouveautés de AWS IoT Greengrass Snap v1.11.x

Ce qui suit décrit les nouveautés et les modifications apportées par la version 1.11.x du AWS IoT Greengrass snap.

- Cette version ne prend en charge que l'`snap_daemon`utilisateur, exposé sous forme d'ID utilisateur (UID) et de groupe (GID). 584788
- Cette version ne prend en charge que les fonctions Lambda non conteneurisées.

> **Important**
>
> Comme les fonctions Lambda non conteneurisées doivent partager le même utilisateur `snap_daemon` (), les fonctions Lambda ne sont pas isolées les unes des autres. Pour plus d'informations, consultez la section Contrôle de l'exécution des fonctions Greengrass Lambda à l'aide d'une configuration spécifique au groupe.

- Cette version prend en charge les environnements d'exécution C, C++, Java 8, Node.js 12.x, Python 2.7, Python 3.7 et Python 3.8.

> **Note**
>
> Pour éviter les environnements d'exécution Python redondants, les fonctions Lambda de Python 3.7 exécutent en fait le moteur d'exécution Python 3.8.

Mise en route avec le snap AWS IoT Greengrass

La procédure suivante vous permet d'installer et de configurer le AWS IoT Greengrass snap sur votre appareil.

Prérequis

Pour exécuter le AWS IoT Greengrass snap, vous devez effectuer les opérations suivantes :

- Exécutez le AWS IoT Greengrass snap sur une distribution Linux compatible, telle qu'Ubuntu, Linux Mint, Debian et Fedora.
- Installez le `snapd` daemon sur votre appareil. Le `snapd` daemon, y compris l'`snap`outil, gère l'environnement Snap de votre appareil.

Pour obtenir la liste des distributions Linux prises en charge et les instructions d'installation, consultez la section Installation de snapd dans la documentation de Snap.

Installation et configuration du AWS IoT Greengrass snap

Le didacticiel suivant explique comment installer et configurer le AWS IoT Greengrass snap sur votre appareil.

> **Note**
>
> - Bien que ce didacticiel utilise une instance Amazon EC2 (x86 t2.micro Ubuntu 20.04), vous pouvez exécuter le AWS IoT Greengrass snap avec du matériel physique, tel qu'un Raspberry Pi.
> - Le `snapd` daemon est préinstallé sur Ubuntu.

1. Installez le `core18` snap en exécutant la commande suivante sur le terminal de votre appareil :

   ```
   sudo snap install core18
   ```

   Le `core18` snap est un snap de base qui fournit un environnement d'exécution avec des bibliothèques couramment utilisées. Ce snap est construit à partir d'Ubuntu 18.04 LTS.
2. Effectuez `snapd` la mise à niveau en exécutant la commande suivante :

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

3. Exécutez la `snap list` commande pour vérifier si le AWS IoT Greengrass snap est installé.

   L'exemple de réponse suivant montre que cela `snapd` est installé, mais ne l'`aws-iot-greengrass`est pas.

```
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. Choisissez l'une des options suivantes pour installer AWS IoT Greengrass snap 1.11.x.

   - Pour installer le AWS IoT Greengrass snap, exécutez la commande suivante :

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

     Exemple de réponse :

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

   - Pour migrer d'une version antérieure vers la version v1.11.x ou effectuer une mise à jour vers la dernière version de correctif disponible, exécutez la commande suivante :

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

   Comme les autres snaps, le AWS IoT Greengrass snap utilise des canaux pour gérer les versions mineures. Les snaps sont automatiquement mis à jour avec la dernière version disponible de la chaîne actuelle. Par exemple, si vous le spécifiez`--channel=1.11.x`, votre AWS IoT Greengrass snap est mis à jour vers la version v1.11.5.

   Vous pouvez exécuter la `snap info aws-iot-greengrass` commande pour obtenir la liste des chaînes disponibles pourAWS IoT Greengrass.

   Exemple de réponse :

```
name:      aws-iot-greengrass
summary:   AWS supported software that extends cloud capabilities to local devices.
publisher: Amazon Web Services (aws#)
store-url: https://snapcraft.io/aws-iot-greengrass
contact:   https://repost.aws/tags/TA4ckIed1sR4enZBey29rKTg/aws-io-t-greengrass
license:   Proprietary
description: |
  AWS IoT Greengrass seamlessly extends AWS onto edge devices so they can act
 locally on the data
  they generate, while still using the cloud for management, analytics, and durable
 storage.
  AWS IoT Greenrgrass snap v1.11.0 enables you to run a limited version of AWS IoT
 Greengrass with
  all necessary dependencies in a containerized environment.
  The AWS IoT Greengrass snap doesn't support connectors and machine learning (ML)
 inference.
  By downloading this software you agree to the Greengrass Core Software License
 Agreement
  (https://s3-us-west-2.amazonaws.com/greengrass-release-license/greengrass-
license-v1.pdf).
  For more information, see Run AWS IoT Greengrass in a snap
  (https://docs.aws.amazon.com/greengrass/latest/developerguide/install-
ggc.html#gg-snap-support) in
  the AWS IoT Greengrass Developer.
  If you need help, try the AWS IoT Greengrass tag on AWS re:Post
  (https://repost.aws/tags/TA4ckIed1sR4enZBey29rKTg/aws-io-t-greengrass) or connect
 with an AWS IQ expert
  (https://iq.aws.amazon.com/services/aws/greengrass).
snap-id: SRDuhPJGj4XPxFNNZQKOTvURAp0wxKnd
channels:
  latest/stable:    1.11.3 2021-06-15 (59) 111MB -
  latest/candidate: 1.11.3 2021-06-14 (59) 111MB -
  latest/beta:      1.11.3 2021-06-14 (59) 111MB -
  latest/edge:      1.11.3 2021-06-14 (59) 111MB -
  1.11.x/stable:    1.11.3 2021-06-15 (59) 111MB -
  1.11.x/candidate: 1.11.3 2021-06-15 (59) 111MB -
  1.11.x/beta:      1.11.3 2021-06-15 (59) 111MB -
  1.11.x/edge:      1.11.3 2021-06-15 (59) 111MB -
```

5. Pour accéder aux ressources spécifiques dont vos fonctions Lambda ont besoin, vous pouvez vous connecter à des interfaces supplémentaires.

Exécutez la commande suivante pour obtenir la liste des interfaces prises en charge par AWS IoT Greengrass Snap :

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

Exemple de réponse :

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

```
network-bind              aws-iot-greengrass:network-bind
  :network-bind         -
network-control           aws-iot-greengrass:network-control
  :network-control      -
opengl                    aws-iot-greengrass:opengl
  :opengl               -
optical-drive             aws-iot-greengrass:optical-drive
  :optical-drive        -
process-control           aws-iot-greengrass:process-control
  :process-control      -
raw-usb                   aws-iot-greengrass:raw-usb                          -
                -
removable-media           aws-iot-greengrass:removable-media                  -
                -
serial-port               aws-iot-greengrass:serial-port                      -
                -
spi                       aws-iot-greengrass:spi                              -
                -
system-observe            aws-iot-greengrass:system-observe
  :system-observe       -
```

Si vous voyez un tiret (-) dans la colonne Emplacement, cela signifie que l'interface correspondante n'est pas connectée.

6. Suivez Installer le logiciel AWS IoT Greengrass Core pour créer un AWS IoT objet, un groupe Greengrass, des ressources de sécurité permettant des communications sécurisées avec et le AWS IoT fichier de configuration du logiciel AWS IoT Greengrass Core. Le fichier de configuration contient une configuration spécifique à votre base Greengrass, telle que l'emplacement des fichiers de certificat et le point de terminaison des données de l'AWS IoTappareil. `config.json`

   > **Note**
   >
   > Si vous avez téléchargé le fichier sur un autre appareil, suivez cette étape pour transférer les fichiers vers le périphérique AWS IoT Greengrass principal.

7. Pour le AWS IoT Greengrass snap, assurez-vous de mettre à jour le fichier config.json, comme indiqué ci-dessous :

   - Remplacez chaque instance de *`CertificateID`* par l'ID du certificat dans le nom du certificat et des fichiers clés.

- Si vous avez téléchargé un certificat d'autorité de certification racine Amazon différent de celui d'Amazon Root CA 1, remplacez chaque instance de *AmazonRootCA1.pem* par le nom du fichier d'autorité de certification racine d'Amazon.

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

8. Exécutez la commande suivante pour ajouter votre AWS IoT Greengrass certificat et vos fichiers de configuration :

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

Déploiement d'une fonction Lambda

Cette section explique comment déployer une fonction Lambda gérée par le client en un clin d'œil. AWS IoT Greengrass

> **Important**
>
> AWS IoT Greengrasssnap v1.11 ne prend en charge que les fonctions Lambda non conteneurisées.

1. Exécutez la commande suivante pour démarrer le AWS IoT Greengrass daemon :

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

   Exemple de réponse :

   ```
   Started.
   ```

   > **Note**
   >
   > En cas d'erreur, vous pouvez utiliser la `snap run` commande pour afficher un message d'erreur détaillé. Pour plus d'informations sur la résolution des problèmes, consultezerreur : impossible d'effectuer les tâches suivantes : - Exécutez la commande de service « start » pour les services ["greengrassd"] de snap "aws-iot-greengrass" ([start snap. aws-iot-greengrass.greengrassd.service] a échoué avec le statut de sortie 1 : Job for snap. aws-iot-greengrass.greengrassd.service a échoué car le processus de contrôle s'est terminé avec un code d'erreur. Voir « systemctl status snap ». aws-iot-greengrass.greengrassd.service » et « journalctl -xe » pour plus de détails.).

2. Exécutez la commande suivante pour vérifier que le daemon est en cours d'exécution :

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

   Exemple de réponse :

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

3. Suivez le module 3 (partie 1) : Lambda fonctionne AWS IoT Greengrass pour créer et déployer une fonction Lambda Hello World. Toutefois, avant de déployer la fonction Lambda, passez à l'étape suivante.

4. Assurez-vous que votre fonction Lambda s'exécute en tant qu'`snap_daemon`utilisateur et en mode sans conteneur. Pour mettre à jour les paramètres de votre groupe Greengrass, procédez comme suit dans la AWS IoT Greengrass console :

   a. Connectez-vous à la AWS IoT Greengrass console.
   b. Dans le volet de navigation de la AWS IoT console, sous Gérer, développez les appareils Greengrass, puis choisissez Groups (V1).
   c. Sous Greengrass groups, choisissez le groupe cible.
   d. Sur la page de configuration du groupe, dans le volet de navigation, choisissez l'onglet Fonctions Lambda.
   e. Dans Environnement d'exécution de la fonction Lambda par défaut, choisissez Modifier, puis procédez comme suit :

      i. Pour Utilisateur et groupe système par défaut, choisissez Autre ID utilisateur/ID de groupe, puis entrez **584788** à la fois l'ID utilisateur du système (numéro) et l'ID du groupe système (numéro).
      ii. Pour la conteneurisation de la fonction Lambda par défaut, sélectionnez Aucun conteneur.
      iii. Choisissez Enregistrer.

Arrêter le AWS IoT Greengrass daemon

Vous pouvez utiliser la `snap stop` commande pour arrêter un service.

Pour arrêter le AWS IoT Greengrass démon, exécutez la commande suivante :

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

La commande devrait être renvoyée`Stopped.`.

Pour vérifier si vous avez réussi à arrêter le snap, exécutez la commande suivante :

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

Exemple de réponse :

```
Service                       Startup   Current   Notes
```

```
aws-iot-greengrass.greengrassd  disabled  inactive  -
```

Désinstaller le snap AWS IoT Greengrass

Pour désinstaller le AWS IoT Greengrass snap, exécutez la commande suivante :

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

Exemple de réponse :

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

Résolution des problèmes liés au AWS IoT Greengrass snap

Utilisez les informations suivantes pour résoudre les problèmes liés au AWS IoT Greengrass snap.

J'ai reçu une autorisation et j'ai refusé des erreurs

Solution : Les erreurs de refus d'autorisation sont souvent dues à des interfaces manquantes. Pour obtenir la liste des interfaces manquantes et des informations de dépannage détaillées, vous pouvez utiliser l'`snappy-debug`outil.

Exécutez la commande suivante pour installer l'outil.

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

Exemple de réponse :

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

Exécutez la `sudo snappy-debug` commande dans une session de terminal séparée. L'opération se poursuit jusqu'à ce qu'une erreur de refus d'autorisation se produise.

Par exemple, si votre fonction Lambda essaie de lire un fichier dans le `$HOME` répertoire, vous pouvez obtenir la réponse suivante :

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

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

Cet exemple montre que la création du `/home/ubuntu/my-file.txt` fichier a provoqué l'erreur d'autorisation. Il suggère également que vous ajoutiez home à`plugs`. Toutefois, cette suggestion n'est pas applicable. Les `home-for-hooks` prises `home-for-greengrassd` et ne disposent que d'un accès en lecture seule.

Pour plus d'informations, consultez The snappy-debug snap dans la documentation Snap.

erreur : impossible d'effectuer les tâches suivantes : - Exécutez la commande de service « start » pour les services ["greengrassd"] de snap "aws-iot-greengrass" ([start snap. aws-iot-greengrass.greengrassd.service] a échoué avec le statut de sortie 1 : Job for snap. aws-iot-greengrass.greengrassd.service a échoué car le processus de contrôle s'est terminé avec un code d'erreur. Voir « systemctl status snap ». aws-iot-greengrass.greengrassd.service » et « journalctl -xe » pour plus de détails.)

Solution : cette erreur peut s'afficher lorsque la `snap start aws-iot-greengrass` commande ne démarre pas le logiciel AWS IoT Greengrass Core.

Pour plus d'informations sur le dépannage, exécutez la commande suivante :

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

Exemple de réponse :

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

Cet exemple montre AWS IoT Greengrass que le `config.json` fichier est introuvable. Vous pouvez vérifier les fichiers de configuration et de certificat.

/var/snap/ aws-iot-greengrass /current/ ggc-write-directory /packages/1.11.5/rootfs/merged n'est pas un chemin absolu ou est un lien symbolique.

Solution : Le AWS IoT Greengrass snap ne prend en charge que les fonctions Lambda non conteneurisées. Assurez-vous d'exécuter vos fonctions Lambda en mode sans conteneur. Pour plus

d'informations, consultez la section Considérations relatives au choix de la conteneurisation des fonctions Lambda dans le Guide du développeur. AWS IoT Greengrass Version 1

Le démon snapd n'a pas pu redémarrer après l'exécution de la commande sudo snap refresh snapd.

Solution : Suivez les étapes 6 à 8 Installation et configuration du AWS IoT Greengrass snap pour ajouter le AWS IoT Greengrass certificat et les fichiers de configuration au AWS IoT Greengrass snap.

Archiver une installation du logiciel AWS IoT Greengrass Core

Lorsque vous effectuez une mise à niveau vers une nouvelle version du logiciel AWS IoT Greengrass Core, vous pouvez archiver la version actuellement installée. Vous conservez ainsi votre environnement d'installation actuel afin de pouvoir tester une nouvelle version logicielle sur le même matériel. Cela simplifie également la restauration de votre version archivée, si vous en avez besoin.

Pour archiver l'installation actuelle et installer une nouvelle version

1. Téléchargez le package d'installation du logiciel AWS IoT Greengrass Core vers lequel vous souhaitez effectuer la mise à niveau.
2. Copiez le package dans l'appareil principal de destination. Pour obtenir des instructions sur le transfert de fichiers, consultez cette étape.

   > **Note**
   >
   > Vous copierez ultérieurement vos certificats, vos clés et votre fichier de configuration actuels dans la nouvelle installation.

   Exécutez les commandes des étapes suivantes dans votre terminal principal.
3. Assurez-vous que le démon Greengrass est arrêté sur l'appareil principal.

   a. Pour vérifier si le démon est en cours d'exécution :

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

   Si la sortie contient une entrée `root` pour `/greengrass/ggc/packages/`*`ggc-version`*`/bin/daemon`, le démon est en cours d'exécution.

> **Note**
>
> Cette procédure suppose que le logiciel AWS IoT Greengrass Core soit installé dans le répertoire `/greengrass`.

b. Pour arrêter le démon :

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

4. Déplacez le répertoire racine Greengrass actuel vers un autre répertoire.

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

5. Décompressez le nouveau logiciel sur l'appareil principal. Remplacez les espaces réservés *os-architecture* et *version* dans la commande.

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

6. Copiez les certificats, les clés et le fichier de configuration archivés dans la nouvelle installation.

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

7. Lancez le démon :

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

Vous pouvez maintenant procéder au déploiement d'un groupe afin de tester la nouvelle installation. En cas d'échec, vous pouvez restaurer l'installation archivée.

Pour restaurer l'installation archivée

1. Arrêtez le démon.
2. Supprimez le nouveau répertoire `/greengrass`.
3. Replacez le répertoire `/greengrass_backup` dans `/greengrass`.
4. Lancez le démon.

Configuration de AWS IoT Greengrass Core

Un AWS IoT Greengrass cœur est un AWS IoT objet (appareil) qui agit comme un hub ou une passerelle dans les environnements périphériques. Comme les autres appareils AWS IoT, un noyau existe dans le registre, possède un shadow d'appareil et utilise un certificat d'appareil pour s'authentifier auprès d'AWS IoT Core et de AWS IoT Greengrass. L'appareil principal exécute le logiciel AWS IoT Greengrass Core, ce qui lui permet de gérer les processus locaux pour les groupes Greengrass, tels que la communication, la synchronisation de shadows et l'échange de jetons.

Le logiciel AWS IoT Greengrass Core fournit les fonctionnalités suivantes :

- Déploiement et exécution locale de connecteurs et de fonctions Lambda.
- Traitez les flux de données localement avec des exportations automatiques vers leAWS Cloud.
- Messagerie MQTT sur le réseau local entre les appareils, les connecteurs et les fonctions Lambda à l'aide d'abonnements gérés.
- Messagerie MQTT entre appareils, connecteurs AWS IoT et fonctions Lambda à l'aide d'abonnements gérés.
- Connexions sécurisées entre les appareils et AWS Cloud utilisation de l'authentification et de l'autorisation des appareils.
- Synchronisation cachée locale des appareils. Les ombres peuvent être configurées pour être synchronisées avecAWS Cloud.
- Accès contrôlé à l'appareil local et aux ressources de volume.
- Déploiement de modèles d'apprentissage automatique formés dans le cloud pour exécution de l'inférence locale.
- Détection automatique d'adresse IP qui permet aux appareils de détecter votre appareil Greengrass principal.
- Déploiement central de nouvelles configurations de groupe ou mises à jour. Une fois les données de configuration téléchargée, l'appareil principal redémarre automatiquement.
- Mises à jour logicielles sécurisées over-the-air (OTA) des fonctions Lambda définies par l'utilisateur.
- Stockage sécurisé et crypté des secrets locaux et accès contrôlé par des connecteurs et des fonctions Lambda.

Fichier de configuration de AWS IoT Greengrass Core

Le fichier de configuration du logiciel AWS IoT Greengrass Core est `config.json`. Il est placé dans le répertoire */greengrass-root*`/config`.

> **Note**
>
> *greengrass-root* indique le chemin d'installation du logiciel AWS IoT Greengrass Core sur votre appareil. Généralement, il s'agit du répertoire `/greengrass`.
> Si vous utilisez l'option de création de groupe par défaut depuis la AWS IoT Greengrass console, le `config.json` fichier est déployé sur le périphérique principal dans un état de fonctionnement.

Vous pouvez vérifier le contenu de ce fichier en exécutant la commande suivante :

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

Voici un exemple de fichier `config.json`. Il s'agit de la version générée lorsque vous créez le noyau depuis la 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"
}
```

Le fichier `config.json` prend en charge les propriétés suivantes :

coreThing

Champ	Description	Remarques
`caPath`	Chemin d'accès au certificat d'authentification racine AWS IoT relatif au répertoire */greengrass-root* `/certs`.	Pour une compatibilité descendante avec les versions antérieures à la version 1.7.0. Cette propriété est ignorée lorsque l'`crypto`objet est présent. **Note** Assurez-vous que vos points de terminaison correspondent à

Champ	Description	Remarques
		votre type de certificat.
`certPath`	Chemin d'accès au certificat de l'appareil principal relatif au répertoire */greengrass-root* `/certs`.	Pour une compatibilité descendante avec les versions antérieures à la version 1.7.0. Cette propriété est ignorée lorsque l'`crypto`objet est présent.
`keyPath`	Chemin d'accès à la clé privée principale relatif au répertoire */greengrass-root*`/certs`.	Pour une compatibilité descendante avec les versions antérieures à la version 1.7.0. Cette propriété est ignorée lorsque l'`crypto`objet est présent.
`thingArn`	Le nom de ressource Amazon (ARN) de l'AWS IoTobjet qui représente le périphérique AWS IoT Greengrass principal.	Trouvez l'ARN de votre cœur dans la AWS IoT Greengrass console, sous Cores, ou en exécutant la commande `aws greengrass get-core-definition-version` CLI.

Champ	Description	Remarques
`iotHost`	Votre point de terminaison AWS IoT.	Recherchez le point de terminaison dans la AWS IoT console sous Paramètres ou en exécutant la commande `aws iot describe-endpoint --endpoint-type iot:Data-ATS` CLI. Cette commande renvoie le point de terminaison ATS (Amazon Trust Services) . Pour de plus amples informations, veuillez consulter la documentation Authentification du serveur. **Note** Assurez-vous que vos points de terminaison correspondent à votre type de certificat. Assurez-vous que vos points de terminaison correspondent à vos Région AWS.

Champ	Description	Remarques
`ggHost`	Votre point de terminaison AWS IoT Greengrass.	Il s'agit de votre point de terminaison `iotHost` avec le préfixe de l'hôte remplacé par greengrass (par exemple, `greengrass-ats.iot.`*`region`*`.amazonaws.com`). Utilisez le même Région AWS que`iotHost`. **Note** Assurez-vous que vos points de terminaison correspondent à votre type de certificat. Assurez-vous que vos points de terminaison correspondent à vos Région AWS.
`iotMqttPort`	Facultatif. Numéro de port à utiliser pour la communication MQTT avec AWS IoT.	Les valeurs valides sont `8883` ou `443`. La valeur par défaut est `8883`. Pour plus d'informations, consultez Connexion au port 443 ou via un proxy réseau.

Champ	Description	Remarques
`iotHttpPort`	Facultatif. Numéro de port utilisé pour créer des connexions HTTPS vers AWS IoT.	Les valeurs valides sont `8443` ou `443`. La valeur par défaut est `8443`. Pour plus d'informations, consultez Connexion au port 443 ou via un proxy réseau.
`ggMqttPort`	Facultatif. Numéro de port à utiliser pour la communication MQTT sur le réseau local.	Les valeurs valides vont de `1024` à `65535`. La valeur par défaut est `8883`. Pour plus d'informations, consultez the section called "Port MQTT pour la messagerie locale".
`ggHttpPort`	Facultatif. Numéro de port utilisé pour créer des connexions HTTPS vers le service AWS IoT Greengrass.	Les valeurs valides sont `8443` ou `443`. La valeur par défaut est `8443`. Pour plus d'informations, consultez Connexion au port 443 ou via un proxy réseau.
`keepAlive`	Facultatif. Période `KeepAlive` MQTT, en secondes	La plage valide est comprise entre 30 et 1 200 secondes. La valeur par défaut est `600`.
`networkProxy`	Facultatif. Objet qui définit une serveur proxy auquel vous connecter.	Le serveur proxy peut être HTTP ou HTTPS. Pour plus d'informations, consultez Connexion au port 443 ou via un proxy réseau.

Champ	Description	Remarques
`mqttOperationTimeout`	Facultatif. Durée (en secondes) permettant au noyau Greengrass de terminer une opération de publication, d'abonnement ou de désabonnement dans des connexions MQTT vers AWS IoT Core.	La valeur par défaut est 5. La valeur minimale est 5.
`ggDaemonPort`	Facultatif. Le numéro de port IPC principal de Greengrass.	Cette propriété est disponible dans la AWS IoT Greengrass version v1.11.0 ou ultérieure. Les valeurs valides sont comprises entre 1024 et 65535. La valeur par défaut est 8000.
`systemComponentAuthTimeout`	Facultatif. Durée (en millisecondes) nécessaire pour permettre à l'IPC principal de Greengrass de terminer l'authentification.	Cette propriété est disponible dans la AWS IoT Greengrass version v1.11.0 ou ultérieure. Les valeurs valides sont comprises entre 500 et 5 000. La valeur par défaut est 5000.

runtime

Champ	Description	Remarques
`maxWorkItemCompter`	Facultatif. Nombre maximal d'éléments de travail que le démon Greengrass peut traiter simultanément. Les	La valeur par défaut est 1024. La valeur maximale est limitée par le matériel de votre appareil.

Champ	Description	Remarques
	éléments de travail qui dépassent cette limite sont ignorés. La file d'attente des éléments de travail est partagée par les composants du système, les fonctions Lambda définies par l'utilisateur et les connecteurs.	L'augmentation de cette valeur augmente la mémoire utilisée par AWS IoT Greengrass. Vous pouvez augmenter cette valeur si vous pensez que votre noyau va recevoir un trafic de messages MQTT important.
`maxConcurrentLimit`	Facultatif. Le nombre maximum de travailleurs Lambda non épinglés simultanés que le daemon Greengrass peut avoir. Vous pouvez spécifier un autre entier pour remplacer ce paramètre.	La valeur par défaut est 25. La valeur minimale est définie par`lruSize`.
`Taille du LRU`	Optional. Defines the minimum value for `maxConcurrentLimit` .	The default value is 25.
`mountAllBlockAppareils`	Optional. Enables AWS IoT Greengrass to use bind mounts to mount all block devices into a container after setting up the OverlayFS.	Cette propriété est disponible dans la AWS IoT Greengrass version v1.11.0 ou ultérieure. Les valeurs valides sont `yes` et no. La valeur par défaut est no. Définissez cette valeur sur `yes` si votre `/usr` répertoir e ne se trouve pas dans la `/` hiérarchie.

Champ	Description	Remarques
`postStartHealthCheckTimeout`	Optional. The time (in milliseconds) after starting that the Greengrass daemon waits for the health check to finish.	The default timeout is 30 seconds (30000 ms).
`cgroup`		
`Utiliser le système D`	Indicates whether your device uses `systemd`.	Valid values are `oui` or `non`. Run the `check_ggc_dependencies` script in Module 1 to see if your device uses `systemd`.

crypto

Le paramètre `crypto` contient les propriétés qui prennent en charge le stockage de la clé privée sur un module de sécurité matérielle (HSM) via PKCS #11 et le stockage secret local. Pour plus d'informations, consultez the section called "Mandataires de sécurité", the section called "Intégration de sécurité matérielle" et Déployer des secrets sur Core. Les configurations pour le stockage des clés privées sur les HSM ou dans le système de fichiers sont prises en charge.

Champ	Description	Remarques
`Trajectorat`	Chemin d'accès absolu au certificat d'autorité de certification racine AWS IoT.	Il doit s'agir de l'URI d'un fichier du formulaire : `file:///absolute/path/to/file` . **Note** Assurez-vous que vos points de terminaison

Champ	Description	Remarques
		correspondent à votre type de certificat.
`PKCS11`		
`Moteur OpenSSL`	Facultatif. Chemin d'accès absolu au fichier de moteur OpenSSL `.so` pour activer la prise en charge PKCS # 11 sur OpenSSL.	Doit être un chemin d'accès à un fichier sur le système de fichiers. Cette propriété est requise si vous utilisez l'agent de mise à jour Greengrass OTA avec sécurité matérielle. Pour plus d'informations, consultez the section called “Configurer les mises à jour OTA”.
`Fournisseur P11`	Chemin absolu vers la bibliothèque libdl-loadable d'implémentation PKCS # 11.	Doit être un chemin d'accès à un fichier sur le système de fichiers.
`Étiquette de machine à sous`	Étiquette d'emplacement qui est utilisée pour identifier le module matériel.	Doit se conformer aux spécifications d'étiquette PKCS # 11.
`slotUserPin`	Le code PIN utilisateur utilisé pour authentifier le noyau de Greengrass auprès du module.	Vous devez disposer d'autorisations suffisantes pour exécuter C_Sign avec les clés privées configurées.
`principals`		
`Certificat IoT`	The certificate and private key that the core uses to make requests to AWS IoT.	

Champ	Description	Remarques
`Certificat IoT. privateKeyPath`	Chemin d'accès à la clé privée principale.	Pour le stockage de système de fichiers, il doit s'agir de l'URI d'un fichier du formulaire : `file:///`*`absolute/path/to/file`* . Pour le stockage HSM, il doit s'agir du chemin d'accès RFC WF-7512 PKCS # 11 qui spécifie l'objet étiquette.
`Certificat IoT .CertificatePath`	Chemin d'accès absolu au certificat de votre noyau.	Il doit s'agir de l'URI d'un fichier du formulaire : `file:///`*`absolute/path/to/file`* .
`MQTT ServerCer tificate`	Facultatif. Clé privée que le noyau utilise en combinaison avec le certificat pour agir en tant que passerelle ou serveur MQTT.	

Champ	Description	Remarques
`MQTTServerCertificate . privateKeyPath`	Chemin d'accès à la clé privée du serveur MQTT local.	Utilisez cette valeur pour spécifier votre propre clé privée pour le serveur MQTT local. Pour le stockage de système de fichiers, il doit s'agir de l'URI d'un fichier du formulaire : `file:///`*`absolute/path/to/file`* . Pour le stockage HSM, il doit s'agir du chemin d'accès RFC WF-7512 PKCS # 11 qui spécifie l'objet étiquette. Si cette propriété n'est pas spécifiée, AWS IoT Greengrass effectue une rotation de la clé basée sur vos paramètres de rotation. Si cette propriété est spécifiée, le client est responsable de la rotation de la clé.
`SecretsManager`	The private key that secures the data key used for encryption. For more information, see Déployer des secrets sur Core.	

Champ	Description	Remarques
`SecretsManager .privateKeyPath`	Chemin d'accès à la clé privée du Secrets Manager local.	Seule une clé RSA est prise en charge. Pour le stockage de système de fichiers, il doit s'agir de l'URI d'un fichier du formulaire : `file:///`*`absolute/path/to/file`* . Pour le stockage HSM, il doit s'agir du chemin d'accès RFC WF-7512 PKCS # 11 qui spécifie l'objet étiquette. La clé privée doit être générée à l'aide du mécanisme de remplissage PKCS#1 v1.5.

Les propriétés de configuration suivantes sont également prises en charge :

Champ	Description	Remarques
`mqttMaxConnectionRetryInterval`	Facultatif. L'intervalle maximal (en secondes) entre les tentatives de connexion MQTT si la connexion est abandonnée.	Spécifiez cette valeur en tant qu'entier non signé. L'argument par défaut est `60`.
`managedRespawn`	Facultatif. Indique que l'agent OTA doit exécuter un code personnalisé avant une mise à jour.	Les valeurs valides sont `true` ou `false`. Pour plus d'informations, consultez Mises à jour OTA du logiciel AWS IoT Greengrass Core.
`writeDirectory`	Facultatif. Le répertoire d'écriture où sont AWS IoT	Pour plus d'informations, consultez Configuration d'un

Champ	Description	Remarques
	Greengrass créées toutes les ressources de lecture/écriture.	répertoire en écriture pour AWS IoT Greengrass.
pidFileDirectory	Facultatif AWS IoT Greengrassenregistre son identifiant de processus (PID) dans ce répertoire.	La valeur par défaut est `/var/run`.

Extended life versions

Les versions suivantes du logiciel AWS IoT Greengrass Core sont en phase de durée de vie prolongée. Ces informations sont incluses à titre de référence uniquement.

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"
  }
}
```

Le fichier `config.json` prend en charge les propriétés suivantes :

coreThing

Champ	Description	Remarques
`caPath`	Chemin d'accès au certificat d'authentification racine AWS IoT relatif au répertoire */greengrass-root* `/certs`.	Pour une compatibilité descendante avec les versions antérieures à la version 1.7.0. Cette propriété est ignorée lorsque l'`cryptoobjet` est présent. **Note** Assurez-vous que vos points de terminaison correspondent à votre type de certificat.
`certPath`	Chemin d'accès au certificat de l'appareil principal relatif au répertoire */greengrass-root* `/certs`.	Pour une compatibilité descendante avec les versions antérieures à la version 1.7.0. Cette propriété est ignorée

Champ	Description	Remarques
		lorsque l'`cryptoobjet` est présent.
`keyPath`	Chemin d'accès à la clé privée principale relatif au répertoire */greengrass-root*`/certs`.	Pour une compatibilité descendante avec les versions antérieures à la version 1.7.0. Cette propriété est ignorée lorsque l'`cryptoobjet` est présent.
`thingArn`	Le nom de ressource Amazon (ARN) de l'AWS IoTobjet qui représente le périphérique AWS IoT Greengrass principal.	Trouvez l'ARN de votre cœur dans la AWS IoT Greengrass console, sous Cores, ou en exécutant la commande `aws greengrass get-core-definition-version` CLI.

Champ	Description	Remarques
`iotHost`	Votre point de terminaison AWS IoT.	Recherchez le point de terminaison dans la AWS IoT console sous Paramètres ou en exécutant la commande `aws iot describe-endpoint --endpoint-type iot:Data-ATS` CLI. Cette commande renvoie le point de terminaison ATS (Amazon Trust Services) . Pour de plus amples informations, veuillez consulter la documentation Authentification du serveur. **Note** Assurez-vous que vos points de terminaison correspondent à votre type de certificat. Assurez-vous que vos points de terminaison correspondent à vos Région AWS.

Champ	Description	Remarques
`ggHost`	Votre point de terminaison AWS IoT Greengrass.	Il s'agit de votre point de terminaison `iotHost` avec le préfixe de l'hôte remplacé par greengrass (par exemple, `greengrass-ats.iot.` *`region`*`.amazonaws.com`). Utilisez le même Région AWS que`iotHost`. **Note** Assurez-vous que vos points de terminaison correspondent à votre type de certificat. Assurez-vous que vos points de terminaison correspondent à vos Région AWS.
`iotMqttPort`	Facultatif. Numéro de port à utiliser pour la communication MQTT avec AWS IoT.	Les valeurs valides sont `8883` ou `443`. La valeur par défaut est `8883`. Pour plus d'informations, consultez Connexion au port 443 ou via un proxy réseau.

Champ	Description	Remarques
`iotHttpPort`	Facultatif. Numéro de port utilisé pour créer des connexions HTTPS vers AWS IoT.	Les valeurs valides sont `8443` ou `443`. La valeur par défaut est `8443`. Pour plus d'informations, consultez Connexion au port 443 ou via un proxy réseau.
`ggMqttPort`	Facultatif. Numéro de port à utiliser pour la communication MQTT sur le réseau local.	Les valeurs valides vont de `1024` à `65535`. La valeur par défaut est `8883`. Pour plus d'informations, consultez the section called "Port MQTT pour la messagerie locale".
`ggHttpPort`	Facultatif. Numéro de port utilisé pour créer des connexions HTTPS vers le service AWS IoT Greengrass.	Les valeurs valides sont `8443` ou `443`. La valeur par défaut est `8443`. Pour plus d'informations, consultez Connexion au port 443 ou via un proxy réseau.
`keepAlive`	Facultatif. Période `KeepAlive` MQTT, en secondes	La plage valide est comprise entre 30 et 1 200 secondes. La valeur par défaut est `600`.
`networkProxy`	Facultatif. Objet qui définit une serveur proxy auquel vous connecter.	Le serveur proxy peut être HTTP ou HTTPS. Pour plus d'informations, consultez Connexion au port 443 ou via un proxy réseau.

Champ	Description	Remarques
`mqttOperationTimeout`	Facultatif. Durée (en secondes) permettant au noyau Greengrass de terminer une opération de publication, d'abonnement ou de désabonnement dans des connexions MQTT vers AWS IoT Core.	Cette propriété est disponible à partir de AWS IoT Greengrass v1.10.2. La valeur par défaut est 5. La valeur minimale est 5.

runtime

Champ	Description	Remarques
`maxWorkItemCompter`	Facultatif. Nombre maximal d'éléments de travail que le démon Greengrass peut traiter simultanément. Les éléments de travail qui dépassent cette limite sont ignorés. La file d'attente des éléments de travail est partagée par les composants du système, les fonctions Lambda définies par l'utilisateur et les connecteurs.	La valeur par défaut est 1024. La valeur maximale est limitée par le matériel de votre appareil. L'augmentation de cette valeur augmente la mémoire utilisée par AWS IoT Greengrass. Vous pouvez augmenter cette valeur si vous pensez que votre noyau va recevoir un trafic de messages MQTT important.
`maxConcurrentLimit`	Facultatif. Le nombre maximum de travailleurs Lambda non épinglés simultanés que le daemon Greengrass peut avoir. Vous pouvez spécifier un autre	La valeur par défaut est 25. La valeur minimale est définie par`lruSize`.

Champ	Description	Remarques
	entier pour remplacer ce paramètre.	
`Taille du LRU`	Optional. Defines the minimum value for `maxConcurrentLimit` .	The default value is 25.
`postStartHealthChe ckTimeout`	Optional. The time (in milliseconds) after starting that the Greengrass daemon waits for the health check to finish.	The default timeout is 30 seconds (30000 ms).
`cgroup`		
`Utiliser le système D`	Indicates whether your device uses `systemd`.	Valid values are `oui` or `non`. Run the `check_ggc _dependencies` script in Module 1 to see if your device uses `systemd`.

crypto

Le paramètre `crypto` contient les propriétés qui prennent en charge le stockage de la clé privée sur un module de sécurité matérielle (HSM) via PKCS #11 et le stockage secret local. Pour plus d'informations, consultez the section called “Mandataires de sécurité”, the section called “Intégration de sécurité matérielle” et Déployer des secrets sur Core. Les configurations pour le stockage des clés privées sur les HSM ou dans le système de fichiers sont prises en charge.

Champ	Description	Remarques
`Trajectorat`	Chemin d'accès absolu au certificat d'autorité de certification racine AWS IoT.	Il doit s'agir de l'URI d'un fichier du formulaire : `file:///absolute/ path/to/file` .

Champ	Description	Remarques
		Note Assurez-vous que vos points de terminaison correspondent à votre type de certificat.
`PKCS11`		
`Moteur OpenSSL`	Facultatif. Chemin d'accès absolu au fichier de moteur OpenSSL `.so` pour activer la prise en charge PKCS # 11 sur OpenSSL.	Doit être un chemin d'accès à un fichier sur le système de fichiers. Cette propriété est requise si vous utilisez l'agent de mise à jour Greengrass OTA avec sécurité matérielle. Pour plus d'informations, consultez the section called "Configurer les mises à jour OTA".
`Fournisseur P11`	Chemin absolu vers la bibliothèque libdl-loadable d'implémentation PKCS # 11.	Doit être un chemin d'accès à un fichier sur le système de fichiers.
`Étiquette de machine à sous`	Étiquette d'emplacement qui est utilisée pour identifier le module matériel.	Doit se conformer aux spécifications d'étiquette PKCS # 11.

Champ	Description	Remarques
`slotUserPin`	Le code PIN utilisateur utilisé pour authentifier le noyau de Greengrass auprès du module.	Vous devez disposer d'autorisations suffisantes pour exécuter C_Sign avec les clés privées configurées.
`principals`		
`Certificat IoT`	The certificate and private key that the core uses to make requests to AWS IoT.	
`Certificat IoT. privateKeyPath`	Chemin d'accès à la clé privée principale.	Pour le stockage de système de fichiers, il doit s'agir de l'URI d'un fichier du formulaire : `file:///`*`absolute/path/to/file`* . Pour le stockage HSM, il doit s'agir du chemin d'accès RFC WF-7512 PKCS # 11 qui spécifie l'objet étiquette.
`Certificat IoT .CertificatePath`	Chemin d'accès absolu au certificat de votre noyau.	Il doit s'agir de l'URI d'un fichier du formulaire : `file:///`*`absolute/path/to/file`* .
`MQTT ServerCer tificate`	Facultatif. Clé privée que le noyau utilise en combinaison avec le certificat pour agir en tant que passerelle ou serveur MQTT.	

Champ	Description	Remarques
`MQTTServerCertificate . privateKeyPath`	Chemin d'accès à la clé privée du serveur MQTT local.	Utilisez cette valeur pour spécifier votre propre clé privée pour le serveur MQTT local. Pour le stockage de système de fichiers, il doit s'agir de l'URI d'un fichier du formulaire : `file:///`*`absolute/path/to/file`* . Pour le stockage HSM, il doit s'agir du chemin d'accès RFC WF-7512 PKCS # 11 qui spécifie l'objet étiquette. Si cette propriété n'est pas spécifiée, AWS IoT Greengrass effectue une rotation de la clé basée sur vos paramètres de rotation. Si cette propriété est spécifiée, le client est responsable de la rotation de la clé.
`SecretsManager`	The private key that secures the data key used for encryption. For more information, see Déployer des secrets sur Core.	

Champ	Description	Remarques
`SecretsManager .privateKeyPath`	Chemin d'accès à la clé privée du Secrets Manager local.	Seule une clé RSA est prise en charge. Pour le stockage de système de fichiers, il doit s'agir de l'URI d'un fichier du formulaire : `file:///`*`absolute/path/to/file`* . Pour le stockage HSM, il doit s'agir du chemin d'accès RFC WF-7512 PKCS # 11 qui spécifie l'objet étiquette. La clé privée doit être générée à l'aide du mécanisme de remplissage PKCS#1 v1.5.

Les propriétés de configuration suivantes sont également prises en charge :

Champ	Description	Remarques
`mqttMaxConnectionR etryInterval`	Facultatif. L'intervalle maximal (en secondes) entre les tentatives de connexion MQTT si la connexion est abandonnée.	Spécifiez cette valeur en tant qu'entier non signé. L'argument par défaut est `60`.
`managedRespawn`	Facultatif. Indique que l'agent OTA doit exécuter un code personnalisé avant une mise à jour.	Les valeurs valides sont `true` ou `false`. Pour plus d'informations, consultez Mises à jour OTA du logiciel AWS IoT Greengrass Core.

Champ	Description	Remarques
`writeDirectory`	Facultatif. Le répertoire d'écriture où sont AWS IoT Greengrass créées toutes les ressources de lecture/écriture.	Pour plus d'informations, consultez Configuration d'un répertoire en écriture pour 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"
  }
}
```

Le fichier `config.json` prend en charge les propriétés suivantes :

coreThing

Champ	Description	Remarques
`caPath`	Chemin d'accès au certificat d'authentification racine AWS IoT relatif au répertoire `/greengrass-root /certs`.	Pour une compatibilité descendante avec les versions antérieures à la version 1.7.0. Cette propriété est ignorée lorsque l'`cryptoobjet` est présent. **Note** Assurez-vous que vos points de terminaison correspondent à votre type de certificat.
`certPath`	Chemin d'accès au certificat de l'appareil principal relatif au répertoire `/greengrass-root /certs`.	Pour une compatibilité descendante avec les versions antérieures à la version 1.7.0. Cette propriété est ignorée lorsque l'`cryptoobjet` est présent.
`keyPath`	Chemin d'accès à la clé privée principale relatif au répertoire `/greengrass-root/certs`.	Pour une compatibilité descendante avec les versions antérieures à la version 1.7.0. Cette propriété est ignorée lorsque l'`cryptoobjet` est présent.

Champ	Description	Remarques
`thingArn`	Le nom de ressource Amazon (ARN) de l'AWS IoTobjet qui représente le périphérique AWS IoT Greengrass principal.	Trouvez l'ARN de votre cœur dans la AWS IoT Greengrass console, sous Cores, ou en exécutant la commande `aws greengrass get-core-definition-version` CLI.

Champ	Description	Remarques
`iotHost`	Votre point de terminaison AWS IoT.	Recherchez le point de terminaison dans la AWS IoT console sous Paramètres ou en exécutant la commande `aws iot describe-endpoint --endpoint-type iot:Data-ATS` CLI. Cette commande renvoie le point de terminaison ATS (Amazon Trust Services) . Pour de plus amples informations, veuillez consulter la documentation Authentification du serveur. **Note** Assurez-vous que vos points de terminaison correspondent à votre type de certificat. Assurez-vous que vos points de terminaison correspondent à vos Région AWS.

Champ	Description	Remarques
`ggHost`	Votre point de terminaison AWS IoT Greengrass.	Il s'agit de votre point de terminaison `iotHost` avec le préfixe de l'hôte remplacé par greengrass (par exemple, `greengrass-ats.iot.` *`region`*`.amazonaws.com`). Utilisez le même Région AWS que `iotHost`. **Note** Assurez-vous que vos points de terminaison correspondent à votre type de certificat. Assurez-vous que vos points de terminaison correspondent à vos Région AWS.
`iotMqttPort`	Facultatif. Numéro de port à utiliser pour la communication MQTT avec AWS IoT.	Les valeurs valides sont `8883` ou `443`. La valeur par défaut est `8883`. Pour plus d'informations, consultez Connexion au port 443 ou via un proxy réseau.

Champ	Description	Remarques
`iotHttpPort`	Facultatif. Numéro de port utilisé pour créer des connexions HTTPS vers AWS IoT.	Les valeurs valides sont `8443` ou `443`. La valeur par défaut est `8443`. Pour plus d'informations, consultez Connexion au port 443 ou via un proxy réseau.
`ggHttpPort`	Facultatif. Numéro de port utilisé pour créer des connexions HTTPS vers le service AWS IoT Greengras s.	Les valeurs valides sont `8443` ou `443`. La valeur par défaut est `8443`. Pour plus d'informations, consultez Connexion au port 443 ou via un proxy réseau.
`keepAlive`	Facultatif. Période `KeepAlive` MQTT, en secondes	La plage valide est comprise entre 30 et 1 200 secondes. La valeur par défaut est `600`.
`networkProxy`	Facultatif. Objet qui définit une serveur proxy auquel vous connecter.	Le serveur proxy peut être HTTP ou HTTPS. Pour plus d'informations, consultez Connexion au port 443 ou via un proxy réseau.

runtime

Champ	Description	Remarques
`maxConcurrentLimit`	Facultatif. Le nombre maximum de travailleurs Lambda non épinglés simultanés que le daemon Greengrass peut avoir. Vous pouvez spécifier un autre	La valeur par défaut est 25. La valeur minimale est définie par`lruSize`.

Champ	Description	Remarques
	entier pour remplacer ce paramètre.	
`Taille du LRU`	Optional. Defines the minimum value for `maxConcurrentLimit` .	The default value is 25.
`postStartHealthChe ckTimeout`	Optional. The time (in milliseconds) after starting that the Greengrass daemon waits for the health check to finish.	The default timeout is 30 seconds (30000 ms).
`cgroup`		
`Utiliser le système D`	Indicates whether your device uses `systemd`.	Valid values are `oui` or `non`. Run the `check_ggc _dependencies` script in Module 1 to see if your device uses `systemd`.

crypto

L'objet `crypto` est ajouté dans v1.7.0. Il présente les propriétés qui prennent en charge le stockage de la clé privée sur un module de sécurité matérielle (HSM) via PKCS # 11 et le stockage secret local. Pour plus d'informations, consultez the section called "Mandataires de sécurité", the section called "Intégration de sécurité matérielle" et Déployer des secrets sur Core. Les configurations pour le stockage des clés privées sur les HSM ou dans le système de fichiers sont prises en charge.

Champ	Description	Remarques
`Trajectorat`	Chemin d'accès absolu au certificat d'autorité de certification racine AWS IoT.	Il doit s'agir de l'URI d'un fichier du formulaire : `file:///absolute/ path/to/file` .

Champ	Description	Remarques
		Note Assurez-vous que vos points de terminaison correspondent à votre type de certificat.
`PKCS11`		
`Moteur OpenSSL`	Facultatif. Chemin d'accès absolu au fichier de moteur OpenSSL `.so` pour activer la prise en charge PKCS # 11 sur OpenSSL.	Doit être un chemin d'accès à un fichier sur le système de fichiers. Cette propriété est requise si vous utilisez l'agent de mise à jour Greengrass OTA avec sécurité matérielle. Pour plus d'informations, consultez the section called "Configurer les mises à jour OTA".
`Fournisseur P11`	Chemin absolu vers la bibliothèque libdl-loadable d'implémentation PKCS # 11.	Doit être un chemin d'accès à un fichier sur le système de fichiers.
`Étiquette de machine à sous`	Étiquette d'emplacement qui est utilisée pour identifier le module matériel.	Doit se conformer aux spécifications d'étiquette PKCS # 11.

Champ	Description	Remarques
`slotUserPin`	Le code PIN utilisateur utilisé pour authentifier le noyau de Greengrass auprès du module.	Vous devez disposer d'autorisations suffisantes pour exécuter C_Sign avec les clés privées configurées.
`principals`		
`Certificat IoT`	The certificate and private key that the core uses to make requests to AWS IoT.	
`Certificat IoT. privateKeyPath`	Chemin d'accès à la clé privée principale.	Pour le stockage de système de fichiers, il doit s'agir de l'URI d'un fichier du formulaire : `file:///`*`absolute/path/to/file`* . Pour le stockage HSM, il doit s'agir du chemin d'accès RFC WF-7512 PKCS # 11 qui spécifie l'objet étiquette.
`Certificat IoT .CertificatePath`	Chemin d'accès absolu au certificat de votre noyau.	Il doit s'agir de l'URI d'un fichier du formulaire : `file:///`*`absolute/path/to/file`* .
`MQTT ServerCertificate`	Facultatif. Clé privée que le noyau utilise en combinaison avec le certificat pour agir en tant que passerelle ou serveur MQTT.	

Champ	Description	Remarques
`MQTTServerCertificate . privateKeyPath`	Chemin d'accès à la clé privée du serveur MQTT local.	Utilisez cette valeur pour spécifier votre propre clé privée pour le serveur MQTT local. Pour le stockage de système de fichiers, il doit s'agir de l'URI d'un fichier du formulaire : `file:///`*`absolute/path/to/file`* . Pour le stockage HSM, il doit s'agir du chemin d'accès RFC WF-7512 PKCS # 11 qui spécifie l'objet étiquette. Si cette propriété n'est pas spécifiée, AWS IoT Greengrass effectue une rotation de la clé basée sur vos paramètres de rotation. Si cette propriété est spécifiée, le client est responsable de la rotation de la clé.
`SecretsManager`	The private key that secures the data key used for encryption. For more information, see Déployer des secrets sur Core.	

Champ	Description	Remarques
`SecretsManager .privateKeyPath`	Chemin d'accès à la clé privée du Secrets Manager local.	Seule une clé RSA est prise en charge. Pour le stockage de système de fichiers, il doit s'agir de l'URI d'un fichier du formulaire : `file:///`*`absolute/path/to/file`* . Pour le stockage HSM, il doit s'agir du chemin d'accès RFC WF-7512 PKCS # 11 qui spécifie l'objet étiquette. La clé privée doit être générée à l'aide du mécanisme de remplissage PKCS#1 v1.5.

Les propriétés de configuration suivantes sont également prises en charge.

Champ	Description	Remarques
`mqttMaxConnectionR etryInterval`	Facultatif. L'intervalle maximal (en secondes) entre les tentatives de connexion MQTT si la connexion est abandonnée.	Spécifiez cette valeur en tant qu'entier non signé. L'argument par défaut est `60`.
`managedRespawn`	Facultatif. Indique que l'agent OTA doit exécuter un code personnalisé avant une mise à jour.	Les valeurs valides sont `true` ou `false`. Pour plus d'informations, consultez Mises à jour OTA du logiciel AWS IoT Greengrass Core.

Champ	Description	Remarques
`writeDirectory`	Facultatif. Le répertoire d'écriture où sont AWS IoT Greengrass créées toutes les ressources de lecture/écriture.	Pour plus d'informations, consultez Configuration d'un répertoire en écriture pour 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"
  }
}
```

Le `config.json` fichier prend en charge les propriétés suivantes.

coreThing

Champ	Description	Remarques
`caPath`	Chemin d'accès au certificat d'authentification racine AWS IoT relatif au répertoire `/greengrass-root /certs`.	Pour une compatibilité descendante avec les versions antérieures à la version 1.7.0. Cette propriété est ignorée lorsque l'`crypto`objet est présent. **Note** Assurez-vous que vos points de terminaison correspondent à votre type de certificat.
`certPath`	Chemin d'accès au certificat de l'appareil principal relatif au répertoire `/greengrass-root /certs`.	Pour une compatibilité descendante avec les versions antérieures à la version 1.7.0. Cette propriété est ignorée lorsque l'`crypto`objet est présent.
`keyPath`	Chemin d'accès à la clé privée principale relatif au répertoire `/greengrass-root/certs`.	Pour une compatibilité descendante avec les versions antérieures à la version 1.7.0. Cette propriété est ignorée lorsque l'`crypto`objet est présent.

Champ	Description	Remarques
`thingArn`	Le nom de ressource Amazon (ARN) de l'AWS IoTobjet qui représente le périphérique AWS IoT Greengrass principal.	Trouvez l'ARN de votre cœur dans la AWS IoT Greengrass console, sous Cores, ou en exécutant la commande `aws greengrass get-core-definition-version` CLI.

Champ	Description	Remarques
`iotHost`	Votre point de terminaison AWS IoT.	Recherchez le point de terminaison dans la AWS IoT console sous Paramètres ou en exécutant la commande `aws iot describe-endpoint --endpoint-type iot:Data-ATS` CLI. Cette commande renvoie le point de terminaison ATS (Amazon Trust Services) . Pour de plus amples informations, veuillez consulter la documentation Authentification du serveur. **Note** Assurez-vous que vos points de terminaison correspondent à votre type de certificat. Assurez-vous que vos points de terminaison correspondent à vos Région AWS.

Champ	Description	Remarques
`ggHost`	Votre point de terminaison AWS IoT Greengrass.	Il s'agit de votre point de terminaison `iotHost` avec le préfixe de l'hôte remplacé par greengrass (par exemple, `greengrass-ats.iot.`*`region`*`.amazonaws.com`). Utilisez le même Région AWS que`iotHost`. **Note** Assurez-vous que vos points de terminaison correspondent à votre type de certificat. Assurez-vous que vos points de terminaison correspondent à vos Région AWS.
`iotMqttPort`	Facultatif. Numéro de port à utiliser pour la communication MQTT avec AWS IoT.	Les valeurs valides sont `8883` ou `443`. La valeur par défaut est `8883`. Pour plus d'informations, consultez Connexion au port 443 ou via un proxy réseau.

Champ	Description	Remarques
`iotHttpPort`	Facultatif. Numéro de port utilisé pour créer des connexions HTTPS vers AWS IoT.	Les valeurs valides sont `8443` ou `443`. La valeur par défaut est `8443`. Pour plus d'informations, consultez Connexion au port 443 ou via un proxy réseau.
`ggHttpPort`	Facultatif. Numéro de port utilisé pour créer des connexions HTTPS vers le service AWS IoT Greengras s.	Les valeurs valides sont `8443` ou `443`. La valeur par défaut est `8443`. Pour plus d'informations, consultez Connexion au port 443 ou via un proxy réseau.
`keepAlive`	Facultatif. Période `KeepAlive` MQTT, en secondes	La plage valide est comprise entre 30 et 1 200 secondes. La valeur par défaut est `600`.
`networkProxy`	Facultatif. Objet qui définit une serveur proxy auquel vous connecter.	Le serveur proxy peut être HTTP ou HTTPS. Pour plus d'informations, consultez Connexion au port 443 ou via un proxy réseau.

runtime

Champ	Description	Remarques
`cgroup`		
`Utiliser le système D`	Indicates whether your device uses `systemd`.	Valid values are `oui` or `non`. Run the `check_ggc _dependencies` script

Champ	Description	Remarques
		in Module 1 to see if your device uses `systemd`.

crypto

L'objet `crypto` est ajouté dans v1.7.0. Il présente les propriétés qui prennent en charge le stockage de la clé privée sur un module de sécurité matérielle (HSM) via PKCS # 11 et le stockage secret local. Pour plus d'informations, consultez the section called "Mandataires de sécurité", the section called "Intégration de sécurité matérielle" et Déployer des secrets sur Core. Les configurations pour le stockage des clés privées sur les HSM ou dans le système de fichiers sont prises en charge.

Champ	Description	Remarques
`Trajectorat`	Chemin d'accès absolu au certificat d'autorité de certification racine AWS IoT.	Il doit s'agir de l'URI d'un fichier du formulaire : `file:///`*`absolute/path/to/file`* . **Note** Assurez-vous que vos points de terminaison correspondent à votre type de certificat.
`PKCS11`		
`Moteur OpenSSL`	Facultatif. Chemin d'accès absolu au fichier de moteur OpenSSL `.so` pour activer la prise en charge PKCS # 11 sur OpenSSL.	Doit être un chemin d'accès à un fichier sur le système de fichiers.

Champ	Description	Remarques
		Cette propriété est requise si vous utilisez l'agent de mise à jour Greengrass OTA avec sécurité matérielle. Pour plus d'informations, consultez the section called "Configurer les mises à jour OTA".
`Fournisseur P11`	Chemin absolu vers la bibliothèque libdl-loadable d'implémentation PKCS # 11.	Doit être un chemin d'accès à un fichier sur le système de fichiers.
`Étiquette de machine à sous`	Étiquette d'emplacement qui est utilisée pour identifier le module matériel.	Doit se conformer aux spécifications d'étiquette PKCS # 11.
`slotUserPin`	Le code PIN utilisateur utilisé pour authentifier le noyau de Greengrass auprès du module.	Vous devez disposer d'autorisations suffisantes pour exécuter C_Sign avec les clés privées configurées.
`principals`		
`Certificat IoT`	The certificate and private key that the core uses to make requests to AWS IoT.	

Champ	Description	Remarques
`Certificat IoT. privateKeyPath`	Chemin d'accès à la clé privée principale.	Pour le stockage de système de fichiers, il doit s'agir de l'URI d'un fichier du formulaire : `file:///`*`absolute/path/to/file`* . Pour le stockage HSM, il doit s'agir du chemin d'accès RFC WF-7512 PKCS # 11 qui spécifie l'objet étiquette.
`Certificat IoT .CertificatePath`	Chemin d'accès absolu au certificat de votre noyau.	Il doit s'agir de l'URI d'un fichier du formulaire : `file:///`*`absolute/path/to/file`* .
`MQTT ServerCer tificate`	Facultatif. Clé privée que le noyau utilise en combinaison avec le certificat pour agir en tant que passerelle ou serveur MQTT.	

Champ	Description	Remarques
`MQTTServerCertificate . privateKeyPath`	Chemin d'accès à la clé privée du serveur MQTT local.	Utilisez cette valeur pour spécifier votre propre clé privée pour le serveur MQTT local. Pour le stockage de système de fichiers, il doit s'agir de l'URI d'un fichier du formulaire : `file:///`*`absolute/path/to/file`* . Pour le stockage HSM, il doit s'agir du chemin d'accès RFC WF-7512 PKCS # 11 qui spécifie l'objet étiquette. Si cette propriété n'est pas spécifiée, AWS IoT Greengrass effectue une rotation de la clé basée sur vos paramètres de rotation. Si cette propriété est spécifiée, le client est responsable de la rotation de la clé.
`SecretsManager`	The private key that secures the data key used for encryption. For more information, see Déployer des secrets sur Core.	

Champ	Description	Remarques
`SecretsManager .privateKeyPath`	Chemin d'accès à la clé privée du Secrets Manager local.	Seule une clé RSA est prise en charge. Pour le stockage de système de fichiers, il doit s'agir de l'URI d'un fichier du formulaire : `file:///`*`absolute/path/to/file`* . Pour le stockage HSM, il doit s'agir du chemin d'accès RFC WF-7512 PKCS # 11 qui spécifie l'objet étiquette. La clé privée doit être générée à l'aide du mécanisme de remplissage PKCS#1 v1.5.

Les propriétés de configuration suivantes sont également prises en charge :

Champ	Description	Remarques
`mqttMaxConnectionR etryInterval`	Facultatif. L'intervalle maximal (en secondes) entre les tentatives de connexion MQTT si la connexion est abandonnée.	Spécifiez cette valeur en tant qu'entier non signé. L'argument par défaut est `60`.
`managedRespawn`	Facultatif. Indique que l'agent OTA doit exécuter un code personnalisé avant une mise à jour.	Les valeurs valides sont `true` ou `false`. Pour plus d'informations, consultez Mises à jour OTA du logiciel AWS IoT Greengrass Core.

Champ	Description	Remarques
`writeDirectory`	Facultatif. Le répertoire d'écriture où sont AWS IoT Greengrass créées toutes les ressources de lecture/écriture.	Pour plus d'informations, consultez Configuration d'un répertoire en écriture pour 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"
  }
}
```

Le fichier `config.json` prend en charge les propriétés suivantes :

coreThing

Champ	Description	Remarques
`caPath`	Chemin d'accès au certificat d'authentification racine AWS IoT relatif au répertoire `/greengrass-root /certs`.	Pour une compatibilité descendante avec les versions antérieures à la version 1.7.0. Cette propriété est ignorée lorsque l'`crypto`objet est présent. **Note** Assurez-vous que vos points de terminaison correspondent à votre type de certificat.
`certPath`	Chemin d'accès au certificat de l'appareil principal relatif au répertoire `/greengrass-root /certs`.	Pour une compatibilité descendante avec les versions antérieures à la version 1.7.0. Cette propriété est ignorée lorsque l'`crypto`objet est présent.
`keyPath`	Chemin d'accès à la clé privée principale relatif au répertoire `/greengrass-root/certs`.	Pour une compatibilité descendante avec les versions antérieures à la version 1.7.0. Cette propriété est ignorée lorsque l'`crypto`objet est présent.

Champ	Description	Remarques
`thingArn`	Le nom de ressource Amazon (ARN) de l'AWS IoTobjet qui représente le périphérique AWS IoT Greengrass principal.	Trouvez l'ARN de votre cœur dans la AWS IoT Greengrass console, sous Cores, ou en exécutant la commande `aws greengrass get-core-definition-version` CLI.

Champ	Description	Remarques
`iotHost`	Votre point de terminaison AWS IoT.	Recherchez le point de terminaison dans la AWS IoT console sous Paramètres ou en exécutant la commande `aws iot describe-endpoint --endpoint-type iot:Data-ATS` CLI. Cette commande renvoie le point de terminaison ATS (Amazon Trust Services) . Pour de plus amples informations, veuillez consulter la documentation Authentification du serveur. **Note** Assurez-vous que vos points de terminaison correspondent à votre type de certificat. Assurez-vous que vos points de terminaison correspondent à vos Région AWS.

Champ	Description	Remarques
`ggHost`	Votre point de terminaison AWS IoT Greengrass.	Il s'agit de votre point de terminaison `iotHost` avec le préfixe de l'hôte remplacé par greengrass (par exemple, `greengrass-ats.iot.`*`region`*`.amazonaws.com`). Utilisez le même Région AWS que`iotHost`. **Note** Assurez-vous que vos points de terminaison correspondent à votre type de certificat. Assurez-vous que vos points de terminaison correspondent à vos Région AWS.
`iotMqttPort`	Facultatif. Numéro de port à utiliser pour la communication MQTT avec AWS IoT.	Les valeurs valides sont `8883` ou `443`. La valeur par défaut est `8883`. Pour plus d'informations, consultez Connexion au port 443 ou via un proxy réseau.
`keepAlive`	Facultatif. Période `KeepAlive` MQTT, en secondes	La plage valide est comprise entre 30 et 1 200 secondes. La valeur par défaut est `600`.

Champ	Description	Remarques
`networkProxy`	Facultatif. Objet qui définit une serveur proxy auquel vous connecter.	Le serveur proxy peut être HTTP ou HTTPS. Pour plus d'informations, consultez Connexion au port 443 ou via un proxy réseau.

runtime

Champ	Description	Remarques
`cgroup`		
`Utiliser le système D`	Indicates whether your device uses `systemd`.	Valid values are `oui` or `non`. Run the `check_ggc_dependencies` script in Module 1 to see if your device uses `systemd`.

crypto

L'objet `crypto`, ajouté dans la version 1.7.0, présente les propriétés qui prennent en charge le stockage de la clé privée sur un module de sécurité matérielle (HSM) via PKCS # 11 et le stockage secret local. Pour plus d'informations, consultez the section called "Intégration de sécurité matérielle" et Déployer des secrets sur Core. Les configurations pour le stockage des clés privées sur les HSM ou dans le système de fichiers sont prises en charge.

Champ	Description	Remarques
`Trajectorat`	Chemin d'accès absolu au certificat d'autorité de certification racine AWS IoT.	Il doit s'agir de l'URI d'un fichier du formulaire : `file:///absolute/path/to/file` .

Champ	Description	Remarques
		Note Assurez-vous que vos points de terminaison correspondent à votre type de certificat.
`PKCS11`		
`Moteur OpenSSL`	Facultatif. Chemin d'accès absolu au fichier de moteur OpenSSL `.so` pour activer la prise en charge PKCS # 11 sur OpenSSL.	Doit être un chemin d'accès à un fichier sur le système de fichiers. Cette propriété est requise si vous utilisez l'agent de mise à jour Greengrass OTA avec sécurité matérielle. Pour plus d'informations, consultez the section called "Configurer les mises à jour OTA".
`Fournisseur P11`	Chemin absolu vers la bibliothèque libdl-loadable d'implémentation PKCS # 11.	Doit être un chemin d'accès à un fichier sur le système de fichiers.
`Étiquette de machine à sous`	Étiquette d'emplacement qui est utilisée pour identifier le module matériel.	Doit se conformer aux spécifications d'étiquette PKCS # 11.

Champ	Description	Remarques
`slotUserPin`	Le code PIN utilisateur utilisé pour authentifier le noyau de Greengrass auprès du module.	Vous devez disposer d'autorisations suffisantes pour exécuter C_Sign avec les clés privées configurées.
`principals`		
`Certificat IoT`	The certificate and private key that the core uses to make requests to AWS IoT.	
`Certificat IoT. privateKeyPath`	Chemin d'accès à la clé privée principale.	Pour le stockage de système de fichiers, il doit s'agir de l'URI d'un fichier du formulaire : `file:///`*`absolute/path/to/file`* . Pour le stockage HSM, il doit s'agir du chemin d'accès RFC WF-7512 PKCS # 11 qui spécifie l'objet étiquette.
`Certificat IoT .CertificatePath`	Chemin d'accès absolu au certificat de votre noyau.	Il doit s'agir de l'URI d'un fichier du formulaire : `file:///`*`absolute/path/to/file`* .
`MQTT ServerCertificate`	Facultatif. Clé privée que le noyau utilise en combinaison avec le certificat pour agir en tant que passerelle ou serveur MQTT.	

Champ	Description	Remarques
`MQTTServerCertificate . privateKeyPath`	Chemin d'accès à la clé privée du serveur MQTT local.	Utilisez cette valeur pour spécifier votre propre clé privée pour le serveur MQTT local. Pour le stockage de système de fichiers, il doit s'agir de l'URI d'un fichier du formulaire : `file:///`*`absolute/path/to/file`* . Pour le stockage HSM, il doit s'agir du chemin d'accès RFC WF-7512 PKCS # 11 qui spécifie l'objet étiquette. Si cette propriété n'est pas spécifiée, AWS IoT Greengrass effectue une rotation de la clé basée sur vos paramètres de rotation. Si cette propriété est spécifiée, le client est responsable de la rotation de la clé.
`SecretsManager`	The private key that secures the data key used for encryption. For more information, see Déployer des secrets sur Core.	

Champ	Description	Remarques
`SecretsManager .privateKeyPath`	Chemin d'accès à la clé privée du Secrets Manager local.	Seule une clé RSA est prise en charge. Pour le stockage de système de fichiers, il doit s'agir de l'URI d'un fichier du formulaire : `file:///`*`absolute/path/to/file`* . Pour le stockage HSM, il doit s'agir du chemin d'accès RFC WF-7512 PKCS # 11 qui spécifie l'objet étiquette. La clé privée doit être générée à l'aide du mécanisme de remplissage PKCS#1 v1.5.

Les propriétés de configuration suivantes sont également prises en charge :

Champ	Description	Remarques
`mqttMaxConnectionR etryInterval`	Facultatif. L'intervalle maximal (en secondes) entre les tentatives de connexion MQTT si la connexion est abandonnée.	Spécifiez cette valeur en tant qu'entier non signé. L'argument par défaut est `60`.
`managedRespawn`	Facultatif. Indique que l'agent OTA doit exécuter un code personnalisé avant une mise à jour.	Les valeurs valides sont `true` ou `false`. Pour plus d'informations, consultez Mises à jour OTA du logiciel AWS IoT Greengrass Core.

Champ	Description	Remarques
`writeDirectory`	Facultatif. Le répertoire d'écriture où sont AWS IoT Greengrass créées toutes les ressources de lecture/écriture.	Pour plus d'informations, consultez Configuration d'un répertoire en écriture pour 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**
>
> Si vous utilisez l'option de création de groupe par défaut depuis la AWS IoT Greengrass console, le `config.json` fichier est déployé sur le périphérique principal dans un état de fonctionnement qui spécifie la configuration par défaut.

Le fichier `config.json` prend en charge les propriétés suivantes :

Champ	Description	Remarques
`caPath`	Chemin d'accès au certificat d'authentification racine AWS IoT relatif au répertoire `/greengrass-root/certs`.	Enregistrez le fichier sous `/greengrass-root/certs`.
`certPath`	Le chemin d'accès AWS IoT Greengrass au certificat principal relatif au `/greengrass-root/certs` répertoire.	Enregistrez le fichier sous `/greengrass-root/certs`.
`keyPath`	Le chemin d'accès à la AWS IoT Greengrass clé privée principale par rapport au `/greengrass-root/certs` répertoire.	Enregistrez le fichier sous `/greengrass-root/certs`.
`thingArn`	Le nom de ressource Amazon (ARN) de l'AWS IoTobjet qui représente le périphérique AWS IoT Greengrass principal.	Trouvez l'ARN de votre cœur dans la AWS IoT Greengrass console, sous Cores, ou en exécutant la commande `aws greengrass get-core-definition-version` CLI.
`iotHost`	Votre point de terminaison AWS IoT.	Trouvez-le dans la AWS IoT console, sous Paramètres, ou en exécutant la commande `aws iot describe-endpoint` CLI.
`ggHost`	Votre point de terminaison AWS IoT Greengrass.	Cette valeur utilise le format `greengrass.iot.region.amazonaw`

Champ	Description	Remarques
		`s.com` . Utilisez la même région que `iotHost`.
`keepAlive`	Période `KeepAlive` MQTT, en secondes	Cette valeur est facultative. L'argument par défaut est `600`.
`mqttMaxConnectionRetryInterval`	L'intervalle maximal (en secondes) entre les tentatives de connexion MQTT si la connexion est abandonnée.	Spécifiez cette valeur en tant qu'entier non signé. Cette valeur est facultative. L'argument par défaut est `60`.
`useSystemd`	Indique si votre appareil utilise `systemd`.	Les valeurs valides sont `yes` ou `no`. Exécutez le script `check_ggc_dependencies` dans Module 1 pour voir si votre appareil utilise `systemd`.
`managedRespawn`	Fonctionnalité de mise à jour optionnelle over-the-air (OTA), qui indique que l'agent OTA doit exécuter un code personnalisé avant une mise à jour.	Les valeurs valides sont `true` ou `false`. Pour plus d'informations, consultez Mises à jour OTA du logiciel AWS IoT Greengrass Core.
`writeDirectory`	Le répertoire d'écriture où sont AWS IoT Greengrass créées toutes les ressources de lecture/écriture.	Cette valeur est facultative. Pour plus d'informations, consultez Configuration d'un répertoire en écriture pour 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
}
```

Le fichier `config.json` existe dans */greengrass-root*`/config` et contient les paramètres suivants :

Champ	Description	Remarques
`caPath`	Chemin d'accès au certificat d'authentification racine AWS IoT relatif au dossier */greengrass-root* `/certs`.	Enregistrez le fichier sous le dossier */greengrass-root*`/certs`.
`certPath`	Le chemin d'accès AWS IoT Greengrass au certificat principal relatif au */greengrass-root* `/certs` dossier.	Enregistrez le fichier sous le dossier */greengrass-root*`/certs`.
`keyPath`	Le chemin d'accès à la AWS IoT Greengrass clé privée principale par rapport au */greengrass-root* `/certs` dossier.	Enregistrez le fichier sous le dossier */greengrass-root*`/certs`.

Champ	Description	Remarques
`thingArn`	Le nom de ressource Amazon (ARN) de l'AWS IoTobjet qui représente le périphérique AWS IoT Greengrass principal.	Trouvez l'ARN de votre cœur dans la AWS IoT Greengrass console, sous Cores, ou en exécutant la commande `aws greengrass get-core-definition-version` CLI.
`iotHost`	Votre point de terminaison AWS IoT.	Trouvez-le dans la AWS IoT console, sous Paramètres, ou en exécutant la `aws iot describe-endpoint` commande.
`ggHost`	Votre point de terminaison AWS IoT Greengrass.	Cette valeur utilise le format `greengrass.iot.` *`region`*`.amazonaws.com`. Utilisez la même région que `iotHost`.
`keepAlive`	Période `KeepAlive` MQTT, en secondes	Cette valeur est facultative. La valeur par défaut est de 600 secondes.
`useSystemd`	Indique si votre appareil utilise `systemd`.	Les valeurs valides sont `yes` ou `no`. Exécutez le script `check_ggc_dependencies` dans Module 1 pour voir si votre appareil utilise `systemd`.

Champ	Description	Remarques
`managedRespawn`	Fonctionnalité de mise à jour optionnelle over-the-air (OTA), qui indique que l'agent OTA doit exécuter un code personnalisé avant une mise à jour.	Pour plus d'informations, consultez Mises à jour OTA du logiciel 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
}
```

Le fichier `config.json` existe dans */greengrass-root*`/config` et contient les paramètres suivants :

Champ	Description	Remarques
`caPath`	Chemin d'accès au certificat d'authentification racine AWS IoT relatif au dossier	Enregistrez le fichier sous le dossier */greengrass-root*`/certs`.

Champ	Description	Remarques
	/greengrass-root / certs.	
certPath	Le chemin d'accès AWS IoT Greengrass au certificat principal relatif au */greengrass-root* / certs dossier.	Enregistrez le fichier sous le dossier */greengrass-root*/certs.
keyPath	Le chemin d'accès à la AWS IoT Greengrass clé privée principale par rapport au */greengrass-root* / certs dossier.	Enregistrez le fichier sous le dossier */greengrass-root*/certs.
thingArn	Le nom de ressource Amazon (ARN) de l'AWS IoTobjet qui représente le AWS IoT Greengrass noyau.	Vous pouvez trouver cette valeur dans la AWS IoT Greengrass console, sous la définition de votre AWS IoT objet.
iotHost	Votre point de terminaison AWS IoT.	Vous pouvez trouver cette valeur dans la AWS IoT console sous Paramètres.
ggHost	Votre point de terminaison AWS IoT Greengrass.	Vous pouvez trouver cette valeur dans la AWS IoT console sous Paramètres avec un greengrass. préfixe.
keepAlive	Période KeepAlive MQTT, en secondes	Cette valeur est facultative. La valeur par défaut est de 600 secondes.

Champ	Description	Remarques
`useSystemd`	Indicateur binaire, si votre appareil utilise `systemd`.	Les valeurs sont `yes` ou `no`. Utilisez le script de dépendance du Module 1 pour voir si votre appareil utilise `systemd`.
`managedRespawn`	Fonctionnalité de mise à jour optionnelle over-the-air (OTA), qui indique que l'agent OTA doit exécuter un code personnalisé avant une mise à jour.	Pour plus d'informations, consultez Mises à jour OTA du logiciel 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"
       }
   }
}
```

Le fichier `config.json` existe dans */greengrass-root*`/config` et contient les paramètres suivants :

Champ	Description	Remarques
`caPath`	Chemin d'accès au certificat d'authentification racine AWS IoT relatif au dossier */greengrass-root* / `certs`.	Enregistrez le fichier sous le dossier */greengrass-root*/`certs`.
`certPath`	Le chemin d'accès AWS IoT Greengrass au certificat principal relatif au */greengrass-root* / `certs` dossier.	Enregistrez le fichier sous le dossier */greengrass-root*/`certs`.
`keyPath`	Le chemin d'accès à la AWS IoT Greengrass clé privée principale par rapport au */greengrass-root* / `certs` dossier.	Enregistrez le fichier sous le dossier */greengrass-root*/`certs`.
`thingArn`	Le nom de ressource Amazon (ARN) de l'AWS IoTobjet qui représente le AWS IoT Greengrass noyau.	Vous pouvez trouver cette valeur dans la AWS IoT Greengrass console, sous la définition de votre AWS IoT objet.
`iotHost`	Votre point de terminaison AWS IoT.	Vous pouvez trouver cette valeur dans la AWS IoT console sous Paramètres.
`ggHost`	Votre point de terminaison AWS IoT Greengrass.	Vous pouvez trouver cette valeur dans la AWS IoT console sous Paramètres avec un `greengrass.` préfixe.

Champ	Description	Remarques
`keepAlive`	Période `KeepAlive` MQTT, en secondes	Cette valeur est facultative. La valeur par défaut est de 600 secondes.
`useSystemd`	Indicateur binaire, si votre appareil utilise `systemd`.	Les valeurs sont `yes` ou `no`. Utilisez le script de dépendance du Module 1 pour voir si votre appareil utilise `systemd`.

GGC v1.0

Dans AWS IoT Greengrass Core v1.0, `config.json` est déployé sur *`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"
       }
   }
}
```

Le fichier `config.json` existe dans */`greengrass-root`*`/configuration` et contient les paramètres suivants :

Champ	Description	Remarques
`caPath`	Chemin d'accès au certificat d'authentification racine AWS IoT relatif au dossier `/greengrass-root/configuration/certs`.	Enregistrez le fichier sous le dossier `/greengrass-root/configuration/certs`.
`certPath`	Le chemin d'accès AWS IoT Greengrass au certificat principal relatif au `/greengrass-root/configuration/certs` dossier.	Enregistrez le fichier sous le dossier `/greengrass-root/configuration/certs`.
`keyPath`	Le chemin d'accès à la AWS IoT Greengrass clé privée principale par rapport au `/greengrass-root/configuration/certs` dossier.	Enregistrez le fichier sous le dossier `/greengrass-root/configuration/certs`.
`thingArn`	Le nom de ressource Amazon (ARN) de l'AWS IoTobjet qui représente le AWS IoT Greengrass noyau.	Vous pouvez trouver cette valeur dans la AWS IoT Greengrass console sous la définition de votre AWS IoT objet.
`iotHost`	Votre point de terminaison AWS IoT.	Vous pouvez trouver cette valeur dans la AWS IoT console sous Paramètres.
`ggHost`	Votre point de terminaison AWS IoT Greengrass.	Vous pouvez trouver cette valeur dans la AWS IoT console sous Paramètres

Champ	Description	Remarques
		avec un `greengrass.` préfixe.
`keepAlive`	Période `KeepAlive` MQTT, en secondes	Cette valeur est facultative. La valeur par défaut est de 600 secondes.
`useSystemd`	Indicateur binaire si votre appareil utilise `systemd`.	Les valeurs sont `yes` ou `no`. Utilisez le script de dépendance du Module 1 pour voir si votre appareil utilise `systemd`.

Les points de terminaison du service doivent correspondre au type de certificat de l'autorité de certification racine

Vos points de terminaison AWS IoT Core et AWS IoT Greengrass doivent correspondre au type de certificat du certificat d'autorité de certification racine sur votre appareil. Si les points de terminaison et le type de certificat ne correspondent pas, les tentatives d'authentification entre le périphérique et AWS IoT Core ou AWS IoT Greengrass échouent. Pour plus d'informations, consultez la section Authentification du serveur dans le guide du AWS IoT développeur.

Si votre appareil utilise un certificat CA racine Amazon Trust Services (ATS), qui est la méthode préférée, il doit également utiliser les points de terminaison ATS pour la gestion des appareils et les opérations du plan de données de découverte. Les points de terminaison ATS incluent le segment `ats`, comme illustré dans la syntaxe suivante pour le point de terminaison AWS IoT Core.

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

> **Note**
>
> Pour des raisons de rétrocompatibilité, prend AWS IoT Greengrass actuellement en charge les anciens certificats et points de terminaison de l'autorité de certification VeriSign racine dans certains Région AWS s. Si vous utilisez un ancien certificat d'autorité de certification VeriSign racine, nous vous recommandons de créer un point de terminaison ATS et d'utiliser

plutôt un certificat d'autorité de certification racine ATS. Sinon, assurez-vous d'utiliser les points de terminaison hérités correspondants. Pour de plus amples informations, veuillez consulter Points de terminaison hérités pris en charge dans le Référence générale d'Amazon Web Services.

Points de terminaison dans config.json

Sur un appareil noyau Greengrass, les points de terminaison sont spécifiés dans l'objet `coreThing` du fichier `config.json`. La propriété `iotHost` représente le point de terminaison AWS IoT Core. La propriété `ggHost` représente le point de terminaison AWS IoT Greengrass. Dans l'exemple d'extrait suivant, ces propriétés spécifient des points de terminaison ATS.

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

Point de terminaison AWS IoT Core

Vous pouvez obtenir votre point de terminaison AWS IoT Core en exécutant la commande de l'interface de ligne de commande aws iot describe-endpoint avec le paramètre `--endpoint-type` qui convient.

- Pour renvoyer un point de terminaison signé ATS, exécutez :

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

- Pour renvoyer un ancien point de terminaison VeriSign signé, exécutez :

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

Point de terminaison AWS IoT Greengrass

Le point de terminaison AWS IoT Greengrass correspond au point de terminaison `iotHost` avec le préfixe de l'hôte remplacé par greengrass. Par exemple, le point de terminaison signé ATS est `greengrass-ats.iot.`*`region`*`.amazonaws.com`. La même région que pour le point de terminaison AWS IoT Core est utilisée.

Connexion au port 443 ou via un proxy réseau

Cette fonctionnalité est disponible pour AWS IoT Greengrass Core v1.7 et versions ultérieures.

Les noyaux Greengrass communiquent avec AWS IoT Core à l'aide du protocole de messagerie MQTT avec l'authentification client TLS. Par convention, MQTT via TLS utilise le port 8883. Toutefois, par mesure de sécurité, les environnements restrictifs peuvent limiter le trafic entrant et sortant à une petite plage de ports TCP. Par exemple, un pare-feu d'entreprise peut ouvrir le port 443 pour le trafic HTTPS, mais fermer les autres ports qui sont utilisés pour des protocoles moins courants, tels que le port 8883 pour le trafic MQTT. D'autres environnements restrictifs peuvent nécessiter que l'ensemble du trafic passe par un proxy HTTP avant de vous connecter à Internet.

Pour activer la communication dans ces scénarios, AWS IoT Greengrass autorise les configurations suivantes :

- MQTT avec l'authentification client TLS via le port 443. Si votre réseau autorise les connexions au port 443, vous pouvez configurer le noyau pour utiliser ce port pour le trafic MQTT au lieu du port 8883 par défaut. Il peut s'agir d'une connexion directe au port 443 ou d'une connexion via un serveur réseau proxy.

  AWS IoT Greengrass utilise l'extension TLS Application Layer Protocol Network (ALPN) pour permettre cette connexion. Comme avec la configuration par défaut, MQTT via TLS sur le port 443 utilise l'authentification client basée sur le certificat.

  Lorsqu'il est configuré pour utiliser une connexion directe au port 443, le cœur prend en charge les mises à jour AWS IoT Greengrass logicielles over-the-air (OTA). Cette prise en charge nécessite AWS IoT Greengrass Core v1.9.3 ou version ultérieure.
- Communication HTTPS via le port 443. Par défaut, AWS IoT Greengrass achemine le trafic HTTPS via le port 8443, mais vous pouvez le configurer afin qu'il utilise le port 443.
- Connexion via un proxy réseau. Vous pouvez configurer un serveur réseau proxy comme intermédiaire pour la connexion au noyau Greengrass. Seule l'authentification de base et les proxys HTTP et HTTPS sont pris en charge.

  La configuration du proxy est transmise aux fonctions Lambda définies par l'utilisateur via `http_proxy` les variables d'`https_proxy`environnement, `no_proxy` et. Les fonctions Lambda définies par l'utilisateur doivent utiliser ces paramètres transmis pour se connecter via le proxy. Les bibliothèques courantes utilisées par les fonctions Lambda pour établir des connexions (telles que les packages boto3 ou cURL et `requests` python) utilisent généralement ces variables

d'environnement par défaut. Si une fonction Lambda spécifie également ces mêmes variables d'environnement, AWS IoT Greengrass elle ne les remplace pas.

> **Important**
>
> Les noyaux Greengrass qui sont configurés pour utiliser un proxy réseau ne prennent pas en charge les mises à jour OTA.

Pour configurer MQTT via le port 443

Cette fonctionnalité nécessite AWS IoT Greengrass Core v1.7 ou version ultérieure.

Cette procédure permet au noyau Greengrass d'utiliser le port 443 pour la messagerie MQTT avec AWS IoT Core.

1. Exécutez la commande suivante pour arrêter le daemon Greengrass :

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

2. Ouvrez *greengrass-root*`/config/config.json` pour le modifier en tant qu'utilisateur su.

3. Dans l'objet `coreThing`, ajoutez la propriété `iotMqttPort` et définissez la valeur sur **443**, comme illustré dans l'exemple suivant.

   ```
   {
       "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. Lancez le démon.

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

Pour configurer HTTPS via le port 443

Cette fonctionnalité nécessite AWS IoT Greengrass Core v1.8 ou version ultérieure.

Cette procédure configure le noyau afin qu'il utilise le port 443 pour la communication HTTPS.

1. Exécutez la commande suivante pour arrêter le daemon Greengrass :

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

2. Ouvrez *greengrass-root*`/config/config.json` pour le modifier en tant qu'utilisateur su.
3. Dans l'objet `coreThing`, ajoutez les propriétés `iotHttpPort` et `ggHttpPort`, comme indiqué dans l'exemple suivant.

   ```
   {
       "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. Lancez le démon.

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

Pour configurer un proxy réseau

Cette fonctionnalité nécessite AWS IoT Greengrass Core v1.7 ou version ultérieure.

Cette procédure permet à AWS IoT Greengrass de se connecter à Internet via un proxy réseau HTTP ou HTTPS.

1. Exécutez la commande suivante pour arrêter le daemon Greengrass :

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

2. Ouvrez *greengrass-root*`/config/config.json` pour le modifier en tant qu'utilisateur su.
3. Dans l'`coreThing`objet, ajoutez l'objet NetworkProxy, comme indiqué dans l'exemple suivant.

```
{
    "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. Lancez le démon.

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

Objet networkProxy

Utilisez l'objet `networkProxy` pour spécifier les informations sur le proxy réseau. Cet objet a les propriétés suivantes.

Champ	Description
`noProxyAddresses`	Facultatif. Liste séparée par des virgules d'adresses IP ou de noms d'hôte qui sont dispensés par le proxy.

`proxy`	Proxy auquel se connecter. Un proxy a les propriétés suivantes.
	• `url`. URL du serveur proxy, au format`scheme://userinfo@host:port` .
	  • `scheme`. Le schéma. Doit être `http` ou `https`.
	  • `userinfo`. Facultatif Informations concernant le nom d'utilisateur et le mot de passe. Si cette valeur est spécifiée, les champs `username` et `password` sont ignorés.
	  • `host`. Le nom d'hôte ou l'adresse IP du serveur proxy.
	  • `port`. Facultatif Numéro de port. Si rien n'est spécifié, les valeurs par défaut suivantes sont utilisées :
	  • `http` : 80
	  • `https` : 443
	• `username`. Facultatif Nom d'utilisateur à utiliser pour s'authentifier auprès du serveur proxy.
	• `password`. Facultatif Mot de passe à utiliser pour s'authentifier auprès du serveur proxy.

Autoriser les points de terminaison

La communication entre les appareils Greengrass et AWS IoT Core ou AWS IoT Greengrass doit être authentifiée. Cette authentification est basée sur les certificats d'appareils X.509 enregistrés et sur des clés de chiffrement. Pour autoriser les demandes authentifiées à passer par des proxys sans chiffrement supplémentaire, autorisez les points de terminaison suivants.

Point de terminaison	Port	Description
`greengrass.` *`region`*`.amazonaws.com`	443	Utilisé pour les opérations de plan de contrôle pour la gestion des groupes.
`prefix``-ats.iot.` *`region`*`.amazonaws.com` or *`prefix`*`.iot.`*`region`*`.amazonaws.com`	MQTT : 8883 ou 443 HTTPS : 8443 ou 443	Utilisé pour les opérations de plan de contrôle pour la gestion des appareils, par exemple la synchronisation shadow. Autorisez l'utilisation d'un ou des deux points de terminaison, selon que vos appareils principaux et clients utilisent des certificats

Point de terminaison	Port	Description
		d'autorité de certification racine (préférés) d'Amazon Trust Services, des certificats d'autorité de certification racine existants ou les deux. Pour plus d'informations, consultez the section called “Les points de terminaison du service doivent correspondre au type de certificat”.

Point de terminaison	Port	Description

`greengrass-ats.iot. region.amazonaws.com`
 	
or
 	
`greengrass.iot. region.amazonaws.com`
	8443 ou 443	Utilisé pour les opérations de détection d'appareils.
		
		Autorisez l'utilisation d'un ou des deux points de terminaison, selon que vos appareils principaux et clients utilisent des certificats d'autorité de certification racine (préférés) d'Amazon Trust Services, des certificats d'autorité de certification racine existants ou les deux. Pour plus d'informations, consultez the section called

Point de terminaison	Port	Description
		“Les points de terminaison du service doivent correspondre au type de certificat”. **Note** Les clients qui se connectent sur le port 443 doivent implémenter l'extension TLS ALPN (Application Layer Protocol Negotiation) et la transmettre `x-amzn-`

Point de terminaison	Port	Description
		`http-ca` comme `ProtocolName` dans le. `ProtocolNameList` Pour plus d'informations, consultez la section Protocoles du guide du AWS IoT développeur.

Point de terminaison	Port	Description
`*.s3.amazonaws.com`	443	Utilisé pour les opérations de déploiement et les over-the-air mises à jour. Ce format comprend le caractère *, car les préfixes des points de terminaison sont contrôlés en interne et peuvent évoluer à tout moment.
`logs.region.amazonaws.com`	443	Requis si le groupe Greengrass est configuré pour écrire des journaux dans CloudWatch.

Configuration d'un répertoire en écriture pour AWS IoT Greengrass

Cette fonctionnalité est disponible pour AWS IoT Greengrass Core v1.6 et versions ultérieures.

Par défaut, le logiciel AWS IoT Greengrass Core est déployé dans un répertoire racine unique où AWS IoT Greengrass effectue toutes les opérations de lecture et d'écriture. Cependant, vous pouvez configurer AWS IoT Greengrass pour qu'il utilise un répertoire distinct pour toutes les opérations d'écriture, y compris la création de répertoires et de fichiers. Dans ce cas, AWS IoT Greengrass utilise deux répertoires de niveau supérieur :

- Le répertoire *greengrass-root*, que vous pouvez conserver accessible en lecture et en écriture ou, si vous le souhaitez, rendre accessible en lecture seule. Celui-ci contient le logiciel AWS IoT Greengrass et d'autres composants critiques qui doivent rester immuables lors de l'exécution (par exemple, les certificats et `config.json`).
- Le répertoire en écriture spécifié. Il contient du contenu inscriptible, tel que des journaux, des informations d'état et des fonctions Lambda définies par l'utilisateur déployées.

Cette configuration entraîne la structure de répertoires suivante.

Répertoire racine 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/
```

Répertoire en écriture

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

Pour configurer un répertoire en écriture

1. Exécutez la commande suivante pour arrêter le démon AWS IoT Greengrass :

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

2. Ouvrez *greengrass-root*`/config/config.json` pour le modifier en tant qu'utilisateur su.

3. Ajoutez `writeDirectory` en tant que paramètre et spécifiez le chemin d'accès au répertoire cible, comme illustré dans l'exemple suivant.

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

> **Note**
>
> Vous pouvez mettre à jour le paramètre `writeDirectory` aussi souvent que vous le souhaitez. Une fois que le paramètre est mis à jour, AWS IoT Greengrass utilise le répertoire d'écriture nouvellement spécifié lors du démarrage suivant, mais ne migre pas le contenu du répertoire en écriture précédent.

4. Maintenant que votre répertoire en écriture est configuré, vous pouvez, si vous le souhaitez, rendre le répertoire *`greengrass-root`* accessible en lecture seule. Pour en savoir plus, consultez Pour rendre le répertoire racine Greengrass accessible en lecture seule.

   Sinon, démarrez le démon AWS IoT Greengrass :

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

Pour rendre le répertoire racine Greengrass accessible en lecture seule

Cette étape est nécessaire uniquement si vous souhaitez que le répertoire racine Greengrass soit accessible en lecture seule. Le répertoire en écriture doit être configuré avant de commencer cette procédure.

1. Accorder des autorisations d'accès aux répertoires requis :

   a. Accordez les autorisations en lecture et en écriture au propriétaire de `config.json`.

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

   b. Définissez ggc_user en tant que propriétaire des répertoires certs et Lambda système.

      ```
      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**
>
> Les comptes ggc_user et ggc_group sont utilisés par défaut pour exécuter les fonctions Lambda du système. Si vous avez configuré l'identité d'accès par défaut au niveau du groupe de façon à utiliser différents comptes, vous devez octroyer des autorisations à cet utilisateur (UID) et à ce groupe (GID) à la place.

2. Rendez le répertoire *greengrass-root* accessible en lecture seule en utilisant la méthode de votre choix.

   > **Note**
   >
   > Une des méthodes pour rendre le répertoire *greengrass-root* accessible en lecture seule consiste à monter le répertoire en lecture seule. Toutefois, pour appliquer des mises à jour over-the-air (OTA) au logiciel AWS IoT Greengrass Core dans un répertoire monté, le répertoire doit d'abord être démonté, puis remonté après la mise à jour. Vous pouvez ajouter ces opérations `umount` et `mount` aux scripts `ota_pre_update` et `ota_post_update`. Pour plus d'informations sur les mises à jour OTA, consultez the section called “Agent de mise à jour OTA Greengrass” et the section called “Commande managedRespawn avec mises à jour OTA”.

3. Lancez le démon.

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

   Si les autorisations de l'étape 1 ne sont pas correctement définies, le démon ne démarre pas.

Configuration des paramètres MQTT

Dans l'AWS IoT Greengrassenvironnement, les appareils clients locaux, les fonctions Lambda, les connecteurs et les composants du système peuvent communiquer entre eux et avec. AWS IoT Core Toutes les communications passent par le noyau, qui gère les abonnements autorisant la communication MQTT entre les entités.

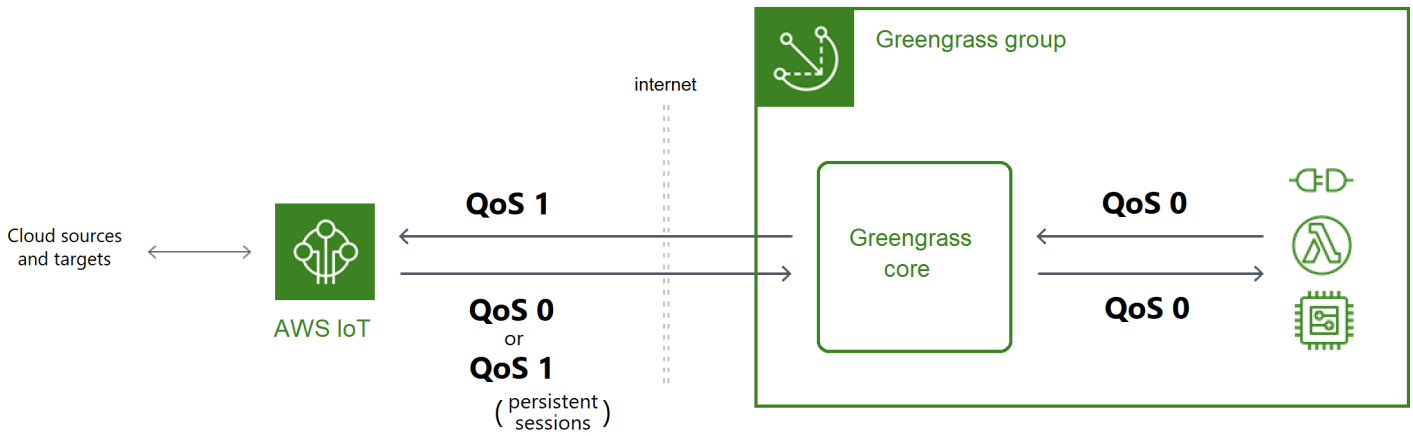
Pour de plus amples informations sur les paramètres MQTT que vous pouvez configurer pour AWS IoT Greengrass, consultez les sections suivantes :

- the section called “Message de qualité de service”
- the section called “File d'attente de messages MQTT”
- the section called “Sessions persistantes MQTT avec AWS IoT Core”
- the section called “ID client pour les connexions MQTT avec AWS IoT”
- Port MQTT pour la messagerie locale
- the section called “Délai d'expiration pour les opérations de publication, d'abonnement et de désinscription dans le cadre des connexions MQTT avec AWS Cloud”

> **Note**
>
> OPC-UA est une norme d'échange d'informations pour la communication industrielle. Pour implémenter la prise en charge de l'OPC-UA sur le cœur de Greengrass, vous pouvez utiliser le connecteur IoT. SiteWise Le connecteur envoie les données d'appareil industrielles provenant de serveurs OPC-UA aux propriétés des ressources dans AWS IoT SiteWise.

Message de qualité de service

AWS IoT Greengrass prend en charge les niveaux de qualité de service (QoS) 0 ou 1, en fonction de votre configuration, ainsi que de la cible et de la direction de la communication. Le noyau Greengrass agit comme un client pour la communication avec AWS IoT Core et comme un courtier de messages pour la communication sur le réseau local.



Pour plus d'informations sur MQTT et QoS, consultez Getting Started sur le site Web de MQTT.

Communication avec le AWS Cloud

- Les messages sortants utilisent QoS 1

  Le noyau envoie des messages destinés aux AWS Cloud cibles à l'aide de QoS 1. AWS IoT Greengrassutilise une file de messages MQTT pour traiter ces messages. Si la livraison du message n'est pas confirmée parAWS IoT, le message est mis en attente pour être réessayé ultérieurement. Le message ne peut pas être réessayé si la file d'attente est pleine. La confirmation de remise du message peut aider à minimiser les pertes de données dues à une connectivité intermittente.

  Comme les messages sortants doivent AWS IoT utiliser QoS 1, le débit maximal auquel le cœur de Greengrass peut envoyer des messages dépend de la latence entre le cœur et. AWS IoT Chaque fois que le noyau envoie un message, il attend d'AWS IoTavoir accusé réception du message avant d'envoyer le message suivant. Par exemple, si le temps d'aller-retour entre le cœur et le sien Région AWS est de 50 millisecondes, le cœur peut envoyer jusqu'à 20 messages par seconde. Tenez compte de ce comportement lorsque vous choisissez l'Région AWSendroit où votre cœur se connecte. Pour ingérer de gros volumes de données IoT dans leAWS Cloud, vous pouvez utiliser le gestionnaire de flux.

  Pour plus d'informations sur la file d'attente de messages MQTT, notamment sur la façon de configurer un cache de stockage local capable de conserver les messages destinés aux AWS Cloud cibles, consultezthe section called “File d'attente de messages MQTT”.

- Les messages entrants utilisent le niveau de qualité QoS 0 (par défaut) ou QoS 1

  Par défaut, le noyau s'abonne avec QoS 0 aux messages AWS Cloud provenant des sources. Si vous activez les sessions persistantes, le noyau s'abonne avec le niveau QoS 1. Cela peut permettre de réduire les pertes de données dues à l'intermittence de la connexion. Pour gérer le niveau de qualité QoS pour ces abonnements, vous configurez les paramètres de persistance sur le composant système spouleur local.

  Pour plus d'informations, notamment sur la manière de permettre au noyau d'établir une session persistante avec AWS Cloud des cibles, consultezthe section called “Sessions persistantes MQTT avec AWS IoT Core”.

Communication avec les cibles locales

Toutes les communications locales utilisent le niveau QoS 0. Le noyau tente d'envoyer un message à une cible locale, qui peut être une fonction Greengrass Lambda, un connecteur ou

un appareil client. Le noyau ne stocke pas de messages et ne confirme pas la livraison. Les messages peuvent être supprimés n'importe où entre les composants.

> **Note**
>
> Bien que la communication directe entre les fonctions Lambda n'utilise pas la messagerie MQTT, le comportement est le même.

File d'attente de messages MQTT pour les cibles cloud

Les messages MQTT destinés aux AWS Cloud cibles sont mis en file d'attente en attente de traitement. Les messages en attente sont traités selon l'ordre FIFO (premier entré, premier sorti). Lorsqu'un message est traité et publié dans AWS IoT Core, il est supprimé de la file d'attente.

Par défaut, le cœur de Greengrass stocke en mémoire les messages non traités destinés aux cibles. AWS Cloud Si vous le souhaitez, vous pouvez configurer le noyau pour stocker les messages non traités dans une mémoire cache locale. Contrairement au stockage en mémoire, le cache de stockage local peut être conservé malgré les redémarrages du noyau (par exemple, après le déploiement d'un groupe ou un redémarrage de l'appareil), ce qui permet à AWS IoT Greengrass de continuer à traiter les messages. Vous pouvez également configurer la taille du stockage.

> **Warning**
>
> Le noyau de Greengrass peut mettre en file d'attente des messages MQTT dupliqués lorsqu'il perd la connexion, car il tente à nouveau une opération de publication avant que le client MQTT ne détecte qu'il est hors ligne. Pour éviter les messages MQTT dupliqués pour les cibles du cloud, configurez la `keepAlive` valeur du noyau à moins de la moitié de sa `mqttOperationTimeout` valeur. Pour plus d'informations, consultez Fichier de configuration de AWS IoT Greengrass Core.

AWS IoT Greengrassutilise le composant du système de spouleur (fonction `GGCloudSpooler` Lambda) pour gérer la file d'attente de messages. Vous pouvez utiliser les variables d'environnement `GGCloudSpooler` suivantes pour configurer les paramètres de stockage.

- GG_CONFIG_STORAGE_TYPE. Emplacement de la file d'attente de messages. Les valeurs suivantes sont valides :

- `FileSystem`. Stockez les messages non traités dans le cache de stockage local sur le disque du périphérique principal physique. Lorsque le noyau redémarre, les messages mis en file d'attente sont conservés en vue de leur traitement. Les messages sont supprimés une fois qu'ils sont traités.
- `Memory` (default). Les messages non traités sont stockés en mémoire. Lorsque le noyau redémarre, les messages mis en file d'attente sont perdus.

  Cette option est optimisée pour les périphériques avec des capacités matérielles restreintes. Lorsque vous utilisez cette configuration, nous vous recommandons de déployer des groupes ou de redémarrer l'appareil lorsque l'interruption de service est la moins importante.

- GG_CONFIG_MAX_SIZE_BYTES. Taille du stockage, en octets. Cette valeur peut être n'importe quel nombre entier non négatif supérieur ou égal à 262144 (256 Ko) ; une taille plus petite empêche le logiciel AWS IoT Greengrass Core de démarrer. La taille par défaut est de 2.5 Mo. Lorsque la taille limite est atteinte, les messages les plus anciens de la file d'attente sont remplacés par de nouveaux messages.

> **Note**
>
> Cette fonctionnalité est disponible pour AWS IoT Greengrass Core v1.6 et versions ultérieures. Les versions antérieures utilisent le stockage en mémoire avec une taille de file d'attente de 2,5 Mo. Vous ne pouvez pas configurer les paramètres de stockage pour les versions antérieures.

Pour mettre en cache des messages dans le stockage local

Vous pouvez configurer AWS IoT Greengrass pour mettre en cache des messages dans le système de fichiers, afin qu'ils soient conservés lors des redémarrages du noyau. Pour ce faire, vous déployez une version de définition de fonction dans laquelle la fonction `GGCloudSpooler` définit le type de stockage sur `FileSystem`. Vous devez utiliser l'API AWS IoT Greengrass pour configurer le cache de stockage local. Vous ne pouvez pas réaliser cette opération dans la console.

La procédure suivante utilise la commande `create-function-definition-version`CLI pour configurer le spouleur afin d'enregistrer les messages en file d'attente dans le système de fichiers. Elle configure également une file d'attente d'une taille de 2,6 Mo.

1. Obtenez les ID du groupe et de la version de groupe Greengrass cible. Cette procédure suppose qu'il s'agit de la dernière version du groupe et du groupe. La requête suivante renvoie le dernier groupe créé.

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

   Vous pouvez également procéder à une interrogation par nom. Les noms de groupe ne devant pas nécessairement être uniques, plusieurs groupes peuvent être renvoyés.

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

   > **Note**
   >
   > Vous pouvez également trouver ces valeurs dans la AWS IoT console. L'ID du groupe s'affiche sur la page Paramètres du groupe. Les identifiants de version du groupe sont affichés dans l'onglet Déploiements du groupe.

2. Copiez les valeurs `LatestVersion` et `Id` du groupe cible dans la sortie.
3. Obtenez la version de groupe la plus récente.

   - Remplacez *`group-id`* par la propriété `Id` que vous avez copiée.
   - Remplacez *`latest-group-version-id`* par l'`LatestVersion` que vous avez copié.

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

4. À partir de l'objet `Definition` de la sortie, copiez le `CoreDefinitionVersionArn` et les ARN de tous les autres composants de groupe à l'exception de `FunctionDefinitionVersionArn`. Vous utilisez ces valeurs lorsque vous créez une nouvelle version de groupe.
5. Depuis `FunctionDefinitionVersionArn` dans la sortie, copiez l'ID de la définition de fonction. L'ID est le GUID qui suit le segment `functions` dans l'ARN, comme illustré dans l'exemple suivant.

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

> **Note**
>
> Vous pouvez également créer une définition de fonction en exécutant la `create-function-definition`commande, puis en copiant l'ID depuis la sortie.

6. Ajoutez une version de définition de fonction à la définition de fonction.

   - *`function-definition-id`*Remplacez-le par celui `Id` que vous avez copié pour la définition de la fonction.
   - *`arbitrary-function-id`*Remplacez-le par un nom pour la fonction, tel que**`spooler-function`**.
   - Ajoutez au tableau toutes les fonctions Lambda que vous souhaitez inclure dans cette version. `functions` Vous pouvez utiliser la `get-function-definition-version`commande pour obtenir les fonctions Greengrass Lambda à partir d'une version de définition de fonction existante.

> **Warning**
>
> Assurez-vous de spécifier une valeur pour `GG_CONFIG_MAX_SIZE_BYTES` qui soit supérieure ou égale à 262 144. Une taille plus petite empêche le logiciel AWS IoT Greengrass Core de démarrer.

```
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**
>
> Si vous avez précédemment défini la variable d'environnement `GG_CONFIG_SUBSCRIPTION_QUALITY` pour prendre en charge les sessions persistantes avec AWS IoT Core, incluez-la dans cette instance de fonction.

7. Copiez l'`Arn` de la version de définition de fonction à partir de la sortie.
8. Créez une version de groupe contenant la fonction Lambda du système.

   - Remplacez *`group-id`* par l'`Id` du groupe.
   - *`core-definition-version-arn`*Remplacez-le par celui `CoreDefinitionVersionArn` que vous avez copié à partir de la dernière version du groupe.
   - *`function-definition-version-arn`*Remplacez-le par celui `Arn` que vous avez copié pour la nouvelle version de définition de fonction.
   - Remplacez les ARN des autres composants de groupe (par exemple `SubscriptionDefinitionVersionArn` ou `DeviceDefinitionVersionArn`) que vous avez copiés de la version de groupe la plus récente.
   - Supprimez tous les paramètres inutilisés. Par exemple, supprimez `--resource-definition-version-arn` si votre version de groupe ne contient aucune ressource.

```
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. Copiez la `Version` à partir de la sortie. Il s'agit de l'ID de la nouvelle version de groupe.
10. Déployez le groupe avec la nouvelle version de groupe.

    - Remplacez *`group-id`* par l'`Id` que vous avez copié pour le groupe.
    - *`group-version-id`*Remplacez-le par celui `Version` que vous avez copié pour la nouvelle version du groupe.

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

Pour mettre à jour les paramètres de stockage, utilisez l'API AWS IoT Greengrass pour créer une nouvelle version de définition de fonction qui contient la fonction `GGCloudSpooler` avec la configuration mise à jour. Ensuite, ajoutez la version de définition de fonction à une nouvelle version de groupe (avec vos autres composants de groupe) et déployez la version de groupe. Si vous souhaitez restaurer la configuration par défaut, vous pouvez déployer une version de définition de fonction qui n'inclut pas la fonction `GGCloudSpooler`.

Cette fonction Lambda du système n'est pas visible dans la console. Toutefois, lorsque la fonction est ajoutée à la version de groupe la plus récente, elle est incluse dans les déploiements que vous effectuez à partir de la console (sauf si vous utilisez l'API pour la remplacer ou la supprimer).

Sessions persistantes MQTT avec AWS IoT Core

Cette fonction est disponible pour AWS IoT Greengrass Core v1.10 et versions ultérieures.

Un noyau Greengrass peut établir une session persistante avec le courtier de messages AWS IoT. Une session persistante est une connexion continue qui permet au noyau de recevoir les messages envoyés pendant qu'il est hors connexion. Le noyau est le client dans la connexion.

Dans une session persistante, le courtier de messages AWS IoT enregistre tous les abonnements souscrits par le noyau pendant la connexion. Si le cœur se déconnecte, le courtier de AWS IoT messages stocke les messages non reconnus et les nouveaux messages publiés sous QoS 1 et destinés à des cibles locales, telles que les fonctions Lambda et les appareils clients. Lorsque le noyau se reconnecte, la session persistante reprend son exécution et le courtier de messages AWS IoT envoie les messages stockés au noyau à un rythme maximum de 10 messages par seconde. Les sessions persistantes ont une période d'expiration par défaut de 1 heure, qui commence lorsque le courtier de messages détecte la déconnexion du noyau. Pour plus d'informations, consultez la section Sessions persistantes MQTT dans le manuel du AWS IoT développeur.

AWS IoT Greengrassutilise le composant du système de spouleur (fonction `GGCloudSpooler` Lambda) pour créer des abonnements ayant AWS IoT pour source. Vous pouvez utiliser la variable d'environnement `GGCloudSpooler` suivante pour configurer des sessions persistantes.

- GG_CONFIG_SUBSCRIPTION_QUALITY. La qualité des abonnements qui ont AWS IoT comme source. Les valeurs suivantes sont valides :
  - `AtMostOnce` (default). Désactive les sessions persistantes. Les abonnements utilisent le niveau QoS 0.
  - `AtLeastOncePersistent`. Active les sessions persistantes. Définit l'indicateur `cleanSession` sur `0` dans les messages `CONNECT` et s'abonne au niveau QoS 1.

    Les messages publiés avec le niveau QoS 1 et qui sont reçus par le noyau atteignent obligatoirement la file d'attente de travail en mémoire du démon Greengrass. Le noyau accuse réception du message après son ajout à la file d'attente. Les communications ultérieures entre la file d'attente et la cible locale (par exemple, la fonction, le connecteur ou le périphérique Greengrass Lambda) sont envoyées sous la forme de QoS 0. AWS IoT Greengrassne garantit pas la livraison aux cibles locales.

    > **Note**
    >
    > Vous pouvez utiliser la propriété de configuration maxWorkItemCount pour contrôler la taille de la file d'attente des éléments de travail. Par exemple, vous pouvez augmenter la taille de la file d'attente si votre charge de travail nécessite un trafic MQTT élevé.

    Lorsque les sessions persistantes sont activées, le noyau ouvre au moins une connexion supplémentaire pour l'échange de messages MQTT avec AWS IoT. Pour plus d'informations, consultez the section called “ID client pour les connexions MQTT avec AWS IoT”.

Pour configurer des sessions permanentes MQTT

Vous pouvez configurer AWS IoT Greengrass pour utiliser les sessions persistantes avec AWS IoT Core. Pour ce faire, vous déployez une version de définition de fonction dans laquelle la fonction `GGCloudSpooler` définit la qualité d'abonnement sur `AtLeastOncePersistent`. Ce paramètre s'applique à tous vos abonnements ayant AWS IoT Core (`cloud`) comme source. Vous devez utiliser l'API AWS IoT Greengrass pour configurer les sessions persistantes. Vous ne pouvez pas réaliser cette opération dans la console.

La procédure suivante utilise la commande `create-function-definition-version`CLI pour configurer le spouleur afin qu'il utilise des sessions persistantes. Dans cette procédure, on suppose que vous mettez à jour la configuration de la version de groupe la plus récente d'un groupe existant.

1. Obtenez les ID du groupe et de la version de groupe Greengrass cible. Cette procédure suppose qu'il s'agit de la dernière version du groupe et du groupe. La requête suivante renvoie le dernier groupe créé.

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

   Vous pouvez également procéder à une interrogation par nom. Les noms de groupe ne devant pas nécessairement être uniques, plusieurs groupes peuvent être renvoyés.

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

   > **Note**
   >
   > Vous pouvez également trouver ces valeurs dans la AWS IoT console. L'ID du groupe s'affiche sur la page Paramètres du groupe. Les identifiants de version du groupe sont affichés dans l'onglet Déploiements du groupe.

2. Copiez les valeurs `LatestVersion` et `Id` du groupe cible dans la sortie.
3. Obtenez la version de groupe la plus récente.

   - Remplacez *`group-id`* par la propriété `Id` que vous avez copiée.
   - Remplacez *`latest-group-version-id`* par l'`LatestVersion` que vous avez copié.

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

4. À partir de l'objet `Definition` de la sortie, copiez le `CoreDefinitionVersionArn` et les ARN de tous les autres composants de groupe à l'exception de `FunctionDefinitionVersionArn`. Vous utilisez ces valeurs lorsque vous créez une nouvelle version de groupe.
5. Depuis `FunctionDefinitionVersionArn` dans la sortie, copiez l'ID de la définition de fonction. L'ID est le GUID qui suit le segment `functions` dans l'ARN, comme illustré dans l'exemple suivant.

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

> **Note**
>
> Vous pouvez également créer une définition de fonction en exécutant la `create-function-definition`commande, puis en copiant l'ID depuis la sortie.

6. Ajoutez une version de définition de fonction à la définition de fonction.

   - *`function-definition-id`*Remplacez-le par celui `Id` que vous avez copié pour la définition de la fonction.
   - *`arbitrary-function-id`*Remplacez-le par un nom pour la fonction, tel que**`spooler-function`**.
   - Ajoutez au tableau toutes les fonctions Lambda que vous souhaitez inclure dans cette version. `functions` Vous pouvez utiliser la `get-function-definition-version`commande pour obtenir les fonctions Greengrass Lambda à partir d'une version de définition de fonction existante.

```
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**
>
> Si vous avez précédemment défini les variables d'environnement `GG_CONFIG_STORAGE_TYPE`ou `GG_CONFIG_MAX_SIZE_BYTES` afin de définir les paramètres de stockage, incluez-les dans cette instance de fonction.

7. Copiez l'`Arn` de la version de définition de fonction à partir de la sortie.

8. Créez une version de groupe contenant la fonction Lambda du système.

   - Remplacez *group-id* par l'`Id` du groupe.
   - *core-definition-version-arn*Remplacez-le par celui `CoreDefinitionVersionArn` que vous avez copié à partir de la dernière version du groupe.
   - *function-definition-version-arn*Remplacez-le par celui `Arn` que vous avez copié pour la nouvelle version de définition de fonction.
   - Remplacez les ARN des autres composants de groupe (par exemple `SubscriptionDefinitionVersionArn` ou `DeviceDefinitionVersionArn`) que vous avez copiés de la version de groupe la plus récente.
   - Supprimez tous les paramètres inutilisés. Par exemple, supprimez `--resource-definition-version-arn` si votre version de groupe ne contient aucune ressource.

   ```
   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. Copiez la `Version` à partir de la sortie. Il s'agit de l'ID de la nouvelle version de groupe.
10. Déployez le groupe avec la nouvelle version de groupe.

    - Remplacez *group-id* par l'`Id` que vous avez copié pour le groupe.
    - *group-version-id*Remplacez-le par celui `Version` que vous avez copié pour la nouvelle version du groupe.

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

11. (Facultatif) Augmentez la propriété maxWorkItemCount dans le fichier de configuration principal. Cela peut aider le noyau à gérer l'augmentation du trafic MQTT et la communication avec les cibles locales.

Pour mettre à jour le noyau en fonction de ces modifications de configuration, utilisez l'API AWS IoT Greengrass pour créer une nouvelle version de définition de fonction contenant la fonction `GGCloudSpooler` avec la configuration mise à jour. Ensuite, ajoutez la version de définition de fonction à une nouvelle version de groupe (avec vos autres composants de groupe) et déployez la version de groupe. Si vous souhaitez restaurer la configuration par défaut, vous pouvez créer une version de définition de fonction qui n'inclut pas la fonction `GGCloudSpooler`.

Cette fonction Lambda du système n'est pas visible dans la console. Toutefois, lorsque la fonction est ajoutée à la version de groupe la plus récente, elle est incluse dans les déploiements que vous effectuez à partir de la console (sauf si vous utilisez l'API pour la remplacer ou la supprimer).

ID client pour les connexions MQTT avec AWS IoT

Cette fonctionnalité est disponible pour AWS IoT Greengrass Core v1.8 et versions ultérieures.

Le noyau Greengrass ouvre les connexions MQTT avec AWS IoT Core pour des opérations telles que la synchronisation shadow et la gestion de certificats. Pour ces connexions, les ID client prévisibles sont basés sur le nom de l'élément principal. Les identifiants clients prévisibles peuvent être utilisés avec les fonctionnalités de surveillance, d'audit et de tarification, y compris les événements AWS IoT Device Defender liés au AWS IoT cycle de vie. Vous pouvez aussi créer des ID client prévisibles logiques (par exemple, les modèles de stratégie d'abonnement basés sur les attributs de certificat).

GGC v1.9 and later

Deux composants de système Greengrass ouvrent des connexions MQTT avec AWS IoT Core. Ces composants utilisent les modèles suivants pour générer l'ID client pour les connexions.

Opération	Modèle d'ID client
Déploiements	*core-thing-name* Exemple : `MyCoreThing` Utilisez cet ID client pour vous connecter , vous déconnecter, vous abonner et vous désabonner des notifications d'événement de cycle de vie.
Abonnements	*core-thing-name* `-c`*n*

Opération	Modèle d'ID client
	Exemple : `MyCoreThing-c01` *n*est un entier qui commence à 00 et augmente à chaque nouvelle connexion jusqu'à un maximum de 250. Le nombre de connexions est déterminé par le nombre d'appareils synchronisant leur état fantôme AWS IoT Core (maximum 2 500 par groupe) et par le nombre d'abonnements ayant `cloud` comme source dans le groupe (maximum 10 000 par groupe). Le composant du système de spouleur se connecte pour échanger des messages relatifs AWS IoT Core à des abonnemen ts avec une source ou une cible cloud. Le spouleur agit également en tant que proxy pour l'échange de messages entre AWS IoT Core d'une part et le service de shadow local ainsi que le gestionnaire de certificats d'appareils d'autre part.

Pour calculer le nombre de connexions MQTT par groupe, utilisez la formule suivante :

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

Où,

- nombre de connexions pour l'agent de déploiement = 1.
- nombre de connexions pour les abonnements =`(2 subscriptions for supporting certificate generation + number of MQTT topics in AWS IoT Core + number of device shadows synced) / 50`.
  - Où, `50` = le nombre maximum d'abonnements par connexion AWS IoT Core pouvant être pris en charge.

> **Note**
>
> Si vous activez les sessions persistantes pour l'abonnement avec AWS IoT Core, le noyau ouvre au moins une connexion supplémentaire à utiliser dans une session persistante. Les composants système ne prennent pas en charge les sessions persistantes ; ils ne peuvent donc pas partager cette connexion.

Pour réduire le nombre de connexions MQTT et réduire les coûts, vous pouvez utiliser les fonctions Lambda locales pour agréger les données à la périphérie. Vous envoyez ensuite les données agrégées auAWS Cloud. Par conséquent, vous utilisez moins de sujets MQTT dansAWS IoT Core. Pour plus d'informations, consultez Tarification d'AWS IoT Greengrass.

GGC v1.8

Plusieurs composants système Greengrass ouvrent des connexions MQTT avec AWS IoT Core. Ces composants utilisent les modèles suivants pour générer l'ID client pour les connexions.

Opération	Modèle d'ID client
Déploiements	*core-thing-name* Exemple : `MyCoreThing` Utilisez cet ID client pour vous connecter , vous déconnecter, vous abonner et vous désabonner des notifications d'événement de cycle de vie.
Échange de messages MQTT avec AWS IoT Core	*core-thing-name* `-spr` Exemple : `MyCoreThing-spr`
Synchronisation shadow	*core-thing-name* `-s`*nn* Exemple : `MyCoreThing-s01` *nn* est un entier qui commence à 00 et s'incrémente à chaque nouvelle connexion jusqu'à un maximum de 03. Le nombre de

Opération	Modèle d'ID client
	connexions est déterminé par le nombre de périphériques (200 périphériques par groupe maximum) qui synchronisent leur état shadow avec AWS IoT Core (50 abonnements par connexion maximum).
Gestion des certificats de périphérique	*core-thing-name* `-dcm` Exemple : `MyCoreThing-dcm`

> **Note**
>
> Dupliquer les ID client utilisés dans les connexions simultanées peut entraîner une boucle connexion-déconnexion infinie. Cela peut se produire si un autre périphérique est codé de manière irréversible pour utiliser le nom du périphérique principal comme ID client dans les connexions. Pour plus d'informations, consultez cette étape de dépannage.

Les appareils Greengrass sont également entièrement intégrés au service d'indexation de flotte d'AWS IoT Device Management. Cela vous permet d'indexer et de rechercher des appareils en fonction de leurs attributs, de leur état shadow et de leur état de connexion dans le cloud. Par exemple, les appareils Greengrass établissent au moins une connexion qui utilise le nom d'objet comme ID client, afin que vous puissiez utiliser l'indexation de connectivité des appareils pour détecter quels appareils Greengrass sont actuellement connectés à AWS IoT Core ou déconnectés de celui-ci. Pour plus d'informations, consultez la section Service d'indexation de flotte dans le Guide du AWS IoT développeur.

Configuration du port MQTT pour la messagerie locale

Cette fonctionnalité nécessite AWS IoT Greengrass Core v1.10 ou version ultérieure.

Le noyau de Greengrass agit en tant que courtier de messages local pour la messagerie MQTT entre les fonctions Lambda locales, les connecteurs et les appareils clients. Par défaut, le noyau utilise le port 8883 pour le trafic MQTT sur le réseau local. Vous pouvez modifier le port pour éviter un conflit avec d'autres logiciels s'exécutant sur le port 8883.

Pour configurer le numéro de port utilisé par le noyau pour le trafic MQTT local

1. Exécutez la commande suivante pour arrêter le daemon Greengrass :

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

2. Ouvrez *greengrass-root*`/config/config.json` pour le modifier en tant qu'utilisateur su.

3. Dans l'objet `coreThing`, ajoutez la propriété `ggMqttPort` et définissez la valeur sur le numéro de port que vous souhaitez utiliser. Les valeurs valides vont de 1 024 à 65 535. L'exemple suivant définit le numéro de port sur `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. Lancez le démon.

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

5. Si la détection IP automatique est activée pour le noyau, la configuration est terminée.

   Si la détection IP automatique n'est pas activée, vous devez mettre à jour les informations de connectivité du noyau. Cela permet aux appareils clients de recevoir le numéro de port correct lors des opérations de découverte afin d'acquérir des informations de connectivité de base. Vous pouvez utiliser la AWS IoT console ou l'AWS IoT GreengrassAPI pour mettre à jour les informations de connectivité de base. Pour cette procédure, vous mettez uniquement à jour le numéro de port. L'adresse IP locale du noyau reste la même.

Pour mettre à jour les informations de connectivité du noyau (console)

1. Sur la page de configuration du groupe, choisissez le noyau Greengrass.
2. Sur la page des détails de base, choisissez l'onglet Points de terminaison du broker MQTT.
3. Choisissez Gérer les points de terminaison, puis choisissez Ajouter un point de terminaison
4. Entrez votre adresse IP locale actuelle et le nouveau numéro de port. L'exemple suivant définit le numéro de port `9000` pour l'adresse IP `192.168.1.8`.
5. Supprimez le point de terminaison obsolète, puis choisissez Mettre à jour

Pour mettre à jour les informations de connectivité du noyau (API)

- Utilisez l'action UpdateConnectivityInfo. L'exemple suivant utilise `update-connectivity-info` dans l'AWS CLI pour définir le numéro de port `9000` de l'adresse 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**
>
> Vous pouvez également configurer le port utilisé par le noyau pour la messagerie MQTT avec AWS IoT Core. Pour plus d'informations, consultez the section called “Connexion au port 443 ou via un proxy réseau”.

Délai d'expiration pour les opérations de publication, d'abonnement et de désinscription dans le cadre des connexions MQTT avec AWS Cloud

Cette fonctionnalité est disponible dans AWS IoT Greengrass version 1.10.2 ou ultérieure.

Vous pouvez configurer le temps (en secondes) alloué au noyau Greengrass pour terminer une opération de publication, d'abonnement ou de désabonnement dans des connexions MQTT vers AWS IoT Core. Vous pouvez modifier ce paramètre si les opérations dépassent le délai d'attente en raison de contraintes de bande passante ou d'une latence élevée. Pour configurer ce paramètre dans le fichier config.json, ajoutez ou modifiez la propriété `mqttOperationTimeout` dans l'objet `coreThing`. Par exemple :

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

Le délai d'attente par défaut est de 5 secondes. Le délai d'attente minimum est de 5 secondes.

Activation de la la détection IP automatique

Vous pouvez configurer AWS IoT Greengrass pour permettre aux appareils clients d'un groupe Greengrass de découvrir automatiquement le noyau de Greengrass. Lorsque cette option est activée, le cœur surveille les modifications apportées à ses adresses IP. Si une adresse change, le noyau publie une liste d'adresses mise à jour. Ces adresses sont mises à la disposition des appareils clients appartenant au même groupe Greengrass que le noyau.

> **Note**
>
> La AWS IoT politique relative aux appareils clients doit `greengrass:Discover` autoriser les appareils à récupérer des informations de connectivité pour le cœur. Pour de plus amples informations sur cette instruction de stratégie, veuillez consulter the section called “Acvery Service Service Service”.

Pour activer cette fonctionnalité depuis la AWS IoT Greengrass console, choisissez Détection automatique lorsque vous déployez votre groupe Greengrass pour la première fois. Vous pouvez également activer ou désactiver cette fonctionnalité sur la page de configuration du groupe en

choisissant l'onglet Fonctions Lambda et en sélectionnant le détecteur IP. La détection automatique des adresses IP est activée si l'option Détecter et remplacer automatiquement les points de terminaison du broker MQTT est sélectionnée.

Pour gérer la découverte automatique avec l'AWS IoT GreengrassAPI, vous devez configurer la fonction Lambda `IPDetector` du système. La procédure suivante montre comment utiliser la commande create-function-definition-versionCLI pour configurer la découverte automatique du noyau de Greengrass.

1. Obtenez les ID du groupe et de la version de groupe Greengrass cible. Cette procédure suppose qu'il s'agit de la dernière version du groupe et du groupe. La requête suivante renvoie le dernier groupe créé.

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

   Vous pouvez également procéder à une interrogation par nom. Les noms de groupe ne devant pas nécessairement être uniques, plusieurs groupes peuvent être renvoyés.

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

   > **Note**
   >
   > Vous pouvez également trouver ces valeurs dans la AWS IoT console. L'ID du groupe s'affiche sur la page Paramètres du groupe. Les identifiants de version du groupe sont affichés dans l'onglet Déploiements du groupe.

2. Copiez les valeurs `LatestVersion` et `Id` du groupe cible dans la sortie.
3. Obtenez la version de groupe la plus récente.

   - Remplacez *`group-id`* par la propriété `Id` que vous avez copiée.
   - Remplacez *`latest-group-version-id`* par l'`LatestVersion` que vous avez copié.

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

4. À partir de l'objet `Definition` de la sortie, copiez le `CoreDefinitionVersionArn` et les ARN de tous les autres composants de groupe à l'exception de `FunctionDefinitionVersionArn`. Vous utilisez ces valeurs lorsque vous créez une nouvelle version de groupe.
5. Depuis `FunctionDefinitionVersionArn` dans la sortie, copiez l'ID de la définition de fonction et la version de la définition de fonction :

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

   > **Note**
   >
   > Si vous le souhaitez, vous pouvez créer une définition de fonction en exécutant la commande `create-function-definition`, puis copier l'ID à partir de la sortie.

6. Utilisez la commande `get-function-definition-version` pour obtenir l'état de la définition actuelle. Utilisez celui *`function-definition-id`*que vous avez copié pour définir la fonction. Par exemple, *`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
   ```

   Notez les configurations de la fonction répertoriée. Vous devez inclure ces valeurs lorsque vous créez une nouvelle version de la définition de fonction. Vous évitez ainsi de perdre les paramètres de votre définition actuelle.
7. Ajoutez une version de définition de fonction à la définition de fonction.

   - *`function-definition-id`*Remplacez-le par celui `Id` que vous avez copié pour la définition de la fonction. Par exemple, *`4d941bc7-92a1-4f45-8d64 EXAMPLEf76c3`*.
   - *`arbitrary-function-id`*Remplacez-le par un nom pour la fonction, tel que**`auto-detection-function`**.
   - Ajoutez au `functions` tableau toutes les fonctions Lambda que vous souhaitez inclure dans cette version, telles que celles répertoriées à l'étape précédente.

```
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. Copiez l'`Arn` de la version de définition de fonction à partir de la sortie.
9. Créez une version de groupe contenant la fonction Lambda du système.

   - Remplacez *`group-id`* par l'`Id` du groupe.
   - *`core-definition-version-arn`*Remplacez-le par celui `CoreDefinitionVersionArn` que vous avez copié à partir de la dernière version du groupe.
   - *`function-definition-version-arn`*Remplacez-le par celui `Arn` que vous avez copié pour la nouvelle version de définition de fonction.
   - Remplacez les ARN des autres composants de groupe (par exemple `SubscriptionDefinitionVersionArn` ou `DeviceDefinitionVersionArn`) que vous avez copiés de la version de groupe la plus récente.
   - Supprimez tous les paramètres inutilisés. Par exemple, supprimez `--resource-definition-version-arn` si votre version de groupe ne contient aucune ressource.

```
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. Copiez la `Version` à partir de la sortie. Il s'agit de l'ID de la nouvelle version de groupe.
11. Déployez le groupe avec la nouvelle version de groupe.

    - Remplacez *`group-id`* par l'`Id` que vous avez copié pour le groupe.

- *group-version-id*Remplacez-le par celui `Version` que vous avez copié pour la nouvelle version du groupe.

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

Si vous souhaitez entrer manuellement l'adresse IP de votre noyau Greengrass, vous pouvez effectuer ce didacticiel avec une autre définition de fonction, qui n'inclut pas la fonction `IPDetector`. Cela empêchera la fonction de détection de localiser et de saisir automatiquement votre adresse IP principale de Greengrass.

Cette fonction Lambda du système n'est pas visible dans la console Lambda. Lorsque la fonction est ajoutée à la version de groupe la plus récente, elle est incluse dans les déploiements que vous effectuez à partir de la console, sauf si vous utilisez l'API pour la remplacer ou la supprimer.

Configuration du système d'initialisation pour le lancement du démon Greengrass

Il est recommandé de configurer votre système d'initialisation de manière à lancer le démon Greengrass au démarrage système, en particulier lorsque vous gérez d'importantes flottes d'appareils.

> **Note**
>
> Si vous avez utilisé `apt` pour installer le logiciel AWS IoT Greengrass Core, vous pouvez utiliser les scripts systemd pour activer le lancement au démarrage. Pour plus d'informations, consultez the section called “Utiliser des scripts systemd pour gérer le cycle de vie du démon Greengrass”.

Il existe différents types de système d'initialisation, par exemple initd, systemd et SystemV, qui utilisent des paramètres de configuration similaires. L'exemple suivant est un fichier de service pour systemd. Le paramètre `Type` est défini sur `forking`, car greengrassd (utilisé pour lancer Greengrass) duplique le processus du démon Greengrass, et le paramètre `Restart` est défini

sur `on-failure` pour demander à systemd de relancer Greengrass si celui-ci entre dans un état d'échec.

> **Note**
>
> Pour voir si votre appareil utilise systemd, exécutez le script `check_ggc_dependencies` comme décrit dans Module 1. Ensuite, pour utiliser systemd, assurez-vous que le paramètre `useSystemd` dans `config.json` est défini sur `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
```

Consultez aussi

- Qu'est-ce qu'AWS IoT Greengrass ?
- the section called “Exigences et plateformes prises en charge”
- Commencer avec AWS IoT Greengrass
- the section called “Présentation du modèle d'objet de groupe”
- the section called “Intégration de sécurité matérielle”

AWS IoT Greengrass Version 1politique de maintenance

Utilisez cette politique de AWS IoT Greengrass V1 maintenance pour comprendre les différents niveaux de maintenance et de mise à jour du AWS IoT Greengrass V1 service et du logiciel AWS IoT Greengrass Core v1.x.

Rubriques

AWS IoT Greengrassschéma de version

AWS IoT Greengrassutilise le versionnement sémantique pour le logiciel AWS IoT Greengrass Core. Les versions sémantiques suivent une majeure. mineur. système de numéro de patch. La version principale s'incrémente en cas de modifications fonctionnelles et d'API qui ne sont pas rétrocompatibles avec les versions majeures précédentes. La version mineure augmente pour les versions qui ajoutent de nouvelles fonctionnalités rétrocompatibles. La version du correctif augmente pour les correctifs de sécurité ou les corrections de bogues. Depuis sa première version majeure, la v1.0.0 AWS IoT Greengrass a publié 11 versions mineures du logiciel AWS IoT Greengrass Core v1.x, la v1.11.6 étant la dernière version. Nous vous recommandons de mettre à jour votre logiciel AWS IoT Greengrass Core vers la dernière version disponible afin de tirer parti des nouvelles fonctionnalités, des améliorations et des corrections de bogues.

En décembre 2020, AWS IoT Greengrass a publié sa première mise à jour de version majeure. Cette mise à jour incluait le AWS IoT Greengrass V2 service et la version 2.0.3 du logiciel AWS IoT Greengrass Core. Pour les nouvelles applications, nous vous recommandons vivement d'utiliser AWS IoT Greengrass Version 2 le logiciel AWS IoT Greengrass Core v2.x. La version 2 reçoit de nouvelles fonctionnalités, inclut toutes les fonctionnalités clés de la V1 et prend en charge des plateformes supplémentaires et des déploiements continus sur de vastes flottes d'appareils. Pour plus d'informations, consultez Qu'est-ce qu'AWS IoT Greengrass V2 ?.

Phases du cycle de vie des versions majeures du logiciel de AWS IoT Greengrass base

Chaque version majeure du logiciel AWS IoT Greengrass Core comporte les trois phases de cycle de vie séquentielles suivantes. Chaque phase du cycle de vie fournit différents niveaux de maintenance sur une période postérieure à la date de sortie initiale.

- Phase de publication : les mises à jour suivantes AWS IoT Greengrass peuvent être publiées :
  - Mises à jour de version mineures qui fournissent de nouvelles fonctionnalités ou des améliorations aux fonctionnalités existantes
  - Mises à jour des versions de correctifs fournissant des correctifs de sécurité et des corrections de bogues
- Phase de maintenance : AWS IoT Greengrass peut publier des mises à jour de version de correctif fournissant des correctifs de sécurité et des corrections de bogues. AWS IoT Greengrassne publiera pas de nouvelles fonctionnalités ou n'améliorera pas les fonctionnalités existantes pendant la phase de maintenance.
- Phase de durée de vie prolongée : AWS IoT Greengrass ne publiera pas de mises à jour fournissant des fonctionnalités, des améliorations aux fonctionnalités existantes, des correctifs de sécurité ou des corrections de bogues. Toutefois, les AWS Cloud terminaux et les opérations d'API resteront disponibles et fonctionneront conformément au contrat de niveau AWS IoT Greengrass de service. Les appareils qui exécutent le logiciel AWS IoT Greengrass Core v1.x peuvent continuer à se connecter AWS Cloud et à fonctionner.

Une fois la phase de vie prolongée terminée pour une version majeure deAWS IoT Greengrass, les AWS Cloud points de terminaison et les opérations d'API seront obsolètes et ne seront plus disponibles. Les appareils qui exécutent le logiciel AWS IoT Greengrass Core v1.x ne pourront pas se connecter aux AWS Cloud services pour fonctionner.

Politique de maintenance pour le logiciel AWS IoT Greengrass Core

Le logiciel AWS IoT Greengrass Core v1.x est entré dans la phase de durée de vie prolongée le 30 juin 2023. Après cette date, le logiciel AWS IoT Greengrass Core v1.x restera en phase de durée de vie prolongée jusqu'à nouvel ordre.

Le logiciel AWS IoT Greengrass Core v2.x est actuellement en phase de lancement, et il le restera jusqu'à nouvel ordre. AWS IoT Greengrasscontinue d'ajouter de nouvelles fonctionnalités et améliorations au logiciel AWS IoT Greengrass Core v2.x. Par exemple, AWS IoT Greengrass a publié le support Windows dans la version 2.5.0 du logiciel AWS IoT Greengrass Core. AWS IoT Greengrasspublie des correctifs de sécurité et des corrections de bogues pour toutes les versions mineures de AWS IoT Greengrass Core v2.x pendant au moins un an après la date de sortie. Pour plus d'informations, consultez la section Nouveautés de AWS IoT Greengrass V2.

Calendrier de la phase de maintenance

Le 30 juin 2023, la phase de maintenance s'est terminée pour le logiciel AWS IoT Greengrass Core v1.11.x. Le 31 mars 2022, la phase de maintenance s'est terminée pour le logiciel AWS IoT Greengrass Core v1.10.x. La phase de maintenance prend fin pour certains artefacts et fonctionnalités du logiciel AWS IoT Greengrass Core v1.x avant ces dates. Pour plus d'informations, veuillez consulter Fin du programme de maintenance.

Si vous avez un AWS Support plan, la phase de maintenance du logiciel AWS IoT Greengrass Core v1.x n'affecte pas votre AWS Support plan. Vous pouvez continuer à ouvrir des AWS Support tickets même après la fin de la phase de maintenance. Si vous avez des questions ou des inquiétudes, contactez votre AWS Support contact ou posez une question sur AWSRe:post en utilisant le AWS IoT Greengrasstag.

Calendrier d'obsolescence

Actuellement, il n'est pas prévu d'arrêter le support du logiciel AWS IoT Greengrass Core v1.x. Les AWS IoT Greengrass V1 points de terminaison et les opérations de l'API resteront disponibles jusqu'à nouvel ordre. Le logiciel AWS IoT Greengrass Core v1.11.6 est entré dans la phase de durée de vie prolongée le 30 juin 2023. Au cours de cette phase, les appareils qui exécutent le logiciel AWS IoT Greengrass Core v1.x peuvent continuer à se connecter au AWS IoT Greengrass V1 service pour fonctionner jusqu'à nouvel ordre.

Si le AWS IoT Greengrass V1 support cesse d'être pris en charge à l'avenir, AWS IoT Greengrass nous fournirons un préavis de 12 mois avant que cela ne se produise. Cela vous aidera à planifier la mise à jour de vos applications à utiliser AWS IoT Greengrass V2 et du logiciel AWS IoT Greengrass Core v2.x. Pour plus d'informations sur la mise à jour de vos applications vers la version V2, voir Passer de la version V2 AWS IoT Greengrass V1 à la version 2.

Politique de support pour les AWS Lambda fonctions des appareils principaux de Greengrass

AWS IoT Greengrassvous permet d'exécuter AWS Lambda des fonctions sur des appareils IoT. AWS Lambdafournit une politique de support et des délais qui déterminent la prise en charge des environnements d'exécution Lambda dans. AWS IoT Greengrass Une fois qu'un environnement d'exécution Lambda atteint la fin de la phase de support, le support de cet environnement d'exécution prend AWS IoT Greengrass également fin. Pour plus d'informations, consultez la politique de support en matière d'exécution dans le guide du AWS Lambda développeur.

Lorsque le support d'un environnement d'exécution Lambda atteint la fin du support, vous ne pouvez pas créer ou mettre à jour les fonctions Lambda qui utilisent cet environnement d'exécution. Cependant, vous pouvez continuer à déployer ces fonctions Lambda sur les appareils principaux de Greengrass et appeler les fonctions Lambda déployées. Cette politique s'applique également àAWS IoT Greengrass V2.

Politique de prise en charge d’AWS IoT Device Tester pour AWS IoT Greengrass V1

AWS IoTDevice Tester (IDT) pour vous AWS IoT Greengrass V1 permet de valider et de qualifier vos AWS IoT Greengrass appareils en vue de leur inclusion dans le catalogue d'AWS Partnerappareils. Depuis le 4 avril 2022, AWS IoT Device Tester (IDT) AWS IoT Greengrass V1 ne génère plus de rapports de qualification signés. Vous ne pouvez plus qualifier de nouveaux AWS IoT Greengrass V1 appareils pour les répertorier dans le catalogue des AWS Partner appareils dans le cadre du programme de qualification des AWS appareils. Bien que vous ne puissiez pas qualifier les appareils Greengrass V1, vous pouvez continuer à utiliser IDT AWS IoT Greengrass V1 pour tester vos appareils Greengrass V1. Nous vous recommandons d'utiliser IDT pour qualifier et AWS IoT Greengrass V2 répertorier les appareils Greengrass dans AWS Partnerle catalogue des appareils. Pour plus d'informations, veuillez consulter Politique de prise en charge d’AWS IoT Device Tester pour AWS IoT Greengrass V1.

Fin du programme de maintenance

Le tableau suivant répertorie les dates de fin de maintenance des artefacts et fonctionnalités de AWS IoT Greengrass Core v1.x. Si vous avez des questions concernant le calendrier ou la politique de maintenance, contactez le AWSSupport.

Artifact ou fonctionnalité	Date de fin de maintenance
Installation du dépôt Greengrass APT	11 février 2022
Connecteur de classification d'images ML	31 mars 2022
Connecteur de détection d'objets ML	31 mars 2022
Connecteur ML Feedback	31 mars 2022
Connecteur AWS IoT Analytics	31 mars 2022
Connecteur de notifications Twilio	31 mars 2022
Connecteur d'intégration Splunk	31 mars 2022
Connecteur Serial Stream	31 mars 2022
ServiceNow MetricBase Connecteur d'intégration	31 mars 2022
Connecteur GPIO Raspberry Pi	31 mars 2022
AWS IoT GreengrassLogiciel de base v1.10.x	31 mars 2022
AWS IoT GreengrassImages Docker du logiciel de base v1.x	30 juin 2022
AWS IoT GreengrassLogiciel de base v1.11.x	30 juin 2023
AWS IoT GreengrassLogiciel de base v1.11.x Snap	31 décembre 2023

Fin de la maintenance des images Docker du logiciel AWS IoT Greengrass Core v1.x

Le 30 juin 2022, AWS IoT Greengrass fin de la maintenance des images Docker du logiciel AWS IoT Greengrass Core v1.x publiées sur Amazon Elastic Container Registry (Amazon ECR) et Docker Hub. Vous pouvez continuer à télécharger ces images Docker depuis Amazon ECR et Docker Hub

jusqu'au 30 juin 2023, soit un an après la fin de la maintenance. Cependant, les images Docker du logiciel AWS IoT Greengrass Core v1.x ne reçoivent plus de correctifs de sécurité ni de corrections de bogues après la fin de la maintenance le 30 juin 2022. Si vous exécutez une charge de travail de production qui dépend de ces images Docker, nous vous recommandons de créer vos propres images Docker à l'aide des Dockerfiles fournis. AWS IoT Greengrass Pour plus d'informations, veuillez consulter Logiciel AWS IoT Greengrass Docker.

Fin de la maintenance du référentiel APT du logiciel AWS IoT Greengrass Core v1.x

Le 11 février 2022, la maintenance de l'option d'installation du logiciel AWS IoT Greengrass Core v1.x à partir d'un référentiel APT a AWS IoT Greengrass pris fin. Le dépôt APT a été supprimé à cette date, vous ne pouvez donc plus l'utiliser pour mettre à jour le logiciel AWS IoT Greengrass Core ou installer le logiciel AWS IoT Greengrass Core sur de nouveaux appareils. Sur les appareils sur lesquels vous avez ajouté le AWS IoT Greengrass référentiel, vous devez le supprimer de la liste des sources. Nous vous recommandons de mettre à jour le logiciel AWS IoT Greengrass Core v1.x à l'aide de fichiers tar.

Fin de la maintenance du logiciel AWS IoT Greengrass Core v1.11.x Snap

Le 31 décembre 2023, la maintenance de la version logicielle AWS IoT Greengrass principale 1.11.x Snap publiée sur snapcraft.io AWS IoT Greengrass prendra fin. Les appareils utilisant actuellement le Snap continueront de fonctionner jusqu'à nouvel ordre. Cependant, le Snap AWS IoT Greengrass principal ne recevra plus de correctifs de sécurité ni de corrections de bogues une fois la maintenance terminée.

Commencer avec AWS IoT Greengrass

Ce didacticiel de mise en route comprend plusieurs modules conçus pour vous montrer AWS IoT Greengrass les bases et vous aider à commencer à utiliser AWS IoT Greengrass. Ce didacticiel couvre des concepts fondamentaux, notamment :

- Configuration des AWS IoT Greengrass cœurs et des groupes.
- Le processus de déploiement pour exécuter AWS Lambda des fonctions en périphérie.
- Connecter AWS IoT des appareils, appelés appareils clients, au AWS IoT Greengrass cœur.
- Création d'abonnements pour permettre la communication MQTT entre les fonctions Lambda locales, les appareils clients et. AWS IoT

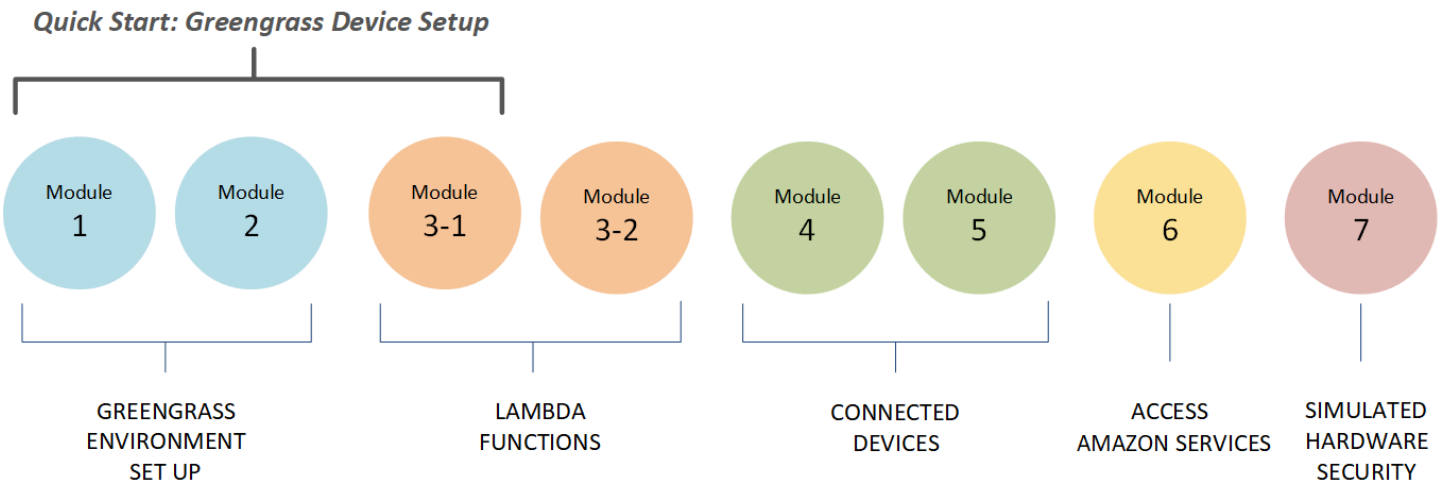
Choisissez comment commencer avec AWS IoT Greengrass

Vous pouvez choisir comment utiliser ce didacticiel pour configurer votre appareil principal (noyau) :

- Exécutez la configuration de l'appareil Greengrass sur votre appareil principal, ce qui vous permet de passer de l'installation des AWS IoT Greengrass dépendances au test d'une fonction Lambda Hello World en quelques minutes. Ce script reproduit les étapes du Module 1 au Module 3-1.

  - ou -

- Parcourez les étapes du Module 1 au Module 3-1 pour examiner de plus près les exigences et les processus de Greengrass. Ces étapes permettent de configurer votre appareil principal, de créer et de configurer un groupe Greengrass contenant une fonction Lambda Hello World et de déployer votre groupe Greengrass. Généralement, cela prend une heure ou deux heures.



Quick Start

La section Configuration de l'appareil Greengrass permet de configurer votre appareil principal et les ressources Greengrass. Le script :

- Installe les AWS IoT Greengrass dépendances.
- Télécharge le certificat d'autorité de certification (CA) racine, ainsi que le certificat et les clés de l'appareil principal.
- Télécharge, installe et configure le logiciel AWS IoT Greengrass Core sur votre appareil.
- Démarre le processus de démon Greengrass sur l'appareil principal (noyau).
- Crée ou met à jour le rôle de service Greengrass, si nécessaire.
- Crée un groupe Greengrass et un appareil principal Greengrass.
- (Facultatif) Crée une fonction Lambda Hello World, un abonnement et une configuration de journalisation locale.
- (Facultatif) Déploie le groupe Greengrass.

Modules 1 et 2

LeModule 1 et le Module 2 décrivent comment configurer votre environnement. (Ou utilisez la section Configuration de l'appareil Greengrass pour que ces modules soient exécutés pour vous.)

- Configurer votre appareil principal pour Greengrass.
- Exécuter le script de vérification des dépendances.
- Créer un groupe Greengrass et un noyau Greengrass.
- Téléchargez et installez la dernière version du logiciel AWS IoT Greengrass Core à partir d'un fichier tar.gz.

- Démarrer le processus de démon Greengrass sur le noyau.

> **Note**
>
> AWS IoT Greengrass fournit également d'autres options pour installer le logiciel de AWS IoT Greengrass base, y compris les `apt` installations sur les plateformes Debian prises en charge. Pour plus d'informations, consultez the section called "Installer le logiciel AWS IoT Greengrass Core".

Modules 3-1 et 3-2

Les modules 3-1 et 3-2 décrivent comment utiliser les fonctions Lambda locales. (Ou utilisez la section Configuration de l'appareil Greengrass pour exécuter le Module 3-1 pour vous.)

- Créez des fonctions Lambda Hello World dans. AWS Lambda
- Ajoutez des fonctions Lambda à votre groupe Greengrass.
- Créez des abonnements qui permettent la communication MQTT entre les fonctions Lambda et. AWS IoT
- Configurez la journalisation locale pour les composants du système Greengrass et les fonctions Lambda.
- Déployez un groupe Greengrass contenant vos fonctions Lambda et vos abonnements.
- Envoyez des messages depuis les fonctions Lambda locales à. AWS IoT
- Invoquez des fonctions Lambda locales depuis. AWS IoT
- Tester des fonctions à la demande et longue durée.

Modules 4 et 5

Le module 4 montre comment les appareils clients se connectent au cœur et communiquent entre eux.

Le module 5 montre comment les appareils clients peuvent utiliser les ombres pour contrôler l'état.

- Enregistrez et approvisionnez AWS IoT les appareils (représentés par des terminaux en ligne de commande).
- Installez le Kit SDK des appareils AWS IoT pour Python. Ceci est utilisé par les appareils clients pour découvrir le noyau de Greengrass.
- Ajoutez les appareils clients à votre groupe Greengrass.

- Créer des abonnements qui autorisent les communications MQTT.
- Déployez un groupe Greengrass contenant vos appareils clients.
- Testez device-to-device la communication.
- Tester les mises à jour d'état de shadow.

Module 6

Le module 6 vous montre comment les fonctions Lambda peuvent accéder au. AWS Cloud

- Créez un rôle de groupe Greengrass qui permet d'accéder aux ressources Amazon DynamoDB.
- Ajoutez une fonction Lambda à votre groupe Greengrass. Cette fonction utilise le AWS SDK pour Python pour interagir avec DynamoDB.
- Créer des abonnements qui autorisent les communications MQTT.
- Testez l'interaction avec DynamoDB.

Module 7

Le Module 7 montre comment configurer un module de sécurité matérielle (HSM) simulé à utiliser avec un noyau Greengrass.

> **Important**
>
> Ce module avancé est fourni uniquement pour l'expérimentation et les tests initiaux. Il n'est pas destiné à une utilisation en production quelle qu'elle soit.

- Installer et configurer un HSM basé sur le logiciel et une clé privée.
- Configurer le noyau Greengrass pour utiliser la sécurité matérielle.
- Tester la configuration de sécurité matérielle.

Prérequis

Pour suivre ce didacticiel, vous aurez besoin des éléments suivants :

- Un Mac, un PC Windows ou un ordinateur UNIX ou similaire.
- Un Compte AWS. Si vous n'en avez pas, veuillez consulter the section called “Créez un Compte AWS”.

- L'utilisation d'une AWS région qui soutient AWS IoT Greengrass. Pour obtenir la liste des régions prises en charge pour AWS IoT Greengrass, voir les AWS points de terminaison et les quotas dans le Références générales AWS.

  > **Note**
  >
  > Prenez note de votre nom Région AWS et assurez-vous qu'il est régulièrement utilisé tout au long de ce didacticiel. Si vous changez de nom Région AWS pendant le didacticiel, vous risquez de rencontrer des problèmes lors de l'exécution des étapes.

- Un Raspberry Pi 4 modèle B, ou un Raspberry Pi 3 modèle B/B+, avec une carte microSD de 8 Go, ou une instance Amazon EC2. Dans la mesure où AWS IoT Greengrass doit dans l'idéal être utilisé avec un matériel physique, nous vous recommandons d'utiliser une carte Raspberry Pi.

  > **Note**
  >
  > Pour obtenir le modèle de votre Raspberry Pi, exécutez la commande suivante :
  >
  > ```
  > cat /proc/cpuinfo
  > ```
  >
  > Dans le bas de la liste, notez la valeur de l'attribut `Revision`, puis consultez le tableau Which Pi have I got?. Par exemple, si la valeur `Revision` indique `a02082`, le tableau confirme qu'il s'agit d'un appareil Pi 3 modèle B.
  > Pour déterminer l'architecture de votre Raspberry Pi, exécutez la commande suivante :
  >
  > ```
  > uname -m
  > ```
  >
  > Pour ce didacticiel, le résultat doit être supérieur ou égal à `armv71`.

- Connaissances de base de Python.

Bien que ce didacticiel soit destiné à fonctionner AWS IoT Greengrass sur un Raspberry Pi, il est AWS IoT Greengrass également compatible avec d'autres plateformes. Pour plus d'informations, consultez the section called “Exigences et plateformes prises en charge”.

Créez un Compte AWS

Si vous n'en avez pas Compte AWS, procédez comme suit pour créer et activer un Compte AWS :

Inscrivez-vous pour un Compte AWS

Si vous n'en avez pas Compte AWS, procédez comme suit pour en créer un.

Pour vous inscrire à un Compte AWS

1. Ouvrez https://portal.aws.amazon.com/billing/signup.
2. Suivez les instructions en ligne.

   Dans le cadre de la procédure d'inscription, vous recevrez un appel téléphonique et vous saisirez un code de vérification en utilisant le clavier numérique du téléphone.

   Lorsque vous vous inscrivez à un Compte AWS, un Utilisateur racine d'un compte AWSest créé. Par défaut, seul l'utilisateur racine a accès à l'ensemble des Services AWS et des ressources de ce compte. Pour des raisons de sécurité, attribuez un accès administratif à un utilisateur et utilisez uniquement l'utilisateur root pour effectuer les tâches nécessitant un accès utilisateur root.

AWS vous envoie un e-mail de confirmation une fois le processus d'inscription terminé. Vous pouvez afficher l'activité en cours de votre compte et gérer votre compte à tout moment en accédant à https://aws.amazon.com/ et en choisissant Mon compte.

Création d'un utilisateur doté d'un accès administratif

Une fois que vous vous êtes inscrit à un utilisateur administratif Compte AWS, que vous Utilisateur racine d'un compte AWS l'avez sécurisé AWS IAM Identity Center, que vous l'avez activé et que vous en avez créé un, afin de ne pas utiliser l'utilisateur root pour les tâches quotidiennes.

Sécurisez votre Utilisateur racine d'un compte AWS

1. Connectez-vous en AWS Management Consoletant que propriétaire du compte en choisissant Utilisateur root et en saisissant votre adresse Compte AWS e-mail. Sur la page suivante, saisissez votre mot de passe.

Pour obtenir de l‘aide pour vous connecter en utilisant l‘utilisateur racine, consultez Connexion en tant qu‘utilisateur racine dans le Guide de l‘utilisateur Connexion à AWS .

2. Activez l‘authentification multifactorielle (MFA) pour votre utilisateur racine.

   Pour obtenir des instructions, voir Activer un périphérique MFA virtuel pour votre utilisateur Compte AWS root (console) dans le guide de l'utilisateur IAM.

Création d'un utilisateur doté d'un accès administratif

1. Activez IAM Identity Center.

   Pour obtenir des instructions, consultez Activation d’ AWS IAM Identity Center dans le Guide de l’utilisateur AWS IAM Identity Center .

2. Dans IAM Identity Center, accordez un accès administratif à un utilisateur.

   Pour un didacticiel sur l'utilisation du Répertoire IAM Identity Center comme source d'identité, voir Configurer l'accès utilisateur par défaut Répertoire IAM Identity Center dans le Guide de AWS IAM Identity Center l'utilisateur.

Connectez-vous en tant qu'utilisateur disposant d'un accès administratif

- Pour vous connecter avec votre utilisateur IAM Identity Center, utilisez l‘URL de connexion qui a été envoyée à votre adresse e-mail lorsque vous avez créé l‘utilisateur IAM Identity Center.

  Pour obtenir de l'aide pour vous connecter en utilisant un utilisateur d'IAM Identity Center, consultez la section Connexion au portail AWS d'accès dans le guide de l'Connexion à AWS utilisateur.

Attribuer l'accès à des utilisateurs supplémentaires

1. Dans IAM Identity Center, créez un ensemble d'autorisations conforme aux meilleures pratiques en matière d'application des autorisations du moindre privilège.

   Pour obtenir des instructions, voir Création d'un ensemble d'autorisations dans le guide de AWS IAM Identity Center l'utilisateur.

2. Affectez des utilisateurs à un groupe, puis attribuez un accès d'authentification unique au groupe.

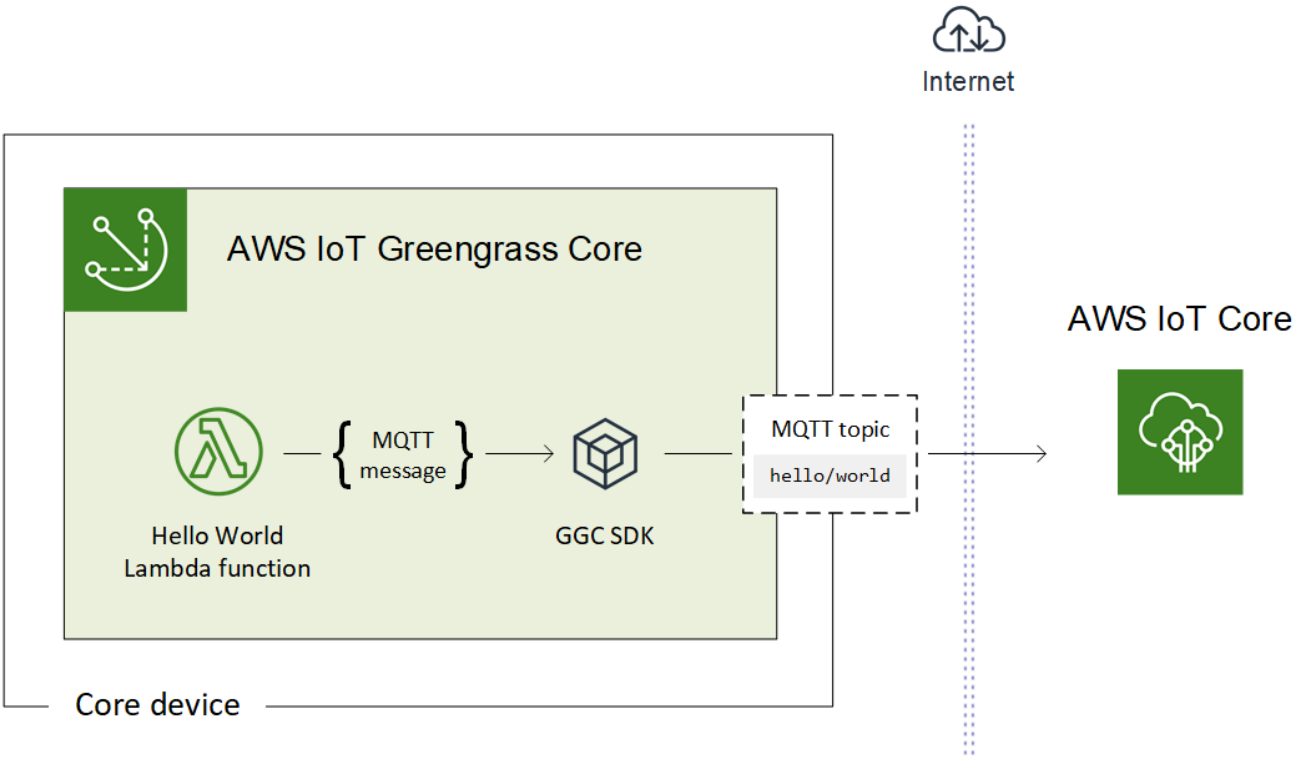
Pour obtenir des instructions, voir Ajouter des groupes dans le guide de AWS IAM Identity Center l'utilisateur.

> **Important**
>
> Pour ce didacticiel, nous partons du principe que votre compte utilisateur IAM dispose d'autorisations d'accès administrateur.

Démarrage rapide : Configuration de l'appareil Greengrass

Greengrass Device Setup est un script qui configure votre appareil principal en quelques minutes, afin que vous puissiez commencer à l'utiliserAWS IoT Greengrass. Utilisez ce script pour :

1. Configurez votre appareil et installez le logicielAWS IoT Greengrass Core.
2. Configurez vos ressources basées sur le cloud.
3. Déployez éventuellement un groupe Greengrass avec une fonction Lambda Hello World qui envoie des messages MQTTAWS IoT depuis leAWS IoT Greengrass noyau. Cela permet de configurer Greengrass environnement présenté dans le diagramme suivant.



Prérequis

Voici les prérequis pour la configuration de l'appareil Greengrass :

- Votre appareil principal (noyau) doit utiliser une plateforme prise en charge. Un gestionnaire de packages approprié doit être installé sur l'appareil : `apt`, `yum` ou `opkg`.

- L'utilisateur Linux qui lance le script doit disposer des autorisations de s'exécuter en tant que `sudo`.

- Vous devez fournir vosCompte AWS informations d'identification. Pour plus d'informations, veuillez consulter the section called “Fournir desCompte AWS informations d'identification”.

> **Note**
>
> La configuration de l'appareil Greengrass installe la dernière version du logiciel AWS IoT Greengrass Core sur l'appareil. En installant le logiciel AWS IoT Greengrass Core, vous acceptez le contrat de licence du logiciel Greengrass Core.

Exécution de la configuration de l'appareil Greengrass

Vous pouvez exécuter la configuration de l'appareil Greengrass en seulement quelques étapes. Une fois que vous avez fourni vosCompte AWS informations d'identification, le script met en service votre appareil principal Greengrass et déploie un groupe Greengrass en quelques minutes. Exécutez les commandes suivantes dans une fenêtre de terminal sur l'appareil cible.

> **Note**
>
> Ces étapes vous montrent comment exécuter le script en mode interactif. Celui-ci vous invite à saisir ou à accepter chaque valeur d'entrée. Pour de plus amples informations sur l'exécution silencieuse du script, veuillez consulter the section called “Exécution de la configuration de l'appareil Greengrass en mode silencieux”.

1. Fournissez vos informations d'identification. Dans cette procédure, nous supposons que vous fournissez des informations d'identification de sécurité temporaires en tant que variables d'environnement.

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

> **Note**
>
> Si vous exécutez la configuration de l'appareil Greengrass sur un Raspbian ou une OpenWrt plate-forme, créez une copie de ces commandes. Vous devez les fournir à nouveau après le redémarrage de l'appareil.

2. Téléchargez et démarrez le script. Vous pouvez utiliser `wget` ou `curl` pour télécharger le 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. Suivez les invites de commande pour les valeurs d'entrée. Vous pouvez appuyer sur la touche Entrée pour utiliser la valeur par défaut ou saisir une valeur personnalisée, puis appuyer sur Entrée.

   Le script écrit dans le terminal des messages d'état similaires à ce qui suit.

```
############## 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. Si votre appareil principal exécute Raspbian ou OpenWrt, redémarrez l'appareil lorsque vous y êtes invité, fournissez vos informations d'identification, puis redémarrez le script.

   a. Lorsque vous êtes invité à redémarrer l'appareil, exécutez l'une des commandes suivantes.

      Pour les plateformes Raspbian :

      ```
      sudo reboot
      ```

      Pour les OpenWrt plateformes :

      ```
      reboot
      ```

   b. Une fois l'appareil redémarré, ouvrez le terminal et fournissez vos informations d'identification en tant que variables d'environnement.

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

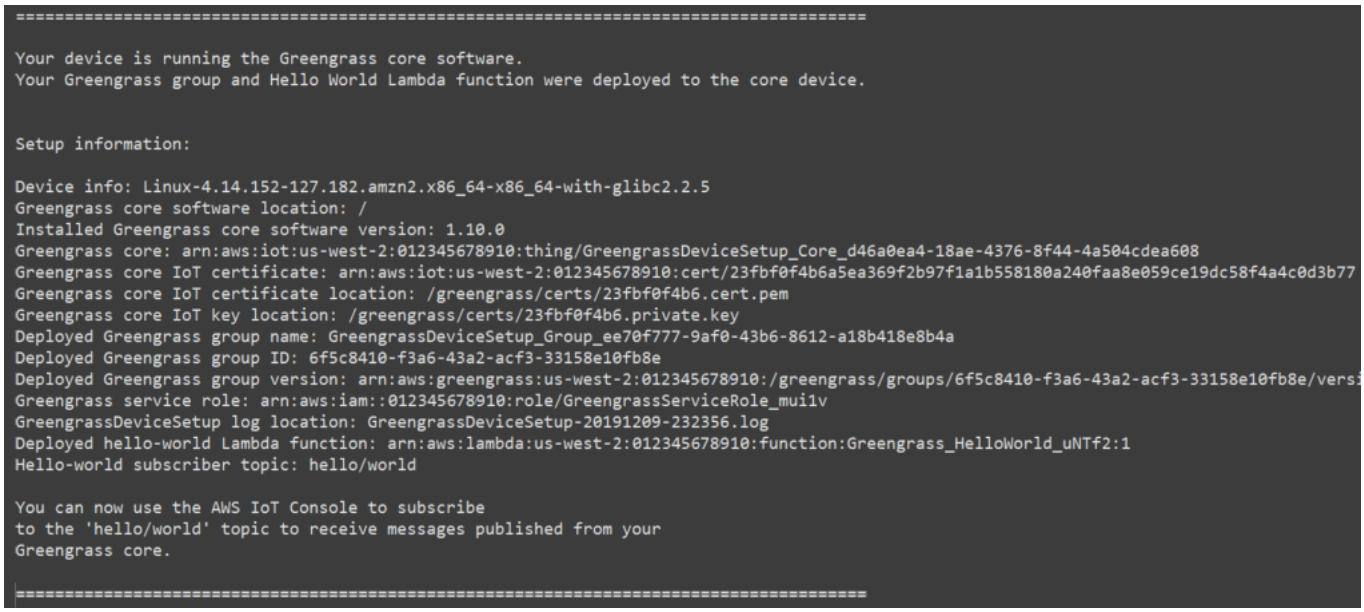
   c. Redémarrez le script.

      ```
      sudo -E ./gg-device-setup-latest.sh bootstrap-greengrass-interactive
      ```

   d. Lorsque vous êtes invité à utiliser vos valeurs d'entrée de la session précédente ou à démarrer une nouvelle installation, entrez `yes` pour réutiliser vos valeurs d'entrée.

> **Note**
>
> Sur les plateformes nécessitant un redémarrage, vos valeurs d'entrée de la session précédente, sauf les informations d'identification, sont stockées temporairement dans le fichier `GreengrassDeviceSetup.config.info`.

Lorsque la configuration est terminée, le terminal affiche un message de statut de réussite semblable au message suivant.



5. Passez en revue le nouveau groupe Greengrass que le script configure à l'aide des valeurs d'entrée que vous fournissez.

   a. Connectez-vous à votre AWS Management Consoleordinateur et ouvrez laAWS IoT console.

   > **Note**
   >
   > Assurez-vous que la valeurRégion AWS sélectionnée dans la console est la même que celle que vous avez utilisée pour configurer votre environnement Greengrass. Par défaut, il s'agit de la Ouest Ouest.

   b. Dans le volet de navigation, développez les appareils Greengrass, puis choisissez Groupes (V1) pour localiser le groupe nouvellement créé.

6. Si vous avez inclus la fonction Hello World Lambda, la configuration de l'appareil Greengrass déploie le groupe Greengrass sur votre appareil principal. Pour tester la fonction Lambda ou pour obtenir des informations sur la manière de supprimer la fonction Lambda du groupe,the section called “Vérification de l'exécution de la fonction Lambda sur l'appareil principal (noyau)” passez au module 3-1 du didacticiel de démarrage.

   > **Note**
   >
   > Assurez-vous que la valeurRégion AWS sélectionnée dans la console est la même que celle que vous avez utilisée pour configurer votre environnement Greengrass. Par défaut, il s'agit de la Ouest Ouest.

   Si vous n'avez pas inclus la fonction Lambda Hello World, vous pouvez créer votre propre fonction Lambda ou essayer d'autres fonctionnalités de Greengrass. Par exemple, vous pouvez ajouter le connecteur de déploiement d'application Docker à votre groupe et l'utiliser pour déployer des conteneurs Docker sur votre appareil principal.

Résolution des problèmes

Vous pouvez utiliser les informations suivantes pour résoudre les problèmes liés à la configuration de l'AWS IoT Greengrassappareil.

Erreur : Python (python3.7) introuvable. Tentative d'installation...

Solution : cette erreur peut s'afficher lorsque vous utilisez une instance Amazon EC2. Cette erreur se produit lorsque Python n'est pas installé dans le`/usr/bin/python3.7` dossier. Pour résoudre cette erreur, déplacez Python dans le répertoire approprié après l'avoir installé :

```
sudo ln -s /usr/local/bin/python3.7 /usr/bin/python3.7
```

Dépannage supplémentaire

Pour résoudre d'autres problèmes liés à la configuration de l'AWS IoT Greengrassappareil, vous pouvez rechercher des informations de débogage dans les fichiers journaux :

- Pour les problèmes liés à la configuration de l'appareil Greengrass, consultez le fichier `/tmp/greengrass-device-setup-bootstrap-`*`epoch-timestamp`*`.log`.
- Pour les problèmes liés au groupe Greengrass ou à la configuration de l'environnement principal, consultez le fichier `GreengrassDeviceSetup-`*`date`*`-`*`time`*`.log` dans le même répertoire que `gg-device-setup-latest.sh` ou à l'emplacement spécifié.

Pour plus d'aide à la résolution des problèmes, consultezRésolution des problèmes ou consultez le AWS IoT Greengrasstag surAWS Re:Post.

Options de configuration de l'appareil Greengrass

Vous configurez la configuration de l'appareil Greengrass pour accéder à vosAWS ressources et configurez votre environnement Greengrass.

Fournir desCompte AWS informations d'identification

La configuration de l'appareil Greengrass utilise vosCompte AWS informations d'identification pour accéder à vosAWS ressources. Il prend en charge les informations d'identification à long terme d'un utilisateur IAM ou les informations d'identification de sécurité temporaires d'un rôle IAM.

Tout d'abord, obtenez vos informations d'identification.

- Pour utiliser des informations d'identification à long terme, fournissez l'ID de clé d'accès et clé d'accès secrète de votre utilisateur IAM. Pour de plus amples informations sur la création de clés d'accès pour les informations d'identification à long terme, veuillez consulter Gestion des clés d'accès pour les utilisateurs IAM dans le Guide de l'utilisateur IAM.

- Pour utiliser des informations d'identification de sécurité temporaires (recommandé), fournissez l'ID de clé d'accès, clé d'accès secrète et jeton de session à partir d'un rôle IAM assumé. Pour plus d'informations sur l'extraction des informations d'identification de sécurité temporaires à partir de laAWS STS`assume-role` commande, consultez la section Utilisation des informations d'identification de sécurité temporaires avec leAWS CLI dans le guide de l'utilisateur IAM.

> **Note**
>
> Pour les besoins de ce didacticiel, nous partons du principe que l'utilisateur IAM ou le rôle IAM dispose d'autorisations d'accès administrateur.

Ensuite, fournissez vos informations d'identification à la configuration de l'appareil Greengrass selon une des deux manières suivantes :

- En tant que variables d'environnement. Définissez les variables d'environnement `AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY` et `AWS_SESSION_TOKEN` (le cas échéant) avant de démarrer le script, comme indiqué à l'étape 1 de the section called “Exécution de la configuration de l'appareil Greengrass”.

- En tant que valeurs d'entrée Entrez les valeurs de votre ID de clé d'accès, de votre clé d'accès secrète et de votre jeton de session (si nécessaire) directement dans le terminal après avoir démarré le script.

La configuration de l'appareil Greengrass n'enregistre pas ou ne stocke pas vos informations d'identification.

Fourniture de valeurs d'entrée

En mode interactif, la configuration de l'appareil Greengrass vous invite à saisir les valeurs d'entrée. Vous pouvez appuyer sur la touche Entrée pour utiliser la valeur par défaut ou saisir une valeur personnalisée, puis appuyer sur Entrée. En mode silencieux, vous fournissez des valeurs d'entrée après le démarrage du script.

Valeurs d'entrée

AWSID de clé d'accès

ID de clé d'accès provenant des informations d'identification de sécurité à long terme ou temporaires. Spécifiez cette option comme valeur d'entrée uniquement si vous ne fournissez pas vos informations d'identification en tant que variables d'environnement. Pour plus d'informations, veuillez consulter the section called “Fournir desCompte AWS informations d'identification”.

Nom de l'option pour le mode silencieux : `--aws-access-key-id`

AWSClé d'accès secrète

Clé d'accès secrète provenant des informations d'identification de sécurité à long terme ou temporaires. Spécifiez cette option comme valeur d'entrée uniquement si vous ne fournissez pas vos informations d'identification en tant que variables d'environnement. Pour plus d'informations, veuillez consulter the section called “Fournir desCompte AWS informations d'identification”.

Nom de l'option pour le mode silencieux : `--aws-secret-access-key`

AWSjeton de session

Jeton de session provenant des informations d'identification de sécurité temporaires. Spécifiez cette option comme valeur d'entrée uniquement si vous ne fournissez pas vos informations d'identification en tant que variables d'environnement. Pour plus d'informations, veuillez consulter the section called “Fournir desCompte AWS informations d'identification”.

Nom de l'option pour le mode silencieux : `--aws-session-token`

Région AWS

CasRégion AWS dans lequel vous souhaitez créer le groupe Greengrass. Pour obtenir la liste des versions prises AWS IoT Greengrassen chargeRégion AWS, reportez-vous au Référence générale d'Amazon Web Services.

Valeur par défaut : `us-west-2`

Nom de l'option pour le mode silencieux : `--region`

Nom du groupe

Nom du groupe Greengrass.

Valeur par défaut : `GreengrassDeviceSetup_Group_`*`guid`*

Nom de l'option pour le mode silencieux : `--group-name`

Nom du noyau

Nom du noyau Greengrass. Le noyau est un appareil AWS IoT (objet) qui exécute le logiciel AWS IoT Greengrass Core. Le noyau est ajouté au registre AWS IoT et au groupe Greengrass. Si vous fournissez un nom, il doit être unique entreCompte AWS etRégion AWS.

Valeur par défaut : `GreengrassDeviceSetup_Core_`*`guid`*

Nom de l'option pour le mode silencieux : `--core-name`

Chemin d'installation du logiciel AWS IoT Greengrass Core

Emplacement dans le système de fichiers de l'appareil où vous souhaitez installer le logiciel AWS IoT Greengrass Core.

Valeur par défaut : `/`

Nom de l'option pour le mode silencieux : `--ggc-root-path`

Fonction Lambda Hello World

Indique s'il faut inclure une fonction Lambda Hello World dans le groupe Greengrass. La fonction publie un message MQTT dans la rubrique `hello/world` toutes les cinq secondes.

Le script crée et publie cette fonction Lambda définie par l'utilisateur dansAWS Lambda et l'ajoute à votre groupe Greengrass. Le script crée également un abonnement dans le groupe qui autorise la fonction d'envoyer des messages MQTT à AWS IoT.

> **Note**
>
> Il s'agit d'une fonction Lambda de Python 3.7. Si Python 3.7 n'est pas installé sur l'appareil et que le script ne peut pas l'installer, le script affiche un message d'erreur dans le terminal. Pour inclure la fonction Lambda dans le groupe, vous devez installer Python 3.7 manuellement et redémarrer le script. Pour créer le groupe Greengrass sans la fonction Lambda, redémarrez le script et entrezno lorsque vous êtes invité à inclure la fonction.

Valeur par défaut : no

Nom de l'option pour le mode silencieux : `--hello-world-lambda` - Cette option ne prend pas de valeur. Incluez-la dans votre commande si vous souhaitez créer la fonction.

Temps de déploiement

Nombre de secondes avant que la configuration de l'appareil Greengrass cesse de vérifier le statut du déploiement du groupe Greengrass. Ceci est utilisé uniquement lorsque le groupe inclut la fonction Lambda Hello World. Sinon, le groupe n'est pas déployé.

Le temps de déploiement dépend de la vitesse de votre réseau. Pour des vitesses réseau lentes, vous pouvez augmenter cette valeur.

Valeur par défaut : `180`

Nom de l'option pour le mode silencieux : `--deployment-timeout`

Chemin d'accès au journal

Emplacement du fichier journal contenant des informations sur les opérations de configuration du groupe et du noyau Greengrass. Utilisez ce journal pour résoudre les problèmes de déploiement et les autres problèmes liés à la configuration du groupe et du noyau Greengrass.

Valeur par défaut : `./`

Nom de l'option pour le mode silencieux : `--log-path`

Niveau de détail

Indique si les informations de journal détaillées doivent être imprimées dans le terminal pendant l'exécution du script. Vous pouvez utiliser ces informations pour résoudre les problèmes de configuration de l'appareil.

Valeur par défaut : `no`

Nom de l'option pour le mode silencieux : `--verbose` - Cette option ne prend pas de valeur. Incluez-la dans votre commande si vous souhaitez imprimer des informations de journal détaillées.

Exécution de la configuration de l'appareil Greengrass en mode silencieux

Vous pouvez exécuter la configuration de l'appareil Greengrass en mode silencieux afin que le script ne vous demande aucune valeur. Pour utiliser le mode silencieux, spécifiez le mode `bootstrap-greengrass` et vos valeurs d'entrée après le démarrage du script. Vous pouvez omettre les valeurs d'entrée si vous souhaitez utiliser leurs valeurs par défaut.

La procédure varie selon que vous fournissez vosCompte AWS informations d'identification sous forme de variables d'environnement avant de démarrer le script ou sous forme de valeurs d'entrée après le démarrage du script.

Fourniture des informations d'identification sous forme de variables d'environnement

1. Fournissez vos informations d'identification en tant que variables d'environnement. L'exemple suivant exporte les informations d'identification temporaires, qui incluent le jeton de session.

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

> **Note**
>
> Si vous exécutez la configuration de l'appareil Greengrass sur un Raspbian ou une OpenWrt plate-forme, créez une copie de ces commandes. Vous devez les fournir à nouveau après le redémarrage de l'appareil.

2. Téléchargez et démarrez le script. Fournissez les valeurs d'entrée si nécessaire. Par exemple :

   - Pour utiliser toutes les valeurs par défaut :

     ```
     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
     ```

   - Pour spécifier des valeurs personnalisées :

     ```
     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**
>
> Pour utiliser `curl` afin de télécharger le script, remplacez `wget -q -O` par `curl` dans la commande.

3. Si votre appareil principal exécute Raspbian ou OpenWrt, redémarrez l'appareil lorsque vous y êtes invité, fournissez vos informations d'identification, puis redémarrez le script.

   a. Lorsque vous êtes invité à redémarrer l'appareil, exécutez l'une des commandes suivantes.

      Pour les plateformes Raspbian :

      ```
      sudo reboot
      ```

      Pour les OpenWrt plateformes :

      ```
      reboot
      ```

   b. Une fois l'appareil redémarré, ouvrez le terminal et fournissez vos informations d'identification en tant que variables d'environnement.

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

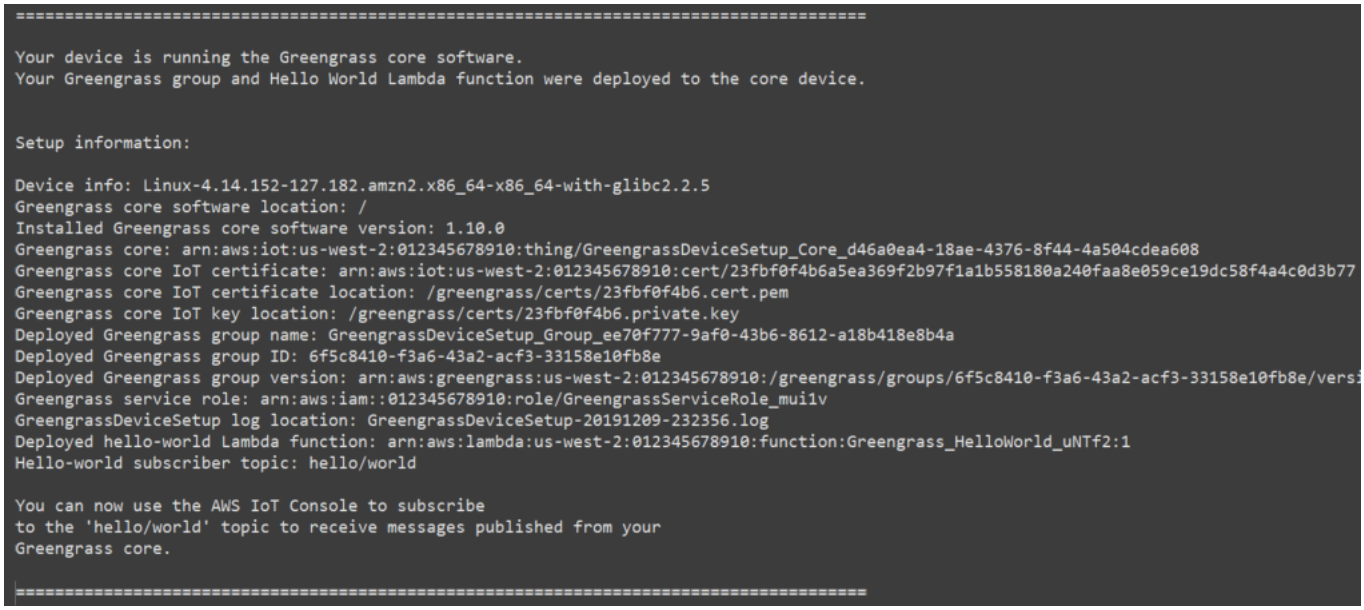
   c. Redémarrez le script.

      ```
      sudo -E ./gg-device-setup-latest.sh bootstrap-greengrass
      ```

   d. Lorsque vous êtes invité à utiliser vos valeurs d'entrée de la session précédente ou à démarrer une nouvelle installation, entrez `yes` pour réutiliser vos valeurs d'entrée.

      > **Note**
      >
      > Sur les plateformes nécessitant un redémarrage, vos valeurs d'entrée de la session précédente, sauf les informations d'identification, sont stockées temporairement dans le fichier `GreengrassDeviceSetup.config.info`.

   Lorsque la configuration est terminée, le terminal affiche un message de statut de réussite semblable au message suivant.



4. Si vous avez inclus la fonction Hello World Lambda, la configuration de l'appareil Greengrass déploie le groupe Greengrass sur votre appareil principal. Pour tester la fonction Lambda ou pour obtenir des informations sur la manière de supprimer la fonction Lambda du groupe,the section called “Vérification de l'exécution de la fonction Lambda sur l'appareil principal (noyau)” passez au module 3-1 du didacticiel de démarrage.

> **Note**
>
> Assurez-vous que la valeurRégion AWS sélectionnée dans la console est la même que celle que vous avez utilisée pour configurer votre environnement Greengrass. Par défaut, il s'agit de la Ouest Ouest.

Si vous n'avez pas inclus la fonction Lambda Hello World, vous pouvez créer votre propre fonction Lambda ou essayer d'autres fonctionnalités de Greengrass. Par exemple, vous pouvez ajouter le connecteur de déploiement d'application Docker à votre groupe et l'utiliser pour déployer des conteneurs Docker sur votre appareil principal.

Fourniture des informations d'identification sous forme de valeurs d'entrée

1. Téléchargez et démarrez le script. Indiquez vos informations d'identification et toute autre valeur d'entrée que vous souhaitez spécifier. Les exemples suivants montrent comment fournir des informations d'identification temporaires, qui incluent le jeton de session.

   - Pour utiliser toutes les valeurs par défaut :

   ```
   wget -q -O ./gg-device-setup-latest.sh https://d1onfpft10uf5o.cloudfront.net/
   greengrass-device-setup/downloads/gg-device-setup-latest.sh && chmod +x ./
   gg-device-setup-latest.sh && sudo -E ./gg-device-setup-latest.sh bootstrap-
   greengrass
   --aws-access-key-id AKIAIOSFODNN7EXAMPLE
   --aws-secret-access-key wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
   --aws-session-token AQoDYXdzEJr1K...o5OytwEXAMPLE=
   ```

   - Pour spécifier des valeurs personnalisées :

   ```
   wget -q -O ./gg-device-setup-latest.sh https://d1onfpft10uf5o.cloudfront.net/
   greengrass-device-setup/downloads/gg-device-setup-latest.sh && chmod +x ./
   gg-device-setup-latest.sh && sudo -E ./gg-device-setup-latest.sh bootstrap-
   greengrass
   --aws-access-key-id AKIAIOSFODNN7EXAMPLE
   --aws-secret-access-key wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
   --aws-session-token AQoDYXdzEJr1K...o5OytwEXAMPLE=
   --region us-east-1
   --group-name Custom_Group_Name
   --core-name Custom_Core_Name
   --ggc-root-path /custom/ggc/root/path
   --deployment-timeout 300
   --log-path /customized/log/path
   --hello-world-lambda
   --verbose
   ```

   > **Note**
   >
   > Si vous utilisez la configuration d'un appareil Greengrass sur un Raspbian ou une OpenWrt plateforme, faites une copie de vos informations d'identification. Vous devez les fournir à nouveau après le redémarrage de l'appareil.

Pour utiliser `curl` afin de télécharger le script, remplacez `wget -q -O` par `curl` dans la commande.

2. Si votre appareil principal exécute Raspbian ou OpenWrt, redémarrez l'appareil lorsque vous y êtes invité, fournissez vos informations d'identification, puis redémarrez le script.

   a. Lorsque vous êtes invité à redémarrer l'appareil, exécutez l'une des commandes suivantes.

      Pour les plateformes Raspbian :

      ```
      sudo reboot
      ```

      Pour les OpenWrt plateformes :

      ```
      reboot
      ```

   b. Redémarrez le script. Vous devez inclure vos informations d'identification dans la commande, mais pas les autres valeurs d'entrée. Par exemple :

      ```
      sudo -E ./gg-device-setup-latest.sh bootstrap-greengrass
      --aws-access-key-id AKIAIOSFODNN7EXAMPLE
      --aws-secret-access-key wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
      --aws-session-token AQoDYXdzEJr1K...o5OytwEXAMPLE=
      ```

   c. Lorsque vous êtes invité à utiliser vos valeurs d'entrée de la session précédente ou à démarrer une nouvelle installation, entrez `yes` pour réutiliser vos valeurs d'entrée.

      > **Note**
      >
      > Sur les plateformes nécessitant un redémarrage, vos valeurs d'entrée de la session précédente, sauf les informations d'identification, sont stockées temporairement dans le fichier `GreengrassDeviceSetup.config.info`.

Lorsque la configuration est terminée, le terminal affiche un message de statut de réussite semblable au message suivant.

```
================================================================================

Your device is running the Greengrass core software.
Your Greengrass group and Hello World Lambda function were deployed to the core device.


Setup information:

Device info: Linux-4.14.152-127.182.amzn2.x86_64-x86_64-with-glibc2.2.5
Greengrass core software location: /
Installed Greengrass core software version: 1.10.0
Greengrass core: arn:aws:iot:us-west-2:012345678910:thing/GreengrassDeviceSetup_Core_d46a0ea4-18ae-4376-8f44-4a504cdea608
Greengrass core IoT certificate: arn:aws:iot:us-west-2:012345678910:cert/23fbf0f4b6a5ea369f2b97f1a1b558180a240faa8e059ce19dc58f4a4c0d3b77
Greengrass core IoT certificate location: /greengrass/certs/23fbf0f4b6.cert.pem
Greengrass core IoT key location: /greengrass/certs/23fbf0f4b6.private.key
Deployed Greengrass group name: GreengrassDeviceSetup_Group_ee70f777-9af0-43b6-8612-a18b418e8b4a
Deployed Greengrass group ID: 6f5c8410-f3a6-43a2-acf3-33158e10fb8e
Deployed Greengrass group version: arn:aws:greengrass:us-west-2:012345678910:/greengrass/groups/6f5c8410-f3a6-43a2-acf3-33158e10fb8e/versi
Greengrass service role: arn:aws:iam::012345678910:role/GreengrassServiceRole_mui1v
GreengrassDeviceSetup log location: GreengrassDeviceSetup-20191209-232356.log
Deployed hello-world Lambda function: arn:aws:lambda:us-west-2:012345678910:function:Greengrass_HelloWorld_uNTf2:1
Hello-world subscriber topic: hello/world

You can now use the AWS IoT Console to subscribe
to the 'hello/world' topic to receive messages published from your
Greengrass core.

================================================================================
```

3. Si vous avez inclus la fonction Hello World Lambda, la configuration de l'appareil Greengrass déploie le groupe Greengrass sur votre appareil principal. Pour tester la fonction Lambda ou pour obtenir des informations sur la manière de supprimer la fonction Lambda du groupe,the section called “Vérification de l'exécution de la fonction Lambda sur l'appareil principal (noyau)” passez au module 3-1 du didacticiel de démarrage.

   > **Note**
   >
   > Assurez-vous que la valeurRégion AWS sélectionnée dans la console est la même que celle que vous avez utilisée pour configurer votre environnement Greengrass. Par défaut, il s'agit de la Ouest Ouest.

   Si vous n'avez pas inclus la fonction Lambda Hello World, vous pouvez créer votre propre fonction Lambda ou essayer d'autres fonctionnalités de Greengrass. Par exemple, vous pouvez ajouter le connecteur de déploiement d'application Docker à votre groupe et l'utiliser pour déployer des conteneurs Docker sur votre appareil principal.

Module 1 : Configuration de l'environnement pour Greengrass

Ce module vous montre comment obtenir une carte out-of-the-box Raspberry Pi, une instance Amazon EC2 ou tout autre appareil prêt à être utilisé parAWS IoT Greengrasscomme vos produitsAWS IoT Greengrassappareil noyau.

> **Tip**
>
> Ou, pour utiliser un script qui configure votre appareil principal à votre place, veuillez consulter the section called “Démarrage rapide : Configuration de l'appareil Greengrass”.

Le module dure approximativement 30 minutes.

Avant de commencer, lisez les conditions requises pour ce didacticiel. Ensuite, suivez les instructions de configuration dans l'une des rubriques suivantes. Choisissez uniquement la rubrique qui s'applique à votre type d'appareil principal.

Rubriques

> **Note**
>
> Pour savoir comment utiliser l'exécution de AWS IoT Greengrass dans un conteneur Docker prédéfini, consultez the section called “Exécuter AWS IoT Greengrass dans un conteneur Docker”.

Configuration d'une carte Raspberry Pi

Suivez les étapes de cette rubrique pour configurer un Raspberry Pi à utiliser en tant queAWS IoT GreengrassCœur.

> **Tip**
>
> AWS IoT Greengrass fournit également d'autres options pour installer le logiciel AWS IoT Greengrass Core. Par exemple, vous pouvez utiliser la configuration de l'appareil Greengrass pour configurer votre environnement et installer la dernière version du logiciel AWS IoT Greengrass Core. Ou, sur les plateformes Debian prises en charge, vous pouvez utiliser le gestionnaire de paquets APT pour installer ou mettre à niveau le logiciel AWS IoT Greengrass Core. Pour plus d'informations, consultez the section called “Installer le logiciel AWS IoT Greengrass Core”.

Si vous configurez une carte Raspberry Pi pour la première fois, vous devez suivre toutes ces étapes. Sinon, vous pouvez passer à l'étape 9. Toutefois, nous vous recommandons de recréer une image de votre carte Raspberry Pi avec le système d'exploitation, comme recommandé à l'étape 2.

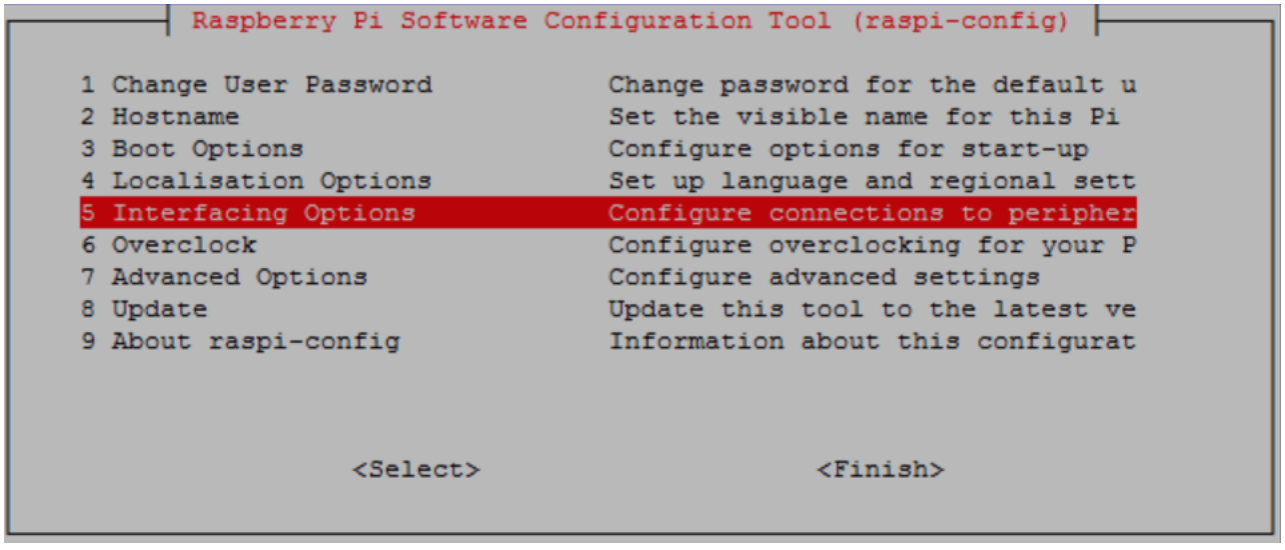
1. Téléchargez et installez un logiciel de formatage de carte SD tel queFormateur de carte mémoire SD. Insérez la carte SD dans votre ordinateur. Démarrez le programme et choisissez le lecteur où vous avez inséré votre carte SD. Vous pouvez effectuer un formatage rapide de la carte SD.
2. Téléchargez le système d'exploitation Raspbian Buster sous la forme d'un fichier `zip`.
3. Lors de l'utilisation d'un outil d'écriture sur carte SD (par exemple, Etcher), suivez les instructions de l'outil pour flasher le fichier `zip` téléchargé sur la carte SD. Comme l'image du système d'exploitation est volumineuse, cette étape peut prendre un peu de temps. Ejectez votre carte SD de votre ordinateur et insérez la carte microSD dans votre Raspberry Pi.
4. Pour le premier démarrage, nous vous recommandons de connecter la carte Raspberry Pi à un moniteur (via HDMI), un clavier et une souris. Connectez ensuite votre carte Pi à une source d'alimentation micro USB et le système d'exploitation Raspbian doit démarrer.
5. Vous pouvez également configurer la disposition du clavier de l'appareil Pi avant de continuer. Pour ce faire, cliquez sur l'icône Raspberry dans l'angle supérieur droit, choisissez Preferences, puis Mouse and Keyboard Settings. Choisissez ensuite l'onglet Keyboard et Keyboard Layout, puis sélectionnez une variante de clavier appropriée.
6. Puis connectez votre Raspberry Pi à Internet via un réseau Wi-Fi ou un câble Ethernet.

> **Note**
>
> Connectez votre Raspberry Pi au même réseau que votre ordinateur et assurez-vous que votre ordinateur et votre Raspberry Pi ont tous deux accès à Internet avant de continuer. Si vous vous trouvez dans un environnement de travail ou derrière un pare-feu, vous devrez peut-être connecter votre Pi et votre ordinateur au réseau invité afin que les deux appareils soient sur le même réseau. Cette approche risque néanmoins de déconnecter votre ordinateur de certaines ressources du réseau local, comme l'intranet. Une solution possible consiste à connecter le Pi ainsi que votre ordinateur au réseau Wi-Fi invité et de relier votre ordinateur au réseau local via un câble Ethernet. Avec cette configuration, votre ordinateur devrait pouvoir se connecter à la carte Raspberry Pi via le réseau Wi-Fi invité et aux ressources de votre réseau local grâce au câble Ethernet.

7. Vous devez configurer SSH sur votre carte Pi pour pouvoir vous y connecter à distance. Sur votre Raspberry Pi, ouvrez une fenêtre de terminal et exécutez la commande suivante :

```
sudo raspi-config
```

Vous devez voir ce qui suit :



Faites défiler la page vers le bas et choisissez Interfacing Options, puis P2 SSH. À l'invite, choisissez Oui. (Utilisez la clé Tab suivie de Enter). SSH doit maintenant être activé. Sélectionnez OK. Utilisez la clé Tab pour choisir Terminer, puis appuyez sur Enter. Si le Raspberry Pi ne redémarre pas automatiquement, exécutez la commande suivante :

```
sudo reboot
```

8. Sur votre Raspberry Pi, dans la fenêtre de terminal, exécutez la commande suivante :

```
hostname -I
```

Cela renvoie l'adresse IP de votre Raspberry Pi.

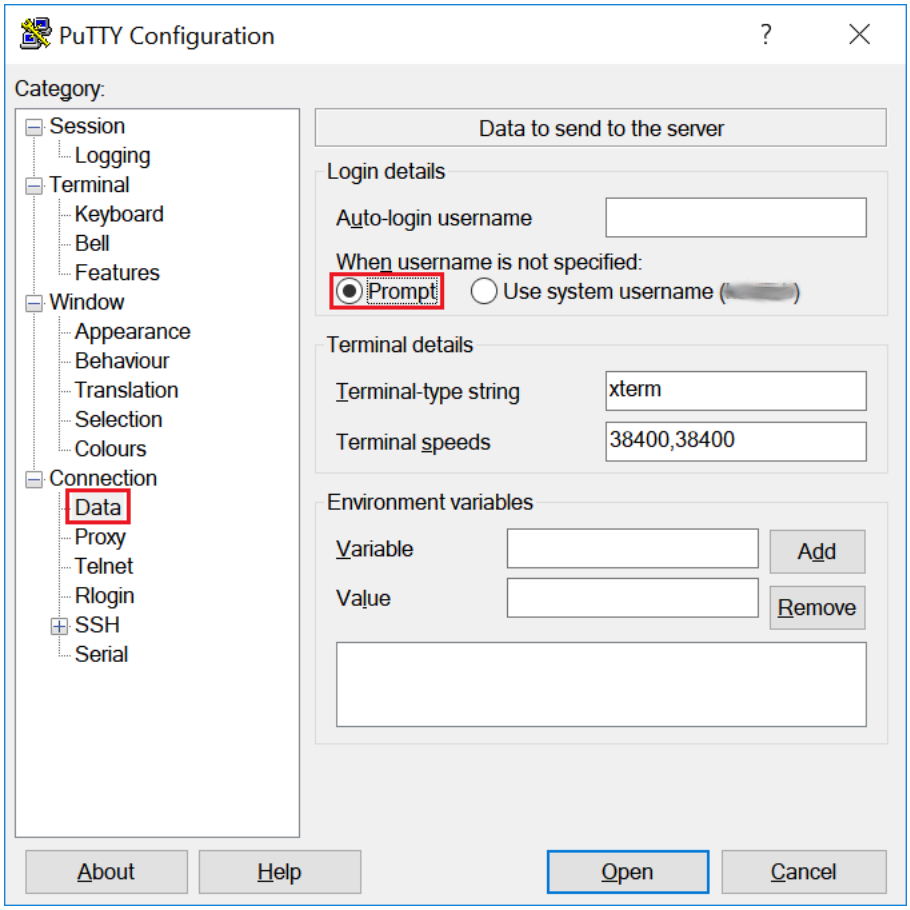
> **Note**
>
> Pour les éléments suivants, si vous recevez un message concernant l'empreinte digitale de la clé ECDSA (`Are you sure you want to continue connecting (yes/no)?`), indiquez `yes`. Le mot de passe par défaut pour le Raspberry Pi est **`raspberry`**.

Si vous utilisez macOS, ouvrez une fenêtre Terminal et tapez ce qui suit :

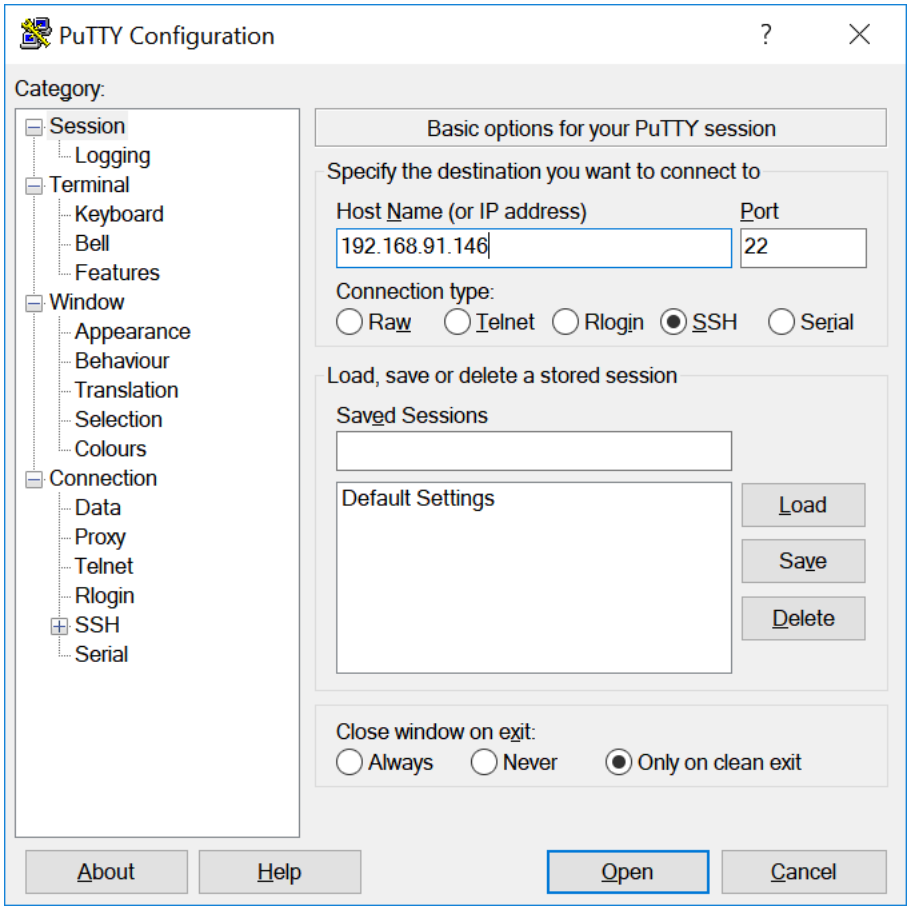
```
ssh pi@IP-address
```

`IP-address` correspond ici à l'adresse IP de votre Raspberry Pi, obtenue en utilisant la commande `hostname -I`.

Si vous utilisez Windows, vous devez installer et configurer PuTTY. Développez Connection, puis choisissez Data et assurez-vous que Prompt est sélectionné :

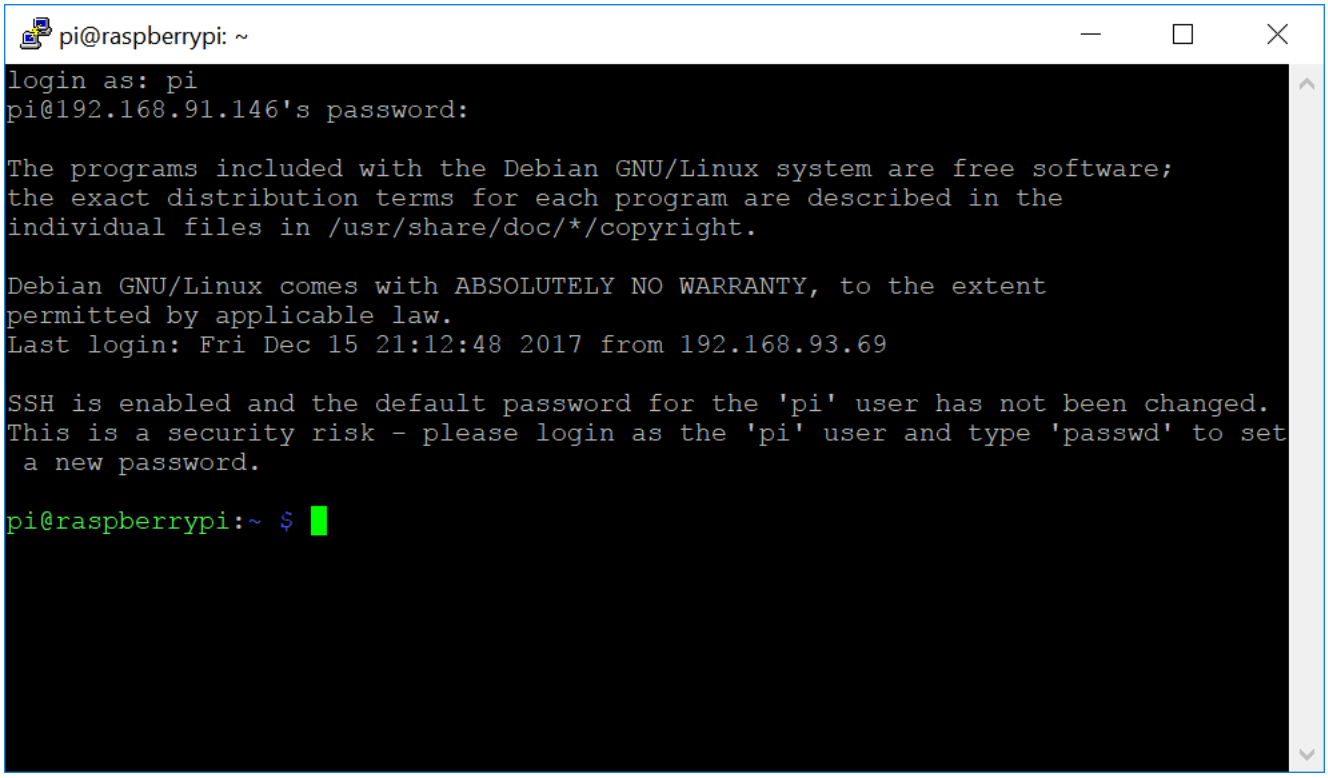


Choisissez ensuite Session, saisissez l'adresse IP du Raspberry Pi, puis sélectionnez Open en utilisant les paramètres par défaut.



Si une alerte de sécurité PuTTY s'affiche, choisissez Yes.

L'ID de connexion et le mot de passe par défaut de la carte Raspberry Pi sont **`pi`** et **`raspberry`**, respectivement.



> **Note**
>
> Si votre ordinateur est connecté à un réseau à distance à l'aide de VPN (par exemple, un réseau professionnel), vous pouvez rencontrer des difficultés pour vous connecter au Raspberry Pi à partir de l'ordinateur à l'aide de SSH.

9. Vous êtes maintenant prêt à configurer la carte Raspberry Pi pour AWS IoT Greengrass. Tout d'abord, exécutez les commandes suivantes à partir d'une fenêtre de terminal Raspberry Pi locale ou d'une fenêtre de terminal SSH :

> **Tip**
>
> AWS IoT Greengrass fournit également d'autres options pour installer le logiciel AWS IoT Greengrass Core. Par exemple, vous pouvez utiliser la configuration de l'appareil Greengrass pour configurer votre environnement et installer la dernière version du logiciel AWS IoT Greengrass Core. Ou, sur les plateformes Debian prises en charge, vous pouvez utiliser le gestionnaire de paquets APT pour installer ou mettre à niveau le logiciel AWS IoT Greengrass Core. Pour plus d'informations, consultez the section called “Installer le logiciel AWS IoT Greengrass Core”.

```
sudo adduser --system ggc_user
sudo addgroup --system ggc_group
```

10. Pour améliorer la sécurité sur l'appareil Pi, activez les protections hardlink et softlink (symlink) au démarrage du système d'exploitation.

    a. Accédez au fichier `98-rpi.conf`.

    ```
    cd /etc/sysctl.d
    ls
    ```

    > **Note**
    >
    > Si vous ne voyez pas le fichier `98-rpi.conf`, suivez les instructions fournies dans le fichier `README.sysctl`.

    b. Utilisez un éditeur de texte (tel que Leafpad, GNU nano ou vi) pour ajouter les deux lignes suivantes à la fin du fichier. Vous devrez peut-être utiliser la commande `sudo` pour modifier en tant qu'utilisateur racine (par exemple, `sudo nano 98-rpi.conf`).

    ```
    fs.protected_hardlinks = 1
    fs.protected_symlinks = 1
    ```

    c. Redémarrez le Pi.

    ```
    sudo reboot
    ```

    Après environ une minute, connectez-vous au Pi à l'aide de SSH, puis exécutez la commande suivante pour confirmer la modification :

    ```
    sudo sysctl -a 2> /dev/null | grep fs.protected
    ```

    Vous devez voir `fs.protected_hardlinks = 1` et `fs.protected_symlinks = 1`.

11. Modifiez votre fichier de démarrage de ligne de commande de manière à activer et monter les cgroups de mémoire. Cela permet à AWS IoT Greengrass de définir la limite de mémoire

pour les fonctions Lambda. Les Cgroups sont également requis pour exécuterAWS IoT Greengrassdans la valeur par défautConteneurisationmode.

a. Accédez à votre répertoire `boot`.

```
cd /boot/
```

b. Utilisez un éditeur de texte pour ouvrir `cmdline.txt`. Ajoutez les éléments suivants à la fin de la ligne existante, et non en tant que nouvelle ligne. Vous devrez peut-être utiliser la commande `sudo` pour modifier en tant qu'utilisateur racine (par exemple, `sudo nano cmdline.txt`).

```
cgroup_enable=memory cgroup_memory=1
```

c. Maintenant, redémarrez le Pi.

```
sudo reboot
```

Votre Raspberry Pi doit maintenant être prêt pour AWS IoT Greengrass.

12. Facultatif. Installez l'environnement d'exécution Java 8, qui est requis par le gestionnaire de flux. Ce didacticiel n'utilise pas le gestionnaire de flux, mais il utilise le flux de travail Default Group creation (Création du groupe par défaut) qui active le gestionnaire de flux par défaut. Utilisez les commandes suivantes pour installer l'environnement d'exécution Java 8 sur l'appareil principal, ou désactivez le gestionnaire de flux avant de déployer votre groupe. Les instructions permettant de désactiver le gestionnaire de flux sont fournies dans le module 3.

```
sudo apt install openjdk-8-jdk
```

13. Pour vous assurer que vous disposez de toutes les dépendances requises, téléchargez et exécutez le vérificateur de dépendances Greengrass depuis leAWS IoT GreengrassExemplesrepository sur GitHub. Ces commandes décompressent et exécutent le vérificateur de dépendance dans le`Downloads`directory.

> **Note**
>
> Le vérificateur de dépendances peut échouer si vous exécutez la version 5.4.51 du noyau Raspbian. Cette version ne monte pas correctement les cgroups de mémoire. Cela peut entraîner l'échec des fonctions Lambda exécutées en mode conteneur.

Pour plus d'informations sur la mise à jour de votre noyau, consultez leCgroups non chargés après la mise à niveau du noyaudans les forums 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
```

Là où `more` s'affiche, appuyez sur la touche Spacebar pour afficher un autre écran de texte.

Important

Ce didacticiel a besoin de l'environnement d'exécution Python 3.7 pour exécuter des fonctions Lambda locales. Lorsque le gestionnaire de flux est activé, il a également besoin de l'environnement d'exécution Java 8. Si le script `check_ggc_dependencies` génère des avertissements concernant des environnements d'exécution prérequis manquants, veillez à installer ces environnements d'exécution avant de continuer. Vous pouvez ignorer les avertissements concernant d'autres environnements d'exécution facultatifs manquants.

Pour en savoir plus sur la commande modprobe, exécutez man modprobe dans le terminal.

Votre configuration Raspberry Pi est terminée. Passez au the section called “Module 2 : Installation deAWS IoT GreengrassLogiciel Core”.

Configuration d'une instance Amazon EC2

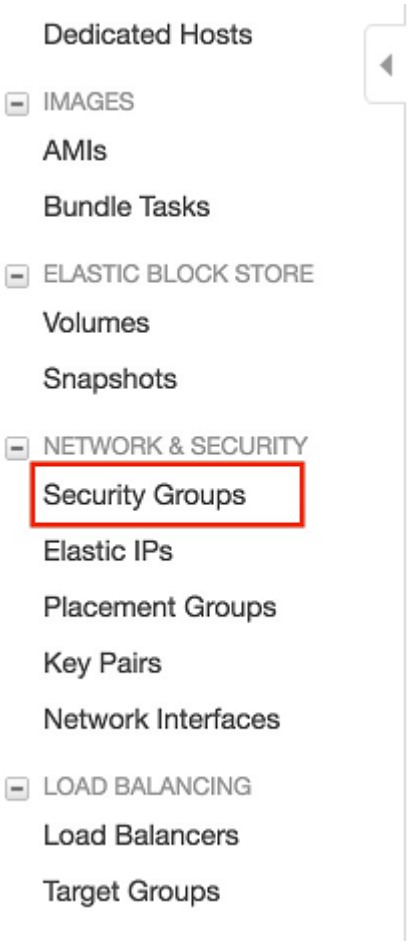
Suivez les étapes décrites dans cette rubrique pour configurer une instance Amazon EC2 à utiliser comme instance principale. AWS IoT Greengrass

> **Tip**
>
> Ou, pour utiliser un script qui configure votre environnement et installe le logiciel AWS IoT Greengrass Core pour vous, voirthe section called “Démarrage rapide : Configuration de l'appareil Greengrass”.

Bien que vous puissiez suivre ce didacticiel à l'aide d'une instance Amazon EC2, AWS IoT Greengrass il doit idéalement être utilisé avec du matériel physique. Nous vous recommandons de configurer un Raspberry Pi au lieu d'utiliser une instance Amazon EC2 lorsque cela est possible. Si vous utilisez un Raspberry Pi, vous n'avez pas besoin de suivre les étapes de cette rubrique.

1. Connectez-vous à l'instance Amazon EC2 AWS Management Consoleet lancez-la à l'aide d'une AMI Amazon Linux. Pour plus d'informations sur les instances Amazon EC2, consultez le guide de démarrage Amazon EC2.
2. Une fois que votre instance Amazon EC2 est en cours d'exécution, activez le port 8883 pour autoriser les communications MQTT entrantes afin que d'autres appareils puissent se connecter au cœur. AWS IoT Greengrass

   a. Dans le volet de navigation de la console Amazon EC2, sélectionnez Security Groups.



b. Sélectionnez le groupe de sécurité pour l'instance que vous venez de lancer, puis cliquez sur l'onglet Règles d'entrée.

c. Choisissez Modifier les règles entrantes.

   Pour activer le port 8883, vous devez ajouter une règle TCP personnalisée au groupe de sécurité. Pour plus d'informations, consultez la section Ajout de règles à un groupe de sécurité dans le guide de l'utilisateur Amazon EC2.

d. Sur la page Modifier les règles entrantes, choisissez Ajouter une règle, entrez les paramètres suivants, puis sélectionnez Enregistrer.

   - Pour Type, choisissez Règle TCP personnalisée.
   - Dans Portée de ports, entrez**8883**.
   - Pour Source, choisissez N'importe où.
   - Pour Description, saisissez **`MQTT Communications`**.

3. Connectez-vous à votre instance EC2 Amazon.

   a. Dans le panneau de navigation, choisissez Instances, puis sélectionnez votre instance et choisissez Connexion.

   b. Suivez les instructions de la page Connectez-vous à votre instance pour vous connecter à votre instance l'aide de SSH et de votre fichier de clé privée.

   Vous pouvez utiliser PuTTY pour Windows ou Terminal pour macOS. Pour plus d'informations, consultez Connect to your Linux instance dans le guide de l'utilisateur Amazon EC2.

   Vous êtes maintenant prêt à configurer votre instance Amazon EC2 pour. AWS IoT Greengrass

4. Une fois connecté à votre instance Amazon EC2, créez les comptes `ggc_user` et `ggc_group` :

```
sudo adduser --system ggc_user
sudo groupadd --system ggc_group
```

> **Note**
>
> Si la commande `adduser` n'est pas disponible sur votre système, utilisez la commande suivante.
>
> ```
> sudo useradd --system ggc_user
> ```

5. Pour améliorer la sécurité, assurez-vous que les protections des liens physiques et des liens souples (liens symboliques) sont activées sur le système d'exploitation de l'instance Amazon EC2 au démarrage.

> **Note**
>
> Les étapes permettant d'activer les protections hardlink et softlink varient selon le système d'exploitation. Consultez la documentation de votre distribution.

   a. Exécutez la commande suivante pour vérifier si les protections hardlink et softlink sont activées :

```
sudo sysctl -a | grep fs.protected
```

Si les protections hardlink et softlinks sont définies sur 1, vos protections sont correctement activées. Passez à l'étape 6.

> **Note**
>
> Les protections softlink sont représentées par `fs.protected_symlinks`.

b. Si les protections hardlink et softlink ne sont pas définies sur 1, activez ces protections. Accédez au fichier de configuration du système.

```
cd /etc/sysctl.d
ls
```

c. À l'aide de l'éditeur de texte de votre choix (tel que Leafpad, GNU nano ou vi), ajoutez les deux lignes suivantes à la fin du fichier de configuration du système. Sur Amazon Linux 1, il s'agit du fichier `00-defaults.conf`. Sur Amazon Linux 2, il s'agit du fichier `99-amazon.conf`. Vous devrez peut-être modifier les autorisations du fichier (à l'aide de la commande `chmod`) afin de pouvoir écrire sur ce fichier, ou utiliser la commande `sudo` pour le modifier en tant qu'utilisateur racine (par exemple, `sudo nano 00-defaults.conf`).

```
fs.protected_hardlinks = 1
fs.protected_symlinks = 1
```

d. Redémarrez l'instance Amazon EC2.

```
sudo reboot
```

Après quelques minutes, connectez-vous à votre instance à l'aide de SSH, puis exécutez la commande suivante pour confirmer la modification.

```
sudo sysctl -a | grep fs.protected
```

Vous devriez voir que les protections hardlink et softlink sont définies avec la valeur 1.

6. Extrayez et exécutez le script suivant pour monter les groupes de contrôle Linux (cgroups). Cela permet AWS IoT Greengrass de définir la limite de mémoire pour les fonctions Lambda. Les

groupes C doivent également s'exécuter AWS IoT Greengrass dans le mode de conteneurisation par défaut.

```
curl https://raw.githubusercontent.com/tianon/cgroupfs-
mount/951c38ee8d802330454bdede20d85ec1c0f8d312/cgroupfs-mount > cgroupfs-mount.sh
chmod +x cgroupfs-mount.sh
sudo bash ./cgroupfs-mount.sh
```

Votre instance Amazon EC2 devrait maintenant être prête pour. AWS IoT Greengrass

7. Facultatif. Installez l'environnement d'exécution Java 8, qui est requis par le gestionnaire de flux. Ce didacticiel n'utilise pas le gestionnaire de flux, mais il utilise le flux de travail Default Group creation (Création du groupe par défaut) qui active le gestionnaire de flux par défaut. Utilisez les commandes suivantes pour installer l'environnement d'exécution Java 8 sur l'appareil principal, ou désactivez le gestionnaire de flux avant de déployer votre groupe. Les instructions permettant de désactiver le gestionnaire de flux sont fournies dans le module 3.

   - Pour les distributions basées sur Debian :

     ```
     sudo apt install openjdk-8-jdk
     ```

   - Pour les distributions basées sur Red Hat :

     ```
     sudo yum install java-1.8.0-openjdk
     ```

8. Pour vous assurer que vous disposez de toutes les dépendances requises, téléchargez et exécutez le vérificateur de dépendances Greengrass depuis le référentiel AWS IoT Greengrass Samples. GitHub Ces commandes téléchargent, décompressent et exécutent le script de vérification des dépendances dans votre instance 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**
>
> Ce didacticiel nécessite le moteur d'exécution Python 3.7 pour exécuter les fonctions Lambda locales. Lorsque le gestionnaire de flux est activé, il a également besoin de l'environnement d'exécution Java 8. Si le script `check_ggc_dependencies` génère des avertissements concernant des environnements d'exécution prérequis manquants, veillez à installer ces environnements d'exécution avant de continuer. Vous pouvez ignorer les avertissements concernant d'autres environnements d'exécution facultatifs manquants.

La configuration de votre instance Amazon EC2 est terminée. Passez au the section called “Module 2 : Installation deAWS IoT GreengrassLogiciel Core”.

Configuration d'autres appareils

Suivez les étapes de cette rubrique pour configurer un appareil (autre qu'un Raspberry Pi) à utiliser en tant que.AWS IoT GreengrassCore.

> **Tip**
>
> Ou, pour utiliser un script qui configure votre environnement et installe le logiciel AWS IoT Greengrass Core pour vous, consultez the section called “Démarrage rapide : Configuration de l'appareil Greengrass”.

Si vous débutez avecAWS IoT Greengrass, nous vous recommandons d'utiliser une carte Raspberry Pi ou une instance Amazon EC2 en tant qu'appareil principal, puis de suivre les.étapes de configurationadapté à votre appareil.

Si vous envisagez de créer un système Linux personnalisé à l'aide du projet Yocto, vous pouvez utiliser la sectionAWS IoT GreengrassRecette Bitbake de la`meta-aws`projet. Cette recette vous aide également à développer une plateforme logicielle qui prend en chargeAWSlogiciel Edge pour applications embarquées. La version Bitbake installe, configure et exécute automatiquement leAWS IoT GreengrassLogiciels de base sur votre appareil.

Yocto

Un projet de collaboration open source qui vous aide à créer des systèmes Linux personnalisés pour des applications intégrées, quelle que soit l'architecture matérielle. Pour plus d'informations, consultez le .Yocto.

`meta-aws`

UnAWSprojet géré qui fournit des recettes Yocto. Vous pouvez utiliser les recettes pour développerAWSlogiciel Edge dans les systèmes Linux conçus avecOpenEmbeddedet Yocto Project. Pour plus d'informations sur cette fonctionnalité prise en charge par la communauté, consultez la section`meta-aws`sur GitHub.

`meta-aws-demos`

UnAWSprojet géré qui contient des démonstrations pour le`meta-aws`projet. Pour plus d'exemples sur le processus d'intégration, consultez la section`meta-aws-demos`sur GitHub.

Pour utiliser un autre appareil ou une autre plateforme prise en charge, suivez les étapes de cette rubrique.

1. Si votre appareil principal est un NVIDIA Jetson, vous devez d'abord flasher le microprogramme avec le JetPack 4.3 Installateur. Si vous configurez un appareil différent, passez à l'étape 2.

   > **Note**
   >
   > Le JetPack La version du programme d'installation que vous utilisez est basée sur la version cible de votre boîte à outils CUDA. Les instructions suivantes utilisent JetPack 4.3 et CUDA Toolkit 10.0. Pour de plus amples informations sur l'utilisation des versions appropriées pour votre appareil, veuillez consulter How to Install Jetpack dans la documentation NVIDIA.

   a. Sur un bureau physique exécutant Ubuntu version 16.04 ou ultérieure, flashez le microprogramme avec le bouton JetPack 4.3 programme d'installation, comme décrit dansTéléchargement et installation JetPack(4.3) dans la documentation NVIDIA.

      Suivez les instructions affichées dans le programme d'installation pour installer tous les packages et toutes les dépendances sur la carte Jetson connectée à l'ordinateur de bureau à l'aide d'un câble Micro-B.

b. Redémarrez votre carte en mode normal, puis connectez un écran à la carte.

> **Note**
>
> Lorsque vous utilisez SSH pour vous connecter à la carte Jetson, utilisez le nom d'utilisateur par défaut (**`nvidia`**) et le mot de passe par défaut (**`nvidia`**).

2. Exécutez les commandes suivantes pour créer l'utilisateur `ggc_user` et le groupe `ggc_group`. Les commandes que vous exécutez varient en fonction de la distribution installée sur votre appareil principal.

   - Si votre appareil principal exécute OpenWrt, exécutez les commandes suivantes :

   ```
   opkg install shadow-useradd
   opkg install shadow-groupadd
   useradd --system ggc_user
   groupadd --system ggc_group
   ```

   - Dans le cas contraire, exécutez les commandes suivantes :

   ```
   sudo adduser --system ggc_user
   sudo addgroup --system ggc_group
   ```

   > **Note**
   >
   > Si la commande `addgroup` n'est pas disponible sur votre système, utilisez la commande suivante.
   >
   > ```
   > sudo groupadd --system ggc_group
   > ```

3. Facultatif. Installez l'environnement d'exécution Java 8, qui est requis par le gestionnaire de flux. Ce didacticiel n'utilise pas le gestionnaire de flux, mais il utilise le flux de travail Default Group creation (Création du groupe par défaut) qui active le gestionnaire de flux par défaut. Utilisez les commandes suivantes pour installer l'environnement d'exécution Java 8 sur l'appareil principal, ou désactivez le gestionnaire de flux avant de déployer votre groupe. Les instructions permettant de désactiver le gestionnaire de flux sont fournies dans le module 3.

   - Pour les distributions basées sur Debian ou Ubuntu :

```
sudo apt install openjdk-8-jdk
```

- Pour les distributions basées sur Red Hat :

```
sudo yum install java-1.8.0-openjdk
```

4. Pour vous assurer que vous disposez de toutes les dépendances requises, téléchargez et exécutez le vérificateur de dépendances Greengrass à partir duAWS IoT GreengrassExemplessur GitHub. Ces commandes décompressent et exécutent le vérificateur de dépendance.

```
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**
>
> Le script `check_ggc_dependencies` s'exécute sur les plateformes prises en charge par AWS IoT Greengrass et nécessite des commandes système Linux spécifiques. Pour plus d'informations, consultez le fichier Readme du vérificateur de dépendance.

5. Installez toutes les dépendances requises sur votre appareil, comme indiqué par la sortie du vérificateur de dépendance. S'il manque des dépendances au niveau du noyau, il se peut que vous ayez besoin de recompiler votre noyau. Pour monter les groupes de contrôle Linux (`cgroups`), vous pouvez exécuter le script cgroupfs-mount. Cela permet à AWS IoT Greengrass de définir la limite de mémoire pour les fonctions Lambda. Des groupes C sont également requis pour exécuterAWS IoT Greengrassdans la valeur par défautconteneurisationmode.

   Si aucune erreur n'apparaît dans la sortie, AWS IoT Greengrass doit être en mesure de s'exécuter avec succès sur votre appareil.

> **Important**
>
> Ce didacticiel a besoin de l'environnement d'exécution Python 3.7 pour exécuter des fonctions Lambda locales. Lorsque le gestionnaire de flux est activé, il a également besoin de l'environnement d'exécution Java 8. Si le script `check_ggc_dependencies` génère des avertissements concernant des environnements d'exécution prérequis manquants, veillez à installer ces environnements d'exécution avant de continuer. Vous pouvez ignorer les avertissements concernant d'autres environnements d'exécution facultatifs manquants.

Pour obtenir la liste des exigences et des dépendances AWS IoT Greengrass, veuillez consulter the section called “Exigences et plateformes prises en charge”.

Module 2 : Installation deAWS IoT GreengrassLogiciel Core

Ce module explique comment installer le logiciel AWS IoT Greengrass Core sur l'appareil de votre choix. Dans ce module, vous commencez par créer un groupe et un noyau Greengrass. Ensuite, vous téléchargez, configurez et démarrez le logiciel sur votre appareil principal. Pour de plus amples informations sur les fonctions logicielles de AWS IoT Greengrass Core, veuillez consulter the section called “Configuration de AWS IoT Greengrass Core”.

Avant de commencer, assurez-vous d'avoir effectué les étapes de configuration du module 1 pour l'appareil que vous avez choisi.

> **Tip**
>
> AWS IoT Greengrass fournit également d'autres options pour installer le logiciel AWS IoT Greengrass Core. Par exemple, vous pouvez utiliser la configuration de l'appareil Greengrass pour configurer votre environnement et installer la dernière version du logiciel AWS IoT Greengrass Core. Ou, sur les plateformes Debian prises en charge, vous pouvez utiliser le gestionnaire de paquets APT pour installer ou mettre à niveau le logiciel AWS IoT Greengrass Core. Pour plus d'informations, consultez the section called “Installer le logiciel AWS IoT Greengrass Core”.

Le module dure approximativement 30 minutes.

Rubriques

- Mettre en service unAWS IoTquelque chose à utiliser comme noyau Greengrass
- Créez unAWS IoT Greengrass groupe pour le noyau
- Installation et exécutionAWS IoT Greengrasssur l'appareil principal

Mettre en service unAWS IoTquelque chose à utiliser comme noyau Greengrass

Greengrassnoyauxsont des appareils qui exécutentAWS IoT GreengrassLogiciel de base pour gérer les processus IoT locaux. Pour configurer un noyau Greengrass, vous devez créer unAWS IoT chose, qui représente un périphérique ou une entité logique qui se connecte àAWS IoT. Lorsque vous enregistrez un appareil en tant queAWS IoTthing, cet appareil peut utiliser un certificat numérique et des clés qui lui permettent d'accéder àAWS IoT. Vous utilisez unAWS IoTpolitiquepour permettre à l'appareil de communiquer avecAWS IoTetAWS IoT GreengrassServices .

Dans cette section, vous pouvez enregistrer votre appareil en tant queAWS IoTUn objet pour l'utiliser comme noyau Greengrass.

Pour créer unAWS IoTchose

1. Accédez à la console AWS IoT.
2. UnderGérer, DéveloppezTous les appareils, puisObjets.
3. Dans la pageObjetspage, choisissezCréation d'objets.
4. Dans la pageCréation d'objetspage, choisissezCréer un objet unique, puisSuivant.
5. Dans la pageSpécifier les propriétés de, procédez de la façon suivante :
   a. PourNom de l'objet, entrez un nom qui représente votre appareil, tel que**`MyGreengrassV1Core`**.
   b. Choisissez Next (Suivant).
6. Dans la pageConfiguration du certificat d'appareilpage, choisissezSuivant.
7. Dans la pageAttacher les stratégies au certificat, procédez de l'une des manières suivantes :
   - Sélectionnez une stratégie existante qui accorde les autorisations requises par les cœurs, puis choisissezCréation d'un objet.

Une fenêtre modale s'ouvre dans laquelle vous pouvez télécharger les certificats et les clés que l'appareil utilise pour se connecter auAWS Cloud.

- Créez et attachez une nouvelle stratégie qui accorde des autorisations principales aux appareils. Procédez comme suit :

  a. Sélectionnez Create policy (Créer la stratégie).

     La page Créer une stratégie s'ouvre dans un nouvel onglet.

  b. Sur la page Créer une stratégie, procédez comme suit :

     i. PourNom de la stratégie, saisissez un nom qui décrit la stratégie, par exemple**`GreengrassV1CorePolicy`**.

     ii. Dans la pageDéclarations de stratégieonglet, sousDocument de stratégie, choisissezJASON.

     iii. Saisissez le document de stratégie suivant. Cette politique permet au noyau de communiquer avecAWS IoT Core, interagissez avec les ombres de l'appareil et communiquez avecAWS IoT Greengrassservice. Pour plus d'informations sur la façon de limiter l'accès à cette stratégie en fonction de votre cas d'utilisation, consultezStratégie AWS IoT minimale pour l'appareil principal AWS IoT Greengrass (noyau).

```
{
  "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. Choisissez Create (Créer) pour créer la politique.

c. Retournez à l'onglet de navigateur avec l'Attacher les stratégies au certificatpage ouverte. Procédez comme suit :

i. DansStratégies, sélectionnez la stratégie que vous avez créée, par exempleGreengrassV1CorePolicy.

Si vous ne voyez pas cette stratégie, sélectionnez le bouton d'actualisation.

ii. ChoisissezCréation d'un objet.

Une fenêtre modale s'ouvre dans laquelle vous pouvez télécharger les certificats et les clés que le noyau utilise pour se connecterAWS IoT.

8. Retournez à l'onglet de navigateur avec l'Attacher les stratégies au certificatpage ouverte. Procédez comme suit :

a. DansStratégies, sélectionnez la stratégie que vous avez créée, par exempleGreengrassV1CorePolicy.

Si vous ne voyez pas cette stratégie, sélectionnez le bouton d'actualisation.

b. ChoisissezCréation d'un objet.

Une fenêtre modale s'ouvre dans laquelle vous pouvez télécharger les certificats et les clés que le noyau utilise pour se connecterAWS IoT.

9. DansTélécharger des certificats et des clésmodal, téléchargez les certificats de l'appareil.

> **Important**
>
> Avant de choisirTerminé, téléchargez les ressources de sécurité.

Procédez comme suit :

a. PourCertificat de l'appareil, choisissezTéléchargementpour télécharger le certificat de l'appareil.

b. PourFichier de clé publique, choisissezTéléchargementpour télécharger la clé publique du certificat.

c. PourFichier de clé privée, choisissezTéléchargementpour télécharger le fichier de clé privée du certificat.

d. VérificationAuthentification du serveurdans leAWS IoTManuel du développeuret choisissez le certificat d'autorité de certification racine approprié. Nous vous recommandons d'utiliser les points de terminaison Amazon Trust Services (ATS) et les certificats d'autorité de certification racine ATS. UnderCertificats d'autorité de certification, choisissezTéléchargementpour un certificat d'autorité de certification racine

e. Sélectionnez Done (Exécuté).

Notez l'ID de certificat commun dans les noms de fichiers pour les clés et le certificat d'appareil. Vous en aurez besoin ultérieurement.

Créez unAWS IoT Greengrass groupe pour le noyau

AWS IoT Greengrassles groupes contiennent des paramètres et d'autres informations sur ses composants, tels que les appareils clients, les fonctions Lambda et les connecteurs. Un groupe définit la configuration d'un noyau, y compris la manière dont ses composants peuvent interagir les uns avec les autres.

Dans cette section, vous allez créer un groupe pour votre noyau.

> **Tip**
>
> Pour un exemple d'utilisation de l'AWS IoT GreengrassAPI pour créer et déployer un groupe, consultez le référentiel gg_group_setup sur GitHub.

Pour créer un groupe pour le noyau

1. Accédez à la console AWS IoT.
2. Sous Gérer, développez les appareils Greengrass et choisissez Groupes (V1).

   > **Note**
   >
   > Si vous ne voyez pas le menu des appareils Greengrass, passez à unRégion AWS menu compatibleAWS IoT Greengrass V1. Pour la liste des régions prises en charge, consultez les AWS IoT Greengrass V1points de terminaison et les quotas dans le Références générales AWS. Vous devez créer l'AWS IoTobjet pour votre cœur dans une région où celaAWS IoT Greengrass V1 est disponible.

3. Sur la page Greengrass groups, choisissez Créer un groupe.
4. Sur la page Créer un groupe Greengrass procédez comme suit :

   a. Pour Nom de groupe Greengrass, saisissez un nom qui décrit le groupe, tel que**`MyGreengrassGroup`**.

   b. Pour Greengrass Core, choisissez l'AWS IoTélément que vous avez créé précédemment, tel que MyGreengrassV1Core.

      La console sélectionne automatiquement le certificat de l'appareil pour vous.

   c. Choisissez Créer un groupe.

Installation et exécutionAWS IoT Greengrasssur l'appareil principal

> **Note**
>
> Ce didacticiel fournit des instructions vous permettant d'exécuterAWS IoT GreengrassUn Raspberry Pi, mais vous pouvez utiliser n'importe quel appareil pris en charge.

Dans cette section, vous allez configurer, installer et exécuter leAWS IoT GreengrassLogiciel Core sur l'appareil noyau.

Pour installer et exécuterAWS IoT Greengrass

1. De laAWS IoT GreengrassLogiciel Corede ce guide, téléchargezAWS IoT GreengrassPackage d'installation du logiciel Core. Choisissez le package qui correspond le mieux à l'architecture du processeur, à la distribution et au système d'exploitation de votre appareil principal.

   - Pour Raspberry Pi, téléchargez le package pour l'architecture Armv7l et le système d'exploitation Linux.
   - Pour une instance Amazon EC2, téléchargez le package pour l'architecture x86_64 et le système d'exploitation Linux.
   - Pour NVIDIA Jetson TX2, téléchargez le package pour l'architecture Armv8 (AArch64) et le système d'exploitation Linux.
   - Pour Intel Atom, téléchargez le package pour l'architecture x86_64 et le système d'exploitation Linux.

2. Au cours des étapes précédentes, vous avez téléchargé cinq fichiers sur votre ordinateur :

   - `greengrass-`*`OS-architecture`*`-1.11.6.tar.gz`— Ce fichier compressé contient leAWS IoT GreengrassLogiciel qui s'exécute sur l'appareil noyau.
   - *`certificateId`*`-certificate.pem.crt`— Fichier de certificat de l'appareil.
   - *`certificateId`*`-public.pem.key`— Fichier de clé publique du certificat de l'appareil.
   - *`certificateId`*`-private.pem.key`— Fichier de clé privée du certificat de l'appareil.
   - `AmazonRootCA1.pem`— Fichier de l'autorité de certification (CA) racine (CA) racine Amazon.

   Au cours de cette étape, vous transférez ces fichiers depuis votre ordinateur vers l'appareil noyau. Procédez comme suit :

   a. Si vous ne connaissez pas l'adresse IP de votre appareil noyau Greengrass, ouvrez une fenêtre de terminal sur l'appareil noyau et exécutez la commande suivante.

      > **Note**
      >
      > Cette commande peut ne pas renvoyer l'adresse IP correcte pour certains appareils. Consultez la documentation de votre appareil pour récupérer votre adresse IP.

```
hostname -I
```

b. Transférez ces fichiers depuis votre ordinateur vers l'appareil noyau. Ces étapes varient en fonction du système d'exploitation de votre ordinateur. Choisissez votre système d'exploitation pour les étapes qui montrent comment transférer des fichiers vers votre appareil Raspberry Pi.

> **Note**
>
> Pour un Raspberry Pi, le nom d'utilisateur par défaut est **`pi`** et le mot de passe par défaut est **`raspberry`**.
> Pour un NVIDIA Jetson TX2, le nom d'utilisateur par défaut est **`nvidia`** et le mot de passe par défaut est **`nvidia`**.

Windows

Pour transférer les fichiers compressés depuis votre ordinateur vers un appareil noyau Raspberry Pi, utilisez un outil tel que WinSCP ou la commande PuTTY pscp. Pour utiliser la commande pscp, ouvrez une fenêtre d'invite de commande sur votre ordinateur et exécutez les éléments suivants :

```
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**
>
> Le numéro de version de cette commande doit correspondre à la version de votre package logiciel AWS IoT Greengrass Core.

macOS

Pour transférer les fichiers compressés depuis votre Mac vers un appareil noyau Raspberry Pi, ouvrez une fenêtre de terminal sur votre ordinateur et exécutez les commandes suivantes. Le*path-to-downloaded-files*est généralement`~/Downloads`.

> **Note**
>
> Il se peut que vous soyez invité à saisir deux mots de passe. Dans ce cas, le premier mot de passe est celui de la commande `sudo` du Mac, et le deuxième celui du 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**
>
> Le numéro de version de cette commande doit correspondre à la version de votre package logiciel AWS IoT Greengrass Core.

UNIX-like system

Pour transférer les fichiers compressés depuis votre ordinateur vers un appareil noyau Raspberry Pi, ouvrez une fenêtre de terminal sur votre ordinateur et exécutez les commandes suivantes :

```
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**
>
> Le numéro de version de cette commande doit correspondre à la version de votre package logiciel AWS IoT Greengrass Core.

Raspberry Pi web browser

Si vous avez utilisé le navigateur web du Raspberry Pi pour télécharger les fichiers compressés, ceux-ci doivent se trouver dans le fichier Pi.`~/Downloads`dossier, tel que`/home/pi/Downloads`. À défaut, les fichiers compressés doivent se trouver dans les Pi`~`dossier, tel que`/home/pi`.

3. Sur l'appareil noyau Greengrass, ouvrez une fenêtre de terminal et accédez au dossier contenant leAWS IoT GreengrassLogiciels et certificats de base. Remplacez*path-to-transferred-files*avec le chemin d'accès où vous avez transféré les fichiers sur l'appareil noyau. Par exemple, sur un Raspberry Pi, exécutez`cd /home/pi`.

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

4. DéballezAWS IoT GreengrassLogiciel Core sur l'appareil principal. Exécutez la commande suivante pour décompresser l'archive du logiciel que vous avez transférée sur l'appareil noyau. Cette commande utilise la`-C /`pour créer l'argument`/greengrass`dans le dossier racine du périphérique noyau.

   ```
   sudo tar -xzvf greengrass-OS-architecture-1.11.6.tar.gz -C /
   ```

   > **Note**
   >
   > Le numéro de version de cette commande doit correspondre à la version de votre package logiciel AWS IoT Greengrass Core.

5. Déplacez les certificats et les clés vers leAWS IoT GreengrassDossier logiciel Core. Exécutez les commandes suivantes pour créer un dossier pour les certificats et y déplacer les certificats et les clés. Remplacez*path-to-transferred-files*par le chemin d'accès où vous avez

transféré les fichiers sur le périphérique principal, et remplacez*certificateId*avec l'ID du certificat dans les noms de fichiers. Par exemple, sur un Raspberry Pi, remplacez*path-to-transferred-files*avec**/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. LeAWS IoT GreengrassLe logiciel principal utilise un fichier de configuration qui spécifie les paramètres du logiciel. Ce fichier de configuration spécifie les chemins d'accès aux fichiers de certificats et auxAWS CloudLes points de terminaison à utiliser. Au cours de cette étape, vous allez créer laAWS IoT GreengrassFichier de configuration du logiciel principal pour votre cœur. Procédez comme suit :

   a. Obtenez l'Amazon Resource Name (ARN) pour votre noyauAWS IoTUn. Procédez comme suit :

      i. DansAWS IoTconsole, UNDERGérer, UNDERAppareils Greengrass, choisissezGroupes (V1).

      ii. Dans la pageGroupes Greengrass, choisissez le groupe que vous avez créé précédemment.

      iii. UNDERPrésentation, choisissezNoyau Greengrass.

      iv. Sur la page des détails du noyau, copiez leAWS IoTARN d'objetet enregistrez-le pour l'utiliser dans leAWS IoT GreengrassFichier de configuration de Core.

   b. ObtenezAWS IoTpoint de terminaison de données de votre appareilCompte AWSdans la région actuelle. Les appareils utilisent ce point de terminaison pour se connecter àAWScommeAWS IoTthings. Procédez comme suit :

      i. DansAWS IoTconsole, choisissezParamètres.

      ii. UNDERPoint de terminaison des données, copiez lePoint de terminaisonet enregistrez-le pour l'utiliser dans leAWS IoT GreengrassFichier de configuration de Core.

   c. Création de l'AWS IoT GreengrassFichier de configuration du logiciel principal. Par exemple, vous pouvez exécuter la commande suivante pour utiliser GNU nano pour créer le fichier.

      ```
      sudo nano /greengrass/config/config.json
      ```

Remplacez le contenu du fichier par le document JSON suivant.

```
{
  "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"
      }
    }
  }
}
```

Ensuite, procédez comme suit :

- Si vous avez téléchargé un certificat d'autorité de certification racine Amazon différent de celui de l'autorité de certification racine Amazon 1, remplacez chaque instance de*AmazonRootCA1.pem*avec le nom du fichier CA racine Amazon.

- Remplacez chaque instance de*certificateId*avec l'ID du certificat dans le nom du certificat et des fichiers clés.
- Remplacez*arn:aws:IoT :région :id-compte:thingMyGreengrassCœur V1*ARN de l'élément de noyau que vous avez enregistré précédemment.
- Remplacez*MyGreengrassNoyau V1*Avec le nom de l'appareil de noyau.
- Remplacez*device-data-prefix-ats.iot.region.amazonaws.com*avec leAWS IoTPoint de terminaison de données d'appareil que vous avez enregistré précédemment.
- Remplacez*région*Avec vos recettesRégion AWS.

Pour en savoir plus sur les options de configuration que vous pouvez spécifier dans ce fichier de configuration, consultez la section.Fichier de configuration de AWS IoT Greengrass Core.

7. Vérifiez que votre appareil noyau est connecté à Internet. Ensuite, lancezAWS IoT GreengrassSur l'appareil noyau.

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

Vous devez voir un message `Greengrass successfully started`. Notez le PID du rôle.

> **Note**
>
> Pour configurer votre appareil principal de manière à lancer AWS IoT Greengrass au démarrage du système, consultez the section called “Lancement de Greengrass au démarrage système”.

Vos pouvez exécuter la commande suivante pour vérifier que le logiciel AWS IoT Greengrass Core (démon Greengrass) fonctionne. Remplacez *PID-number* par votre PID :

```
ps aux | grep PID-number
```

Vous devriez voir une entrée pour le PID avec un chemin vers le démon Greengrass en cours d'exécution (par exemple, `/greengrass/ggc/packages/1.11.6/bin/daemon`). Si vous

rencontrez des problèmes pour démarrer AWS IoT Greengrass, consultez Résolution des problèmes.

Module 3 (partie 1) : Fonctions Lambda surAWS IoT Greengrass

Ce module vous montre comment créer et déployer une fonction Lambda qui envoie des messages MQTT depuis votreAWS IoT GreengrassAppareil noyau. Le module décrit les configurations de fonctions Lambda, les abonnements utilisés pour autoriser la messagerie MQTT et les déploiements sur un appareil principal.

Module 3 (Partie 2)traite les différences entre les fonctions Lambda à la demande et longue durée s'exécutant sur leAWS IoT Greengrassnoyau.

Avant de commencer, assurez-vous d'avoir terminéModule 1etModule 2et courirAWS IoT GreengrassAppareil noyau.

> **Tip**
>
> Ou, pour utiliser un script qui configure votre appareil principal à votre place, veuillez consulter the section called “Démarrage rapide : Configuration de l'appareil Greengrass”. Le script peut également créer et déployer la fonction Lambda utilisée dans ce module.

Ce module dure environ 30 minutes.

Rubriques

- Création et empaquetage d'une fonction Lambda
- Configurez la fonction Lambda pourAWS IoT Greengrass
- Déployer des configurations cloud sur un appareil Greengrass Core
- Vérification de l'exécution de la fonction Lambda sur l'appareil principal (noyau)

Création et empaquetage d'une fonction Lambda

L'exemple de fonction Lambda Python dans ce module utilise la fonction LambdaAWS IoT GreengrassKit SDK Corepour Python pour publier des messages MQTT.

Au cours de cette étape, vous :

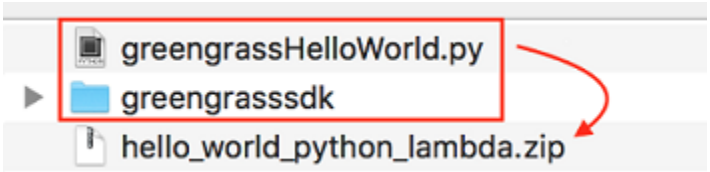
- Télécharger leAWS IoT GreengrassKit SDK Core pour Python sur votre ordinateur (et non leAWS IoT Greengrasspériphérique principal).
- Créez un package de déploiement de votre fonction Lambda qui contient le code de la fonction et les dépendances.
- Utilisez la console Lambda pour créer une fonction Lambda et télécharger le package de déploiement.
- Publiez une version de la fonction Lambda et créez un alias qui pointe vers la version.

Pour compléter ce module, Python 3.7 doit être installé sur votre appareil noyau.

1. À partir desAWS IoT GreengrassKit SDK Corepage de téléchargement, téléchargez leAWS IoT GreengrassSDK Core pour Python sur votre ordinateur.
2. Décompressez le package téléchargé pour obtenir le code de la fonction Lambda et le kit SDK.

   La fonction Lambda dans ce module utilise :

   - Le fichier `greengrassHelloWorld.py` dans `examples\HelloWorld`. Il s'agit de votre code de fonction Lambda. La fonction publie l'un des deux messages possibles toutes les cinq secondes dans la rubrique `hello/world`.
   - Le dossier `greengrasssdk`. Il s'agit du kit SDK.
3. Copiez le dossier `greengrasssdk` dans le dossier `HelloWorld` qui contient `greengrassHelloWorld.py`.
4. Pour créer le package de déploiement de votre fonction Lambda, enregistrez`greengrassHelloWorld.py`et l'`greengrasssdk`dossier vers un dossier compressé`zip`fichier dénommé`hello_world_python_lambda.zip`. Le fichier py et le dossier du `greengrasssdk` doivent être à la racine du répertoire.

   Sur les systèmes UNIX (y compris le terminal Mac), vous pouvez utiliser la commande suivante pour compresser le fichier et le dossier :

```
zip -r hello_world_python_lambda.zip greengrasssdk greengrassHelloWorld.py
```

> **Note**
>
> En fonction de votre distribution, il se peut que vous deviez installer `zip` d'abord (par exemple, en exécutant `sudo apt-get install zip`). La commande d'installation peut varier pour votre distribution.

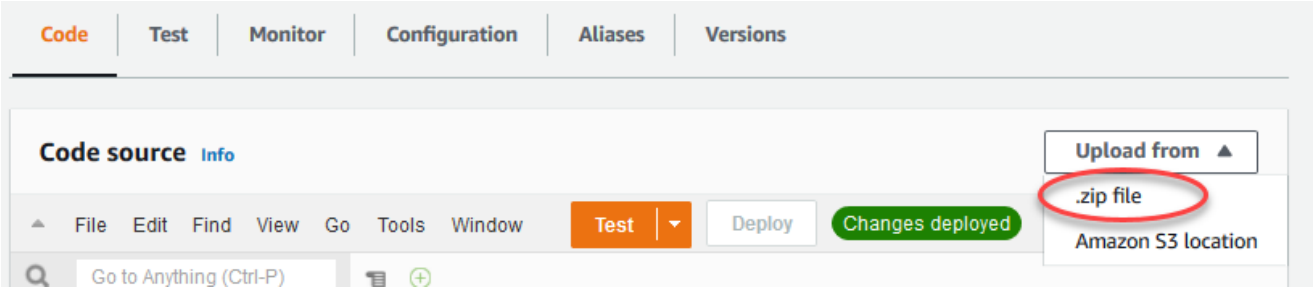
Vous êtes maintenant prêt à créer votre fonction Lambda et à télécharger le package de déploiement.

5. Ouvrez la console Lambda et choisissezCréation de fonction.
6. Choisissez Créer à partir de zéro.
7. Nommez votre fonction **`Greengrass_HelloWorld`** et définissez les champs restants comme suit :

   - Pour Runtime, sélectionnez Python 3.7.
   - PourAutorisations, conservez le paramètre par défaut. Cela crée un rôle d'exécution qui accorde des autorisations Lambda de base. Ce rôle n'est pas utilisé parAWS IoT Greengrass.

   Sélectionnez Create function (Créer une fonction).
8. Chargez le package de déploiement de votre fonction Lambda :

   a. Dans la pageCodeonglet, sousSource de code, choisissezChargement à partir des. Dans le menu déroulant, cliquez sur.fichier .zip.

   b. ChoisissezCharger, puis choisissez votre`hello_world_python_lambda.zip`package de déploiement. Ensuite, choisissez Enregistrer.

c. Dans la pageCodepour la fonction, sousParamètres d'exécution, choisissezModifier, puis entrez les valeurs suivantes.

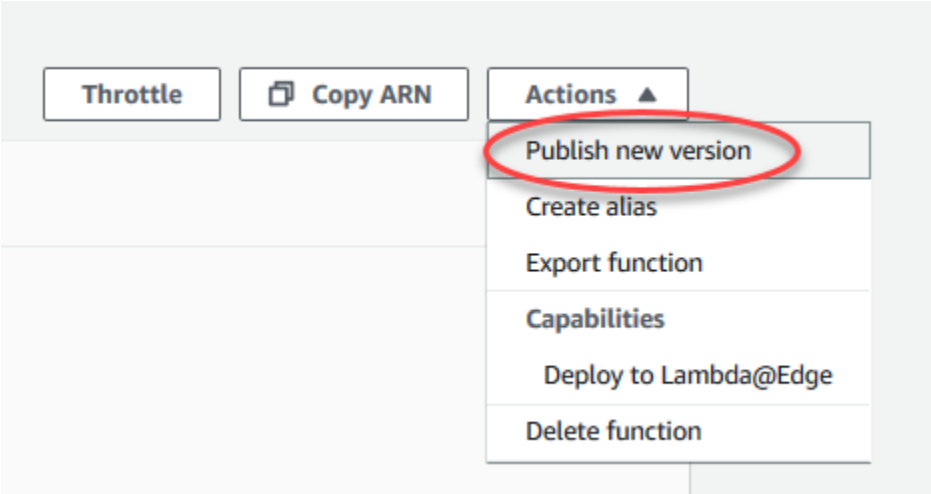
- Pour Runtime, sélectionnez Python 3.7.
- Pour Handler (Gestionnaire), entrez **`greengrassHelloWorld.function_handler`**.

d. Choisissez Save (Enregistrer).

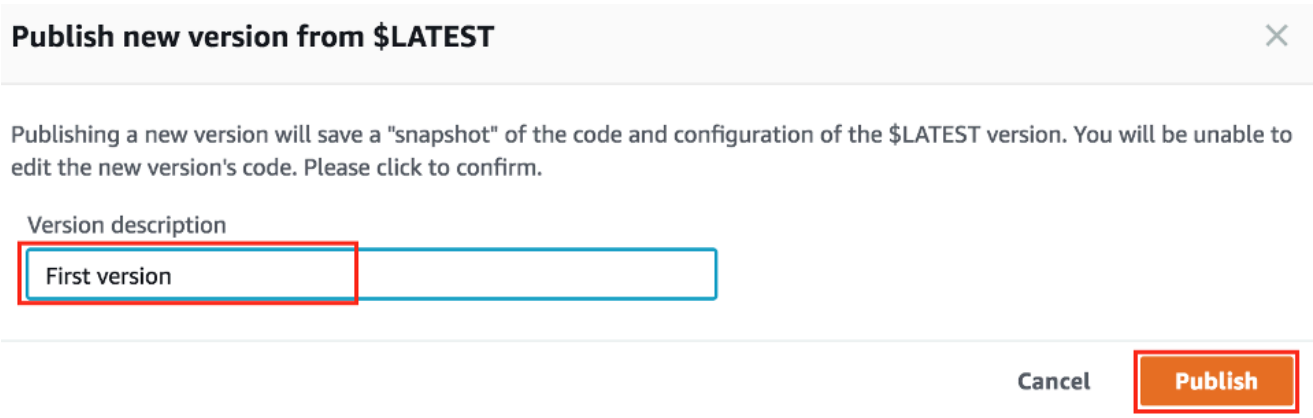
> **Note**
>
> LeTestsur le bouton.AWS Lambdane fonctionne pas avec cette fonction. LeAWS IoT GreengrassLe SDK principal ne contient pas de modules nécessaires pour exécuter vos fonctions Greengrass Lambda indépendamment dans leAWS Lambdaconsole Ces modules (par exemple,`greengrass_common`) sont fournis aux fonctions après leur déploiement dans votre cœur Greengrass.

9. Publication de la fonction Lambda :

a. À partir desActionsen haut de la page, puis choisissezPublier une nouvelle version.



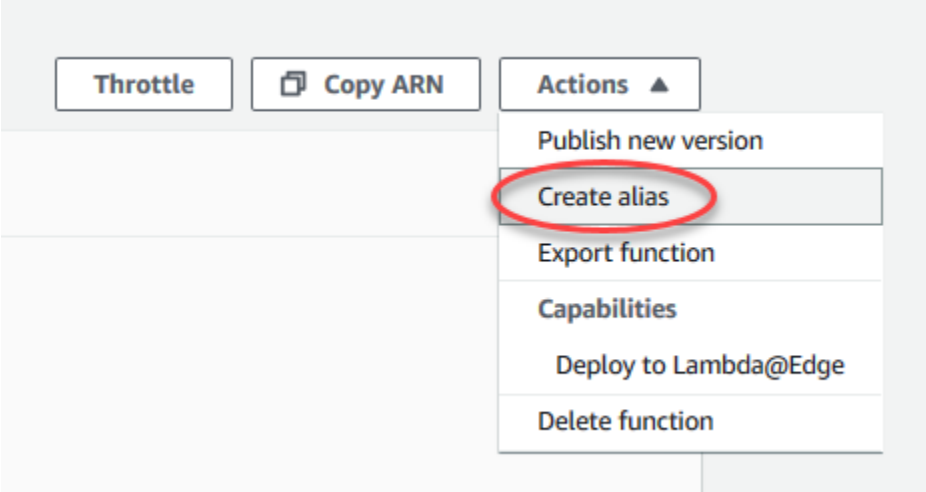
b. Dans Description de la version, saisissez **`First version`**, puis choisissez Publish.



10. Création d'unaliaspour la fonction Lambdaversion :

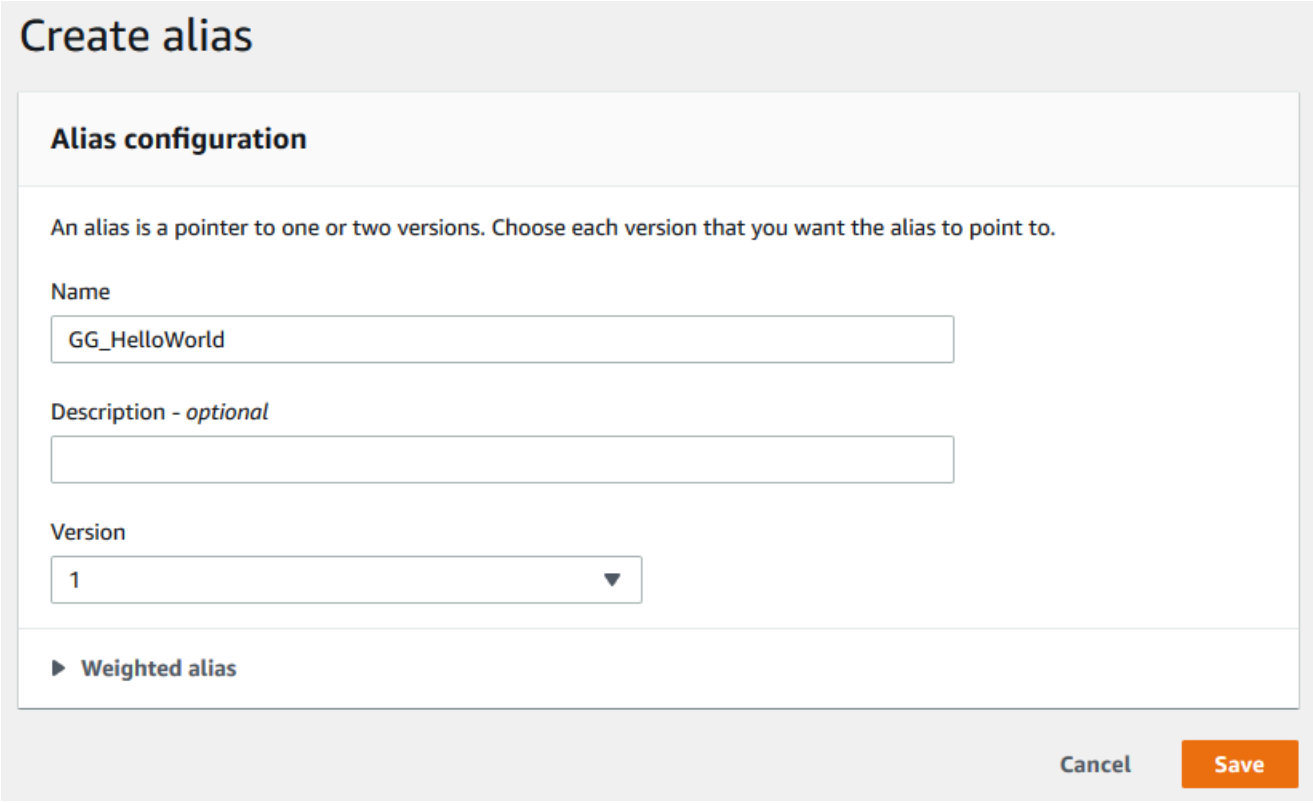
> **Note**
>
> Les groupes Greengrass peuvent référencer une fonction Lambda par alias (recommandé) ou par version. L'utilisation d'un alias facilite la gestion des mises à jour de code, car vous n'avez pas besoin de changer votre table d'abonnement ou votre définition de groupe lorsque le code de la fonction est mise à jour. Au lieu de cela, il vous suffit de pointer l'alias vers la nouvelle version de la fonction.

a. À partir desActionsen haut de la page, puis choisissezCréer un alias.



b. Nommez l'alias.**`GG_HelloWorld`**, définissez la version sur**`1`**(qui correspond à la version que vous venez de publier), puis choisissezEnregistrer.

> **Note**
>
> AWS IoT Greengrassne prend pas en charge les alias Lambda pour$LATESTversions.



Configurez la fonction Lambda pourAWS IoT Greengrass

Vous êtes maintenant prêt à configurer votre fonction Lambda pourAWS IoT Greengrass.

Au cours de cette étape, vous :

- Utilisation de l'AWS IoTpour ajouter la fonction Lambda à votre groupe Greengrass.
- Configurez les paramètres spécifiques au groupe pour la fonction Lambda.
- Ajoutez un abonnement au groupe qui permet à la fonction Lambda de publier des messages MQTT versAWS IoT.
- Configurez les paramètres du journal local pour le groupe.

1. DansAWS IoTVolet de navigation de la console sousGérer, développezAppareils Greengrass, puis choisissezGroupes (V1).
2. UnderGroupes Greengrass, choisissez le groupe que vous avez créé dansModule 2.

3. Sur la page de configuration de groupe, choisissez l'Fonctions Lambda, puis faites défiler l'écran jusqu'à la sectionFonctions My Lambdasection et choisissezAjouter une fonction Lambda.
4. Sélectionnez le nom de la fonction Lambda que vous avez créée à l'étape précédente (Herbe verte _HelloWorld, pas le nom d'alias).
5. Pour la version, choisissezAlias : GG_HelloWorld.
6. DansConfiguration de la fonction Lambda, effectuez les modifications suivantes :

   - Définir la propriétéUtilisateur et groupe du systèmepourUtiliser le groupe par défaut.
   - Définir la propriétéConteneurisation de fonction LambdapourUtiliser le groupe par défaut.
   - Dans Expiration, définissez le délai sur 25 secondes. La fonction Lambda passe en veille pendant 5 secondes avant chaque appel.
   - PourPinned, choisissezVrai.

   > **Note**
   >
   > UNlongue durée(ouépinglé) La fonction Lambda démarre automatiquement aprèsAWS IoT Greengrassdémarre et continue à fonctionner dans son propre conteneur. Cela contraste avec undemandeFonction Lambda, qui démarre quand elle est appelée et qui s'arrête lorsqu'il n'y a plus de tâches à exécuter. Pour plus d'informations, consultez the section called “Configuration du cycle de vie”.

7. ChoisissezAjouter une fonction Lambdapour enregistrer les changements. Pour plus d'informations sur les propriétés des fonctions Lambda, consultezthe section called “Contrôle de l'exécution de la fonction Greengrass Lambda”.

   Ensuite, créez un abonnement qui permet à la fonction Lambda d'envoyerMQTTMessages versAWS IoT Core.

   Une fonction Lambda Greengrass peut échanger des messages MQTT avec :

   - Appareils du groupe Greengrass.
   - Connecteurs dans le groupe.
   - D'autres fonctions Lambda du groupe.
   - AWS IoT Core.

- Service Shadow local. Pour plus d'informations, consultez the section called “Module 5 : Interaction avec les device shadows”.

Le groupe utilise des abonnements pour contrôler la façon dont ces entités peuvent communiquer entre elles. Les abonnements permettent des interactions prévisibles et offrent une couche de sécurité.

Un abonnement se compose d'une source, d'une cible et d'une rubrique. La source est le créateur du message. La cible est la destinataire du message. La rubrique vous permet de filtrer les données envoyées de la source vers la cible. La source ou la cible peut être un appareil Greengrass, une fonction Lambda, un connecteur, un device shadow ouAWS IoT Core.

> **Note**
>
> Un abonnement est dirigé dans le sens où les messages sont acheminés dans une direction spécifique : de la source vers la cible. Pour autoriser la communication bidirectionnelle, vous devez configurer deux abonnements.

> **Note**
>
> Actuellement, le filtre de rubrique d'abonnement n'autorise pas plus d'une seule+personnage dans un sujet. Le filtre de sujet n'autorise qu'un seul#personnage à la fin d'une rubrique.

`LeGreengrass_HelloWorld`La fonction Lambda envoie des messages uniquement au`hello/world`rubrique dansAWS IoT Core, vous avez donc uniquement besoin de créer un abonnement à partir de la fonction Lambda versAWS IoT Core. Vous créez ceci à l'étape suivante.

8. Sur la page de configuration de groupe, choisissez l'Subscriptionsonglet, puis choisissezAjouter un abonnement.

   Pour accéder à un exemple montrant comment créer un abonnement à l'aide de l'AWS CLI, voircreate-subscription-definitiondans leAWS CLIRéférence des commandes.

9. DansType de source, choisissezFonction Lambdaet, pour leSource, choisissezHerbe verte _HelloWorld.

10. PourTarget type (Type de cible), choisissezServiceet, pour leTargetsélectionnerCloud IoT.
11. PourFiltre de rubriques, saisissez**`hello/world`**, puis choisissezCréer un abonnement.
12. Configurez les paramètres de journalisation du groupe. Dans le cadre de ce didacticiel, vous configurezAWS IoT Greengrassles composants système et les fonctions Lambda définies par l'utilisateur pour écrire des journaux dans le système de fichiers de l'appareil principal (noyau).

    a. Sur la page de configuration de groupe, choisissez l'JournauxOnglet.

    b. DansConfiguration de journaux locauxsection, choisissezModifier.

    c. Dans la pageModifier la configuration de journaux locaux, conservez les valeurs par défaut pour les niveaux de journalisation et les tailles de stockage, puis choisissezEnregistrer.

    Vous pouvez utiliser les journaux pour résoudre les problèmes que vous pouvez rencontrer lors de l'exécution de ce didacticiel. Lorsque vous résolvez des problèmes, vous pouvez temporairement définir le niveau de journalisation sur Debug (Débogage). Pour plus d'informations, consultez the section called “Accès aux journaux du système de fichiers”.
13. Si l'environnement d'exécution Java 8 n'est pas installé sur votre appareil principal, vous devez l'installer ou désactiver le gestionnaire de flux.

    > **Note**
    >
    > Ce didacticiel n'utilise pas le gestionnaire de flux, mais il utilise le flux de travail Default Group creation (Création du groupe par défaut) qui active le gestionnaire de flux par défaut. Si le gestionnaire de flux est activé mais que Java 8 n'est pas installé, le déploiement du groupe échoue. Pour de plus amples informations, veuillez consulter les exigences relatives au gestionnaire de flux.

    Pour désactiver le gestionnaire de flux :

    a. Sur la page des paramètres du groupe, choisissez l'Fonctions LambdaOnglet.

    b. SousFonctions Lambda systèmesection, sélectionnezGestionnaire de fluxet choisissezModifier.

    c. Choisissez Disable (Désactiver), puis Save (Enregistrer).

Déployer des configurations cloud sur un appareil Greengrass Core

1. Vérifiez que votre appareil Greengrass Core est connecté à Internet. Par exemple, essayez d'accéder à une page Web.
2. Assurez-vous que le démon Greengrass est en cours d'exécution sur votre appareil principal. Dans votre terminal principal, exécutez les commandes suivantes pour vérifier que le démon est en cours d'exécution et démarrez-le, si nécessaire.

   a. Pour vérifier si le démon est en cours d'exécution :

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

   Si la sortie contient une entrée `root` pour `/greengrass/ggc/packages/1.11.6/bin/daemon`, le démon est en cours d'exécution.

   b. Pour démarrer le démon :

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

   Vous êtes prêt à déployer la fonction Lambda et les configurations d'abonnement sur votre appareil Greengrass Core.
3. DansAWS IoTVolet de navigation de la console sousGérer, DéveloppezAppareils Greengrass, puis choisissezGroupes (V1).
4. UnderGroupe Greengrass, choisissez le groupe que vous avez créé dansModule 2.
5. Sur la page de configuration de groupe, choisissezDéploiement.
6. Dans la pageFonctions Lambdaonglet, dans leFonctions Lambda du systèmesection, choisissezDétecteur IP.
7. ChoisissezModifieret SelectDétecter et remplacer automatiquement les points de terminaison des courtiers MQTT. Les appareils peuvent ainsi acquérir automatiquement des informations de connectivité pour le noyau, telles que l'adresse IP, le DNS et le numéro de port. La détection automatique est recommandée, mais AWS IoT Greengrass prend également en charge les points de terminaison spécifiés manuellement. Vous êtes uniquement invité à indiquer la méthode de découverte lors du déploiement initial du groupe.

Le premier déploiement peut durer quelques minutes. Une fois le déploiement terminé, vous devriez voir le message Successfully completed (Terminé avec succès) s'afficher dans la colonne Status (État) de la page Deployments (Déploiements) :

> **Note**
>
> L'état du déploiement est également affiché sous le nom du groupe dans l'en-tête de la page.

Pour bénéficier d'une aide à la résolution des problèmes, consultez Résolution des problèmes.

Vérification de l'exécution de la fonction Lambda sur l'appareil principal (noyau)

1. Dans le volet de navigation de laAWS IoTconsole, sousTest, choisissezClient de test MQTT.
2. Cliquez sur l'ongletS'abonner à la rubriqueonglet.
3. Saisissez**`hello/world`**dans leFiltre de rubriqueset développezConfiguration supplémentaire.
4. Entrez les informations répertoriées dans chacun des champs suivants :
   - Pour Qualité de service, choisissez 0.
   - Pour Affichage de la charge utile MQTT, choisissez Afficher les charges utiles sous forme de chaînes (plus de précision).
5. Choisissez Subscribe.

Si la fonction Lambda est en cours d'exécution sur votre appareil, elle publiera dans la rubrique des messages semblables à ce qui suit dans`hello/world`Sujet :

Bien que la fonction Lambda continue d'envoyer des messages MQTT à la`hello/world`, n'arrêtez pasAWS IoT Greengrassdémon. Les autres modules sont écrits avec l'hypothèse selon laquelle il est en cours d'exécution.

Vous pouvez supprimer la fonction et l'abonnement du groupe :

- Sur la page de configuration des groupes, sousFonctions Lambda, sélectionnez la fonction Lambda que vous souhaitez supprimer et choisissezSupprimez.
- Sur la page de configuration des groupes, sousSubscriptions, choisissez l'abonnement, puisSupprimer.

La fonction et l'abonnement sont supprimés lors du déploiement suivant.

Module 3 (partie 2) : Fonctions Lambda surAWS IoT Greengrass

Ce module aborde les différences entre les fonctions Lambda à la demande et longue durée s'exécutant sur le.AWS IoT Greengrass.

Avant de commencer, exécutez le script de configuration de l'appareil Greengrass ou assurez-vous d'avoir effectué le module 1, lemodule 2 et le module 3 (partie 1).

Ce module dure environ 30 minutes.

Rubriques

- Créer et empaqueter la fonction Lambda
- Configuration de fonctions Lambda longue durée pourAWS IoT Greengrass

- Test des fonctions Lambda longue durée
- Test des fonctions Lambda sur demande

Créer et empaqueter la fonction Lambda

Au cours de cette étape, vous :

- Créez un package de déploiement de la fonction Lambda qui contient le code de la fonction et les dépendances.
- Utilisez la console Lambda pour créer une fonction Lambda et télécharger le package de déploiement.
- Publiez une version de la fonction Lambda et créez un alias qui pointe vers la version.

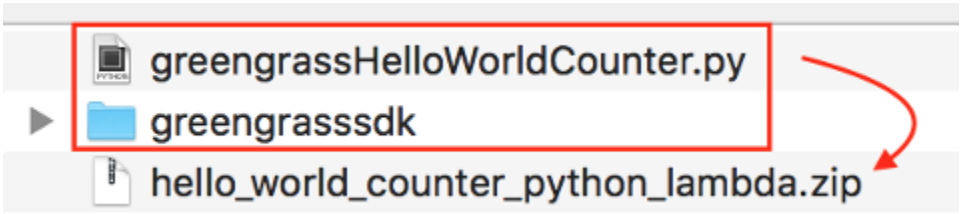
1. Sur votre ordinateur, accédez auAWS IoT GreengrassSDK principal pour Python que vous avez téléchargé et extrait dansthe section called “Création et empaquetage d'une fonction Lambda”dans le module 3-1.

   La fonction Lambda de ce module utilise :

   - Le fichier `greengrassHelloWorldCounter.py` dans `examples\HelloWorldCounter`. Il s'agit de votre code de la fonction Lambda.
   - Le dossier `greengrasssdk`. Il s'agit du kit SDK.
2. Créez un package de déploiement de votre fonction Lambda :

   a. Copiez le dossier `greengrasssdk` dans le dossier `HelloWorldCounter` qui contient `greengrassHelloWorldCounter.py`.

   b. Enregistrez `greengrassHelloWorldCounter.py` et le dossier `greengrasssdk` dans un fichier `zip` nommé `hello_world_counter_python_lambda.zip`. Le fichier py et le dossier du `greengrasssdk` doivent être à la racine du répertoire.



Pour les systèmes UNIX (y compris le terminal Mac) pour lesquels `zip` est installé, vous pouvez utiliser la commande suivante pour compresser le fichier et le dossier :

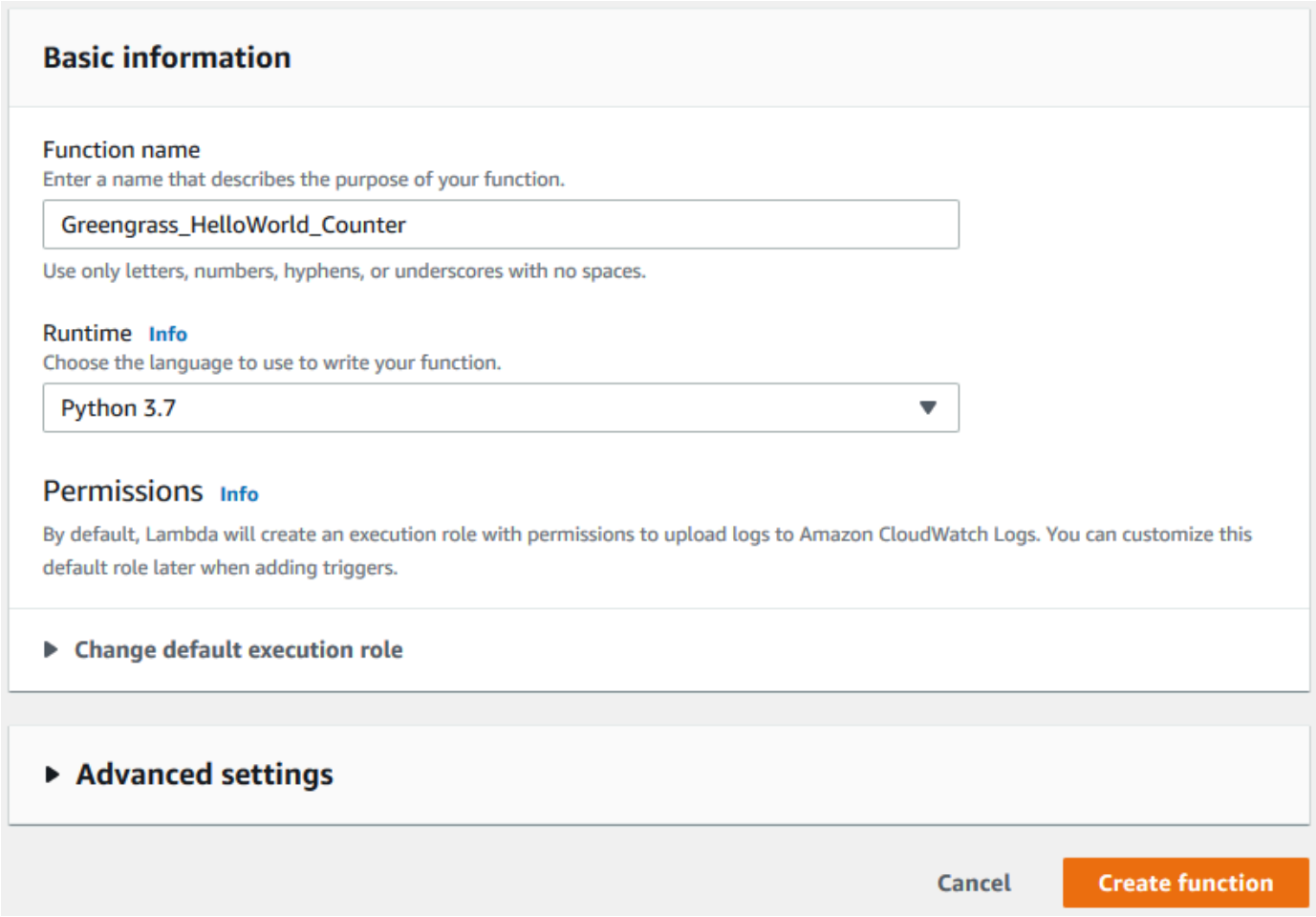
```
zip -r hello_world_counter_python_lambda.zip greengrasssdk
 greengrassHelloWorldCounter.py
```

Vous êtes maintenant prêt à créer votre fonction Lambda et à télécharger le package de déploiement.

3. Ouvrez la console Lambda et sélectionnezCréer une fonction.
4. Choisissez Créer à partir de zéro.
5. Nommez votre fonction **`Greengrass_HelloWorld_Counter`** et définissez les champs restants comme suit :

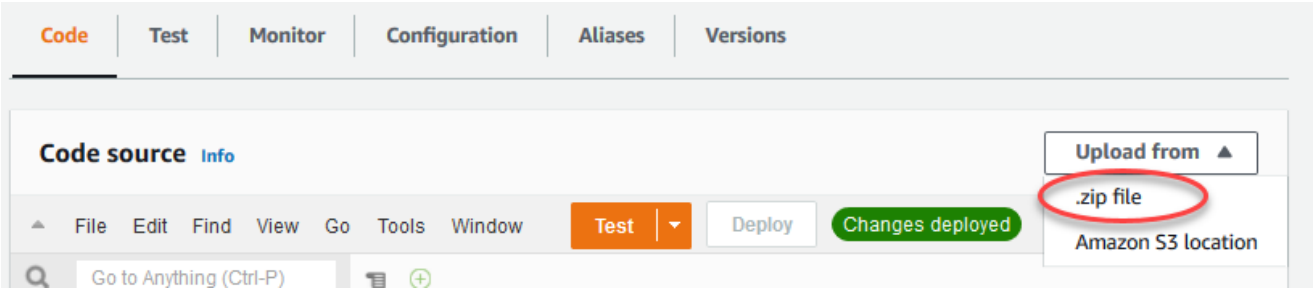
   - Pour Runtime, sélectionnez Python 3.7.
   - PourAutorisations, conservez le paramètre par défaut. Cela crée un rôle d'exécution qui accorde des autorisations Lambda de base. Ce rôle n'est pas utilisé parAWS IoT Greengrass. Vous pouvez également réutiliser le rôle que vous avez créé dans le module 3-1.

   Sélectionnez Create function (Créer une fonction).



6. Chargez votre package de déploiement de votre fonction Lambda

   a. Dans la pageCodeonglet, sousCode source, choisissezChargement à partir de. Dans le menu déroulant, sélectionnezfichier .zip.



   b. ChoisissezCharger, puis choisissez votre`hello_world_counter_python_lambda.zip`package de déploiement. Ensuite, choisissez Enregistrer.

   c. Dans la pageCodepour la fonction, sousParamètres d'exécution, choisissezModifier, puis entrez les valeurs suivantes.

- Pour Runtime, sélectionnez Python 3.7.
- Pour Handler (Gestionnaire), entrez **`greengrassHelloWorldCounter.function_handler`**.

d. Choisissez Save (Enregistrer).

> **Note**
>
> LeTestsur leAWS Lambdane fonctionne pas avec cette fonction. LeAWS IoT GreengrassLe SDK principal ne contient pas de modules nécessaires pour exécuter vos fonctions Greengrass Lambda indépendamment dans leAWS Lambdaconsole Ces modules (par exemple,`greengrass_common`) sont fournis aux fonctions après leur déploiement dans votre cœur Greengrass.

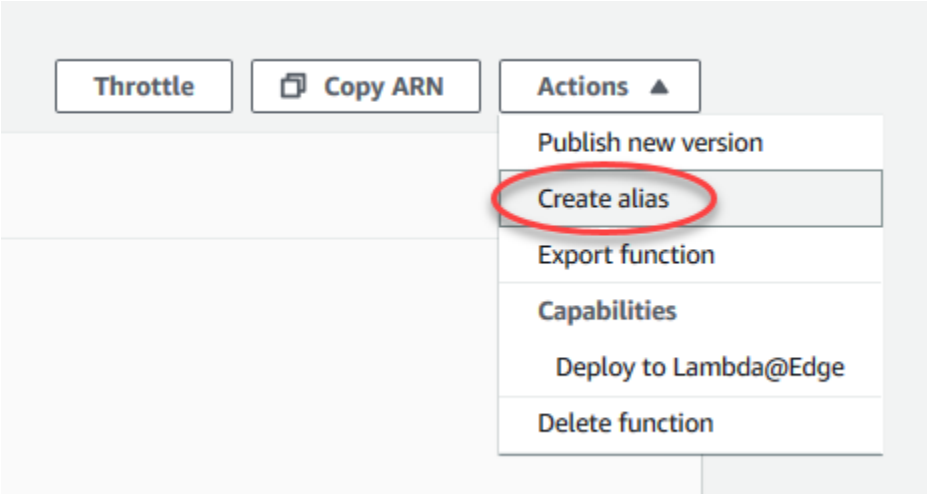
7. Publiez la première version de la fonction.

   a. À partir de laActionsdans le haut de la page, cliquez surPublier une nouvelle version. Pour Description de la version, saisissez **`First version`**.

   b. Choisissez Publish.

8. Créez un alias pour la version de la fonction.

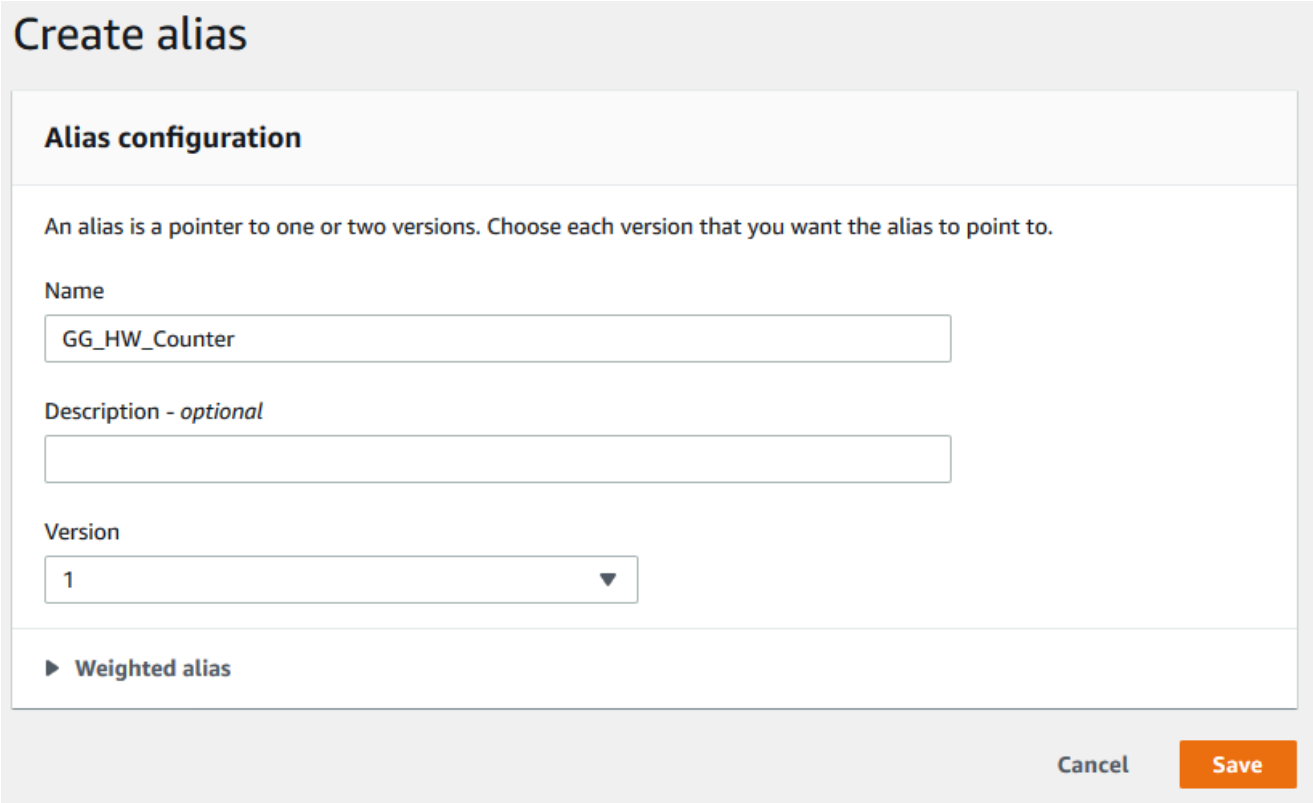
   a. À partir de laActionsdans le haut de la page, cliquez surCréer un alias.



   b. Pour Name (Nom), saisissez **`GG_HW_Counter`**.

   c. Pour Version, choisissez 1.

   d. Choisissez Save (Enregistrer).



Les alias créent une entité unique pour votre fonction Lambda à laquelle les appareils Greengrass peuvent s'abonner. De cette façon, vous n'avez pas à mettre à jour les abonnements avec de nouveaux numéros de version de la fonction Lambda chaque fois que la fonction est modifiée.

Configuration de fonctions Lambda longue durée pourAWS IoT Greengrass

Vous êtes maintenant prêt à configurer votre fonction Lambda pourAWS IoT Greengrass.

1. DansAWS IoTVolet de navigation de la console,Gérer, développezAppareils Greengrass, puis choisissezGroupes (V1).
2. UnderGroupes Greengrass, choisissez le groupe que vous avez créé dansModule 2.
3. Sur la page de configuration de groupe, choisissez leFonctions Lambdaonglet, puis sousFonctions My Lambda, choisissezAddition.
4. PourFonction Lambda, choisissezHerbe verte _HelloWorld_Compteur.
5. PourVersion de la fonction Lambda, choisissez l'alias de la version que vous avez publiée.

6. PourDélai d'expiration (secondes), saisissez**25**. La fonction Lambda passe en veille pendant 20 secondes avant chaque appel.
7. PourPinned, choisissezVrai.
8. Acceptez les valeurs par défaut pour tous les autres champs, puis choisissezAjout d'une fonction Lambda.

Test des fonctions Lambda longue durée

UNlongue duréeLa fonction Lambda démarre automatiquement lorsqueAWS IoT GreengrassCore démarre et s'exécute dans un seul conteneur (ou sandbox). Les variables et la logique de prétraitement définies à l'extérieur du gestionnaire de fonctions sont conservées pour chaque appel du gestionnaire de fonctions. Plusieurs appels du gestionnaire de fonctions sont placés en file d'attente jusqu'à ce que les appels antérieurs aient été exécutés.

Le code `greengrassHelloWorldCounter.py` utilisé dans ce module définit une variable `my_counter` en dehors du gestionnaire de fonctions.

> **Note**
>
> Vous pouvez afficher le code dans laAWS Lambdaou dans laAWS IoT GreengrassSDK de Core pour Pythonsur GitHub.

Au cours de cette étape, vous créez des abonnements qui permettent à la fonction LambdaAWS IoTpour échanger des messages MQTT. Ensuite, vous déployez le groupe et testez la fonction.

1. Sur la page de configuration de groupe, choisissezSubscriptions, puisAddition.
2. UnderType de source, choisissezFonction Lambda, puisHerbe verte_HelloWorld_Compteur.
3. UnderTarget type (Type de cible), choisissezService, choisissezCloud IoT.
4. Pour Filtre de rubrique, tapez **`hello/world/counter`**.
5. Choisissez Create subscription (Créer un abonnement).

   L'abonnement unique va dans une seule direction : à partir du`Greengrass_HelloWorld_Counter`fonction Lambda pourAWS IoT. Pour appeler (ou déclencher) cette fonction Lambda depuis le cloud, vous devez créer un abonnement dans le sens inverse.

6. Suivez les étapes 1 à 5 pour ajouter un autre abonnement utilisant les valeurs suivantes. L'abonnement autorise la fonction Lambda à recevoir des messages d'AWS IoT. Vous utilisez cet abonnement lorsque vous envoyez un message à partir duAWS IoTconsole qui appelle la fonction.

   - Pour la source, choisissezService, puisCloud IoT.
   - Pour la cible, choisissezFonction Lambda, puisHerbe verte_HelloWorld_Compteur.
   - Pour le filtre de rubriques, tapez **`hello/world/counter/trigger`**.

   L'extension `/trigger` est utilisée dans ce filtre de rubriques, car vous avez créé deux abonnements et vous ne souhaitez pas qu'ils interfèrent l'un avec l'autre.
7. Assurez-vous que le démon Greengrass est en cours d'exécution comme décrit dansDéploiement des configurations cloud sur un appareil Core.
8. Sur la page de configuration de groupe, choisissezDéploiement.
9. Une fois que votre déploiement est terminé, revenez à laAWS IoTpage d'accueil de la console et choisissezTest.
10. Configurez les champs suivants :

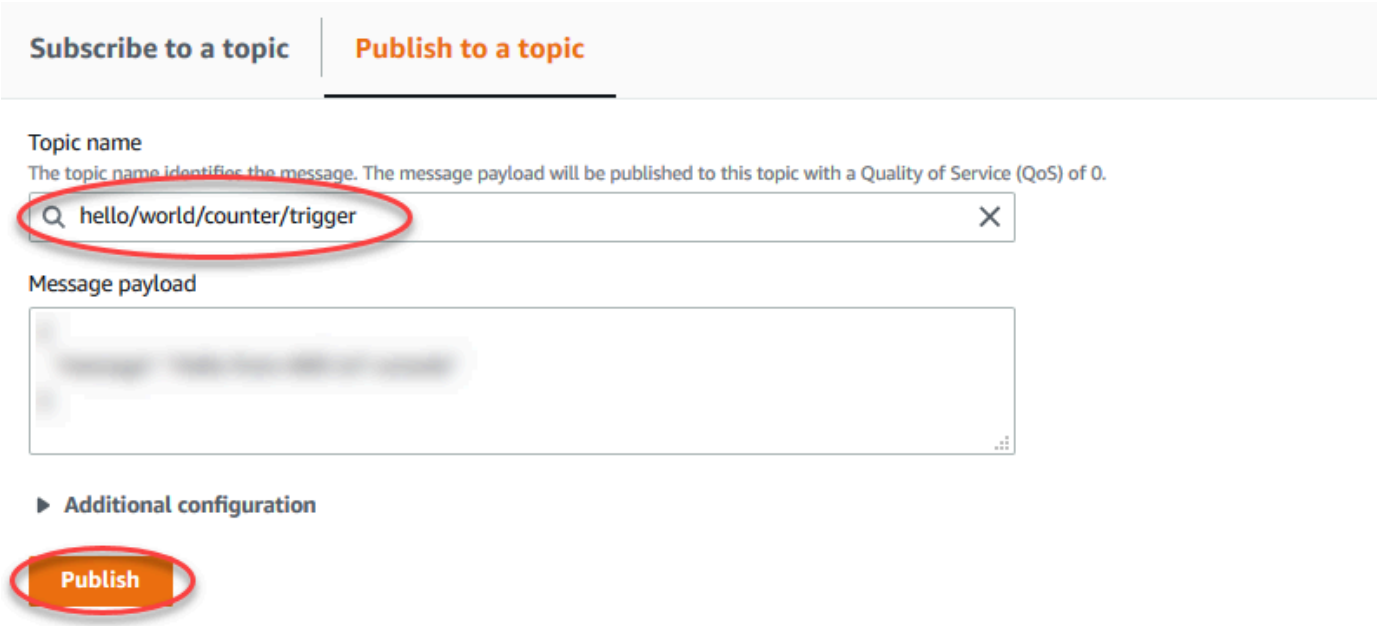
    - Pour Rubrique d'abonnement, entrez **`hello/world/counter`**.
    - Pour Qualité de service, choisissez 0.
    - Pour Affichage de la charge utile MQTT, choisissez Afficher les charges utiles sous forme de chaînes (plus de précision).
11. Choisissez Subscribe.

    Contrairement à la Partie 1 de ce module, vous ne devriez pas voir de messages après vous être abonné à `hello/world/counter`. En effet, le code `greengrassHelloWorldCounter.py` qui publie dans la rubrique `hello/world/counter` est à l'intérieur du gestionnaire de fonctions, qui s'exécute uniquement lorsque la fonction est appelée.

    Dans ce module, vous avez configuré`Greengrass_HelloWorld_Counter`fonction Lambda à appeler lorsqu'elle reçoit un message MQTT sur la`hello/world/counter/trigger`rubrique.

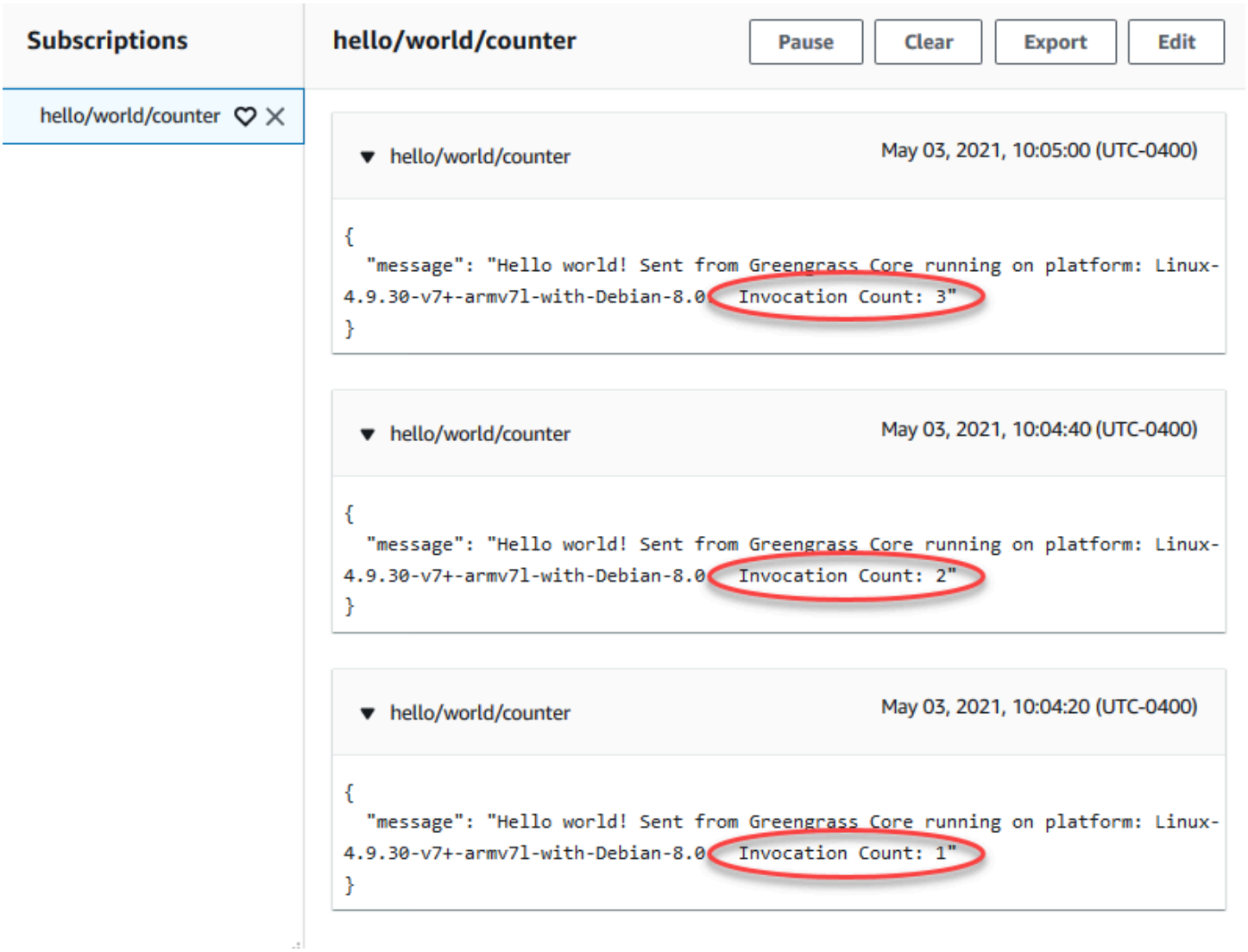
    LeHerbe verte_HelloWorld_CompteurpourCloud IoTL'abonnement autorise la fonction à d'envoyer des messages àAWS IoTsur le`hello/world/counter`rubrique. LeCloud IoTpourHerbe verte_HelloWorld_Compteurl'abonnement permetAWS IoTpour envoyer des messages à la fonction sur la`hello/world/counter/trigger`rubrique.

12. Pour tester le cycle de vie longue durée de vie, appelez la fonction Lambda en publiant un message dans la`hello/world/counter/trigger`rubrique. Vous pouvez utiliser le message par défaut.



> **Note**
>
> La fonction `Greengrass_HelloWorld_Counter` ignore le contenu des messages reçus. La fonction exécute simplement le code figurant dans `function_handler`, qui envoie un message à la rubrique `hello/world/counter`. Vous pouvez consulter ce code à partir duAWS IoT GreengrassSDK de Core pour Pythonsur GitHub.

Chaque fois qu'un message est publié dans la rubrique `hello/world/counter/trigger`, la variable `my_counter` est incrémentée. Ce nombre d'appels est indiqué dans les messages envoyés à partir de la fonction Lambda. Comme le gestionnaire de la fonction inclut un cycle de veille de 20 secondes (`time.sleep(20)`), le déclenchement répété du gestionnaire place les réponses en file d'attente depuis leAWS IoT GreengrassCœur.



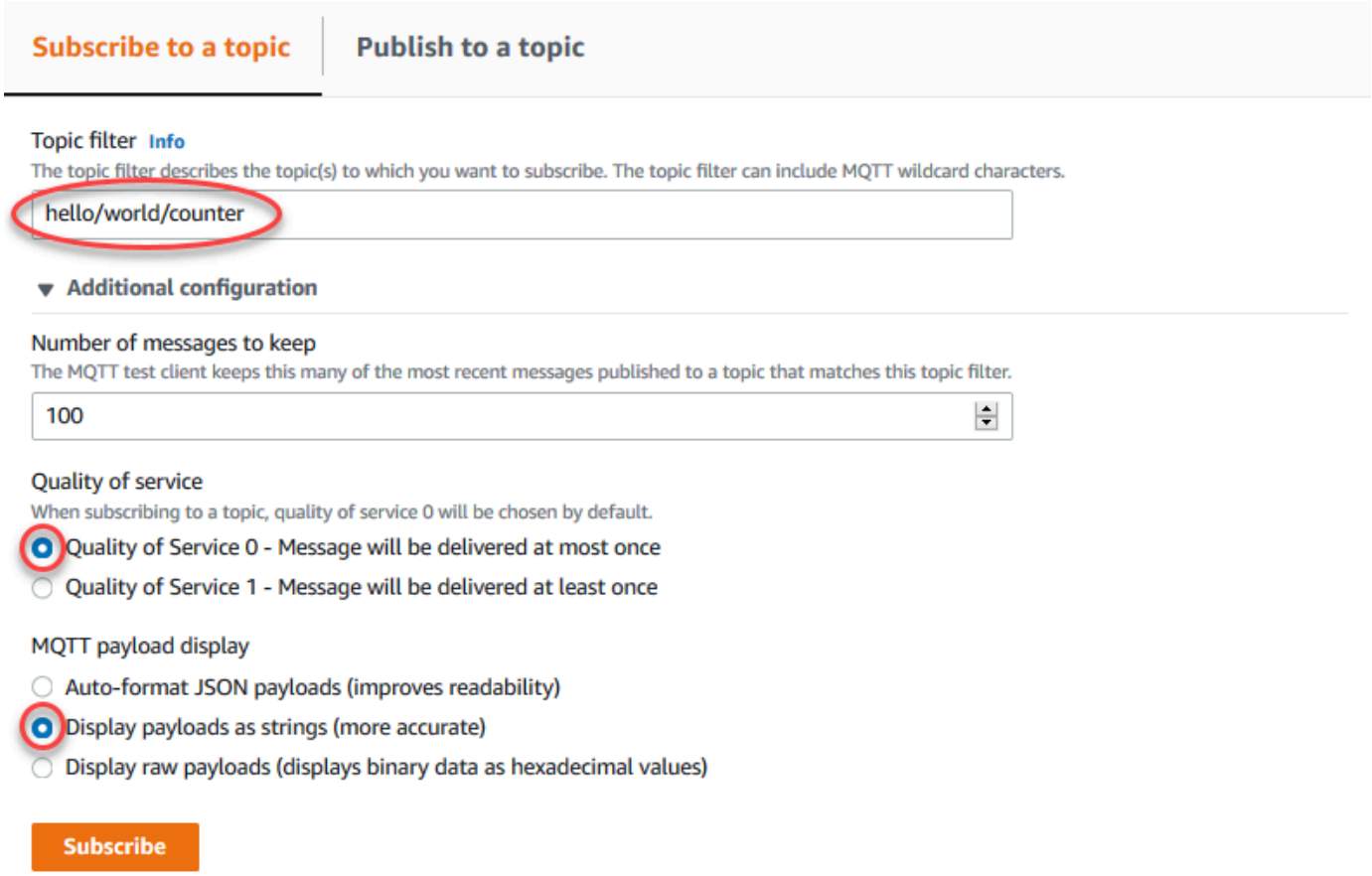
Test des fonctions Lambda sur demande

UnA LaUne fonction Lambda est similaire en termes de fonctionnalité à une fonction du cloudAWS Lambda. Plusieurs appels d'une fonction Lambda à la demande peuvent être exécutés en parallel. Un appel de la fonction Lambda crée un conteneur séparé pour traiter les appels ou réutilise un conteneur existant, si les ressources le permettent. Les variables ou le prétraitement définis en dehors du gestionnaire de fonctions ne sont pas conservés quand des conteneurs sont créés.

1. Sur la page de configuration de groupe, choisissez leFonctions LambdaOnglet.
2. UnderMes fonctions Lambda, choisissez leGreengrass_HelloWorld_CounterFonction Lambda.
3. Dans la pageGreengrass_HelloWorld_Counterpage des détails, choisissezModifier.
4. PourPinned, choisissezFaux, puis choisissezEnregistrer.

5. Sur la page de configuration de groupe, choisissezDéploiement.
6. Une fois que votre déploiement est terminé, revenez auAWS IoTpage d'accueil de la console et choisissezTest.
7. Configurez les champs suivants :

   - Pour Rubrique d'abonnement, entrez **`hello/world/counter`**.
   - Pour Qualité de service, choisissez 0.
   - Pour Affichage de la charge utile MQTT, choisissez Afficher les charges utiles sous forme de chaînes (plus de précision).

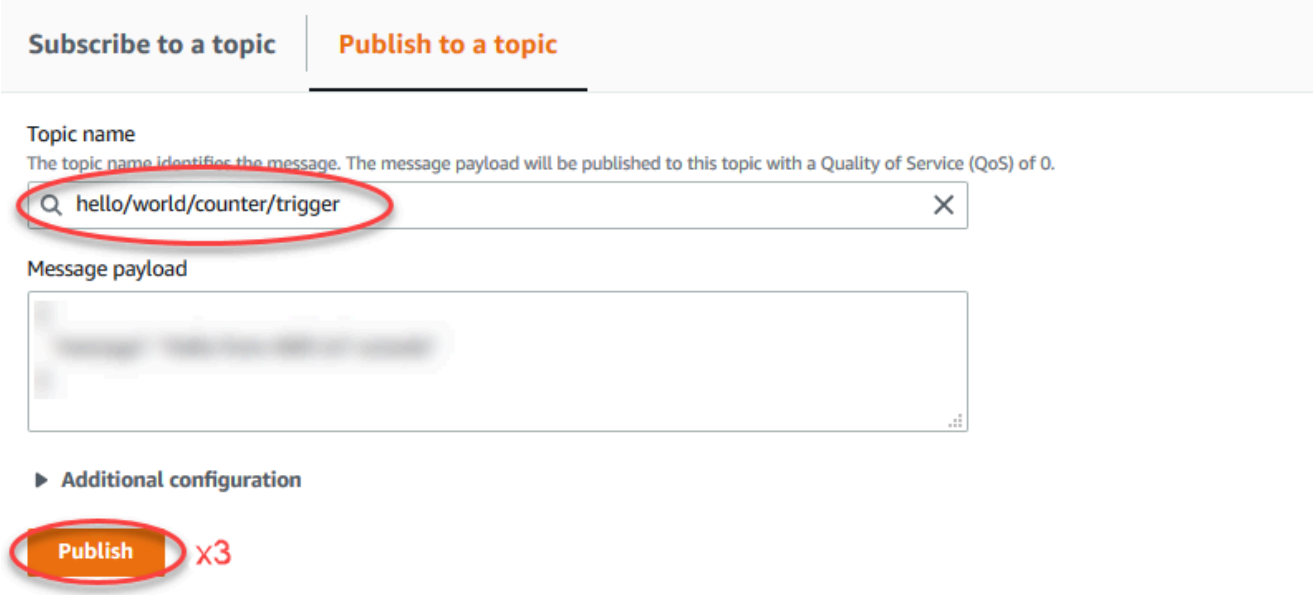


8. Choisissez Subscribe.

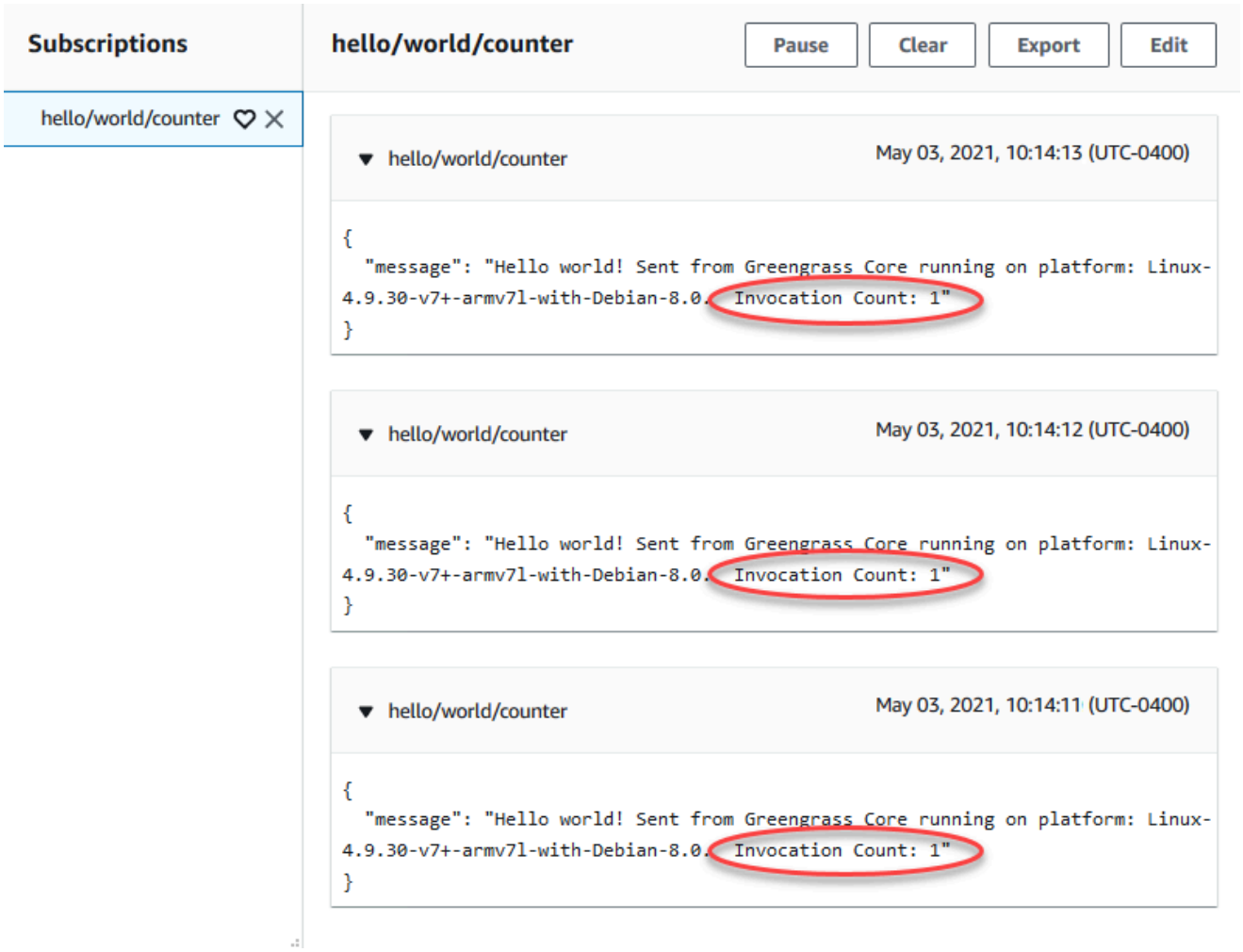
> **Note**
>
> Vous ne devriez pas voir de message après vous être abonné.

9. Pour tester le cycle de vie à la demande, appelez la fonction en publiant un message dans la rubrique `hello/world/counter/trigger`. Vous pouvez utiliser le message par défaut.

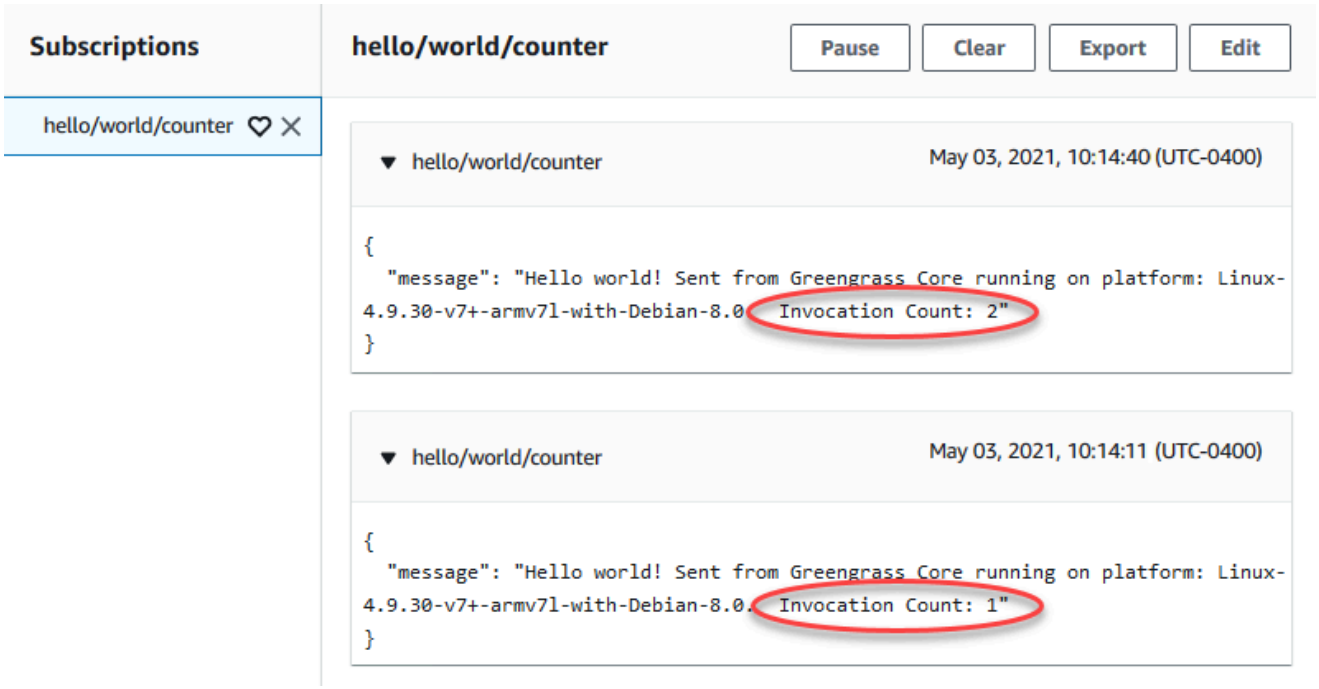
   a. ChoisissezPubliertrois fois de suite en ne dépassant pas cinq secondes entre chaque clic.



   Chaque publication appelle le gestionnaire de fonctions et crée un conteneur pour chaque appel. Le nombre d'appels n'est pas incrémenté pour chacune des trois fois où vous avez déclenché la fonction, car chaque fonction Lambda à la demande possède son propre conteneur/sandbox.



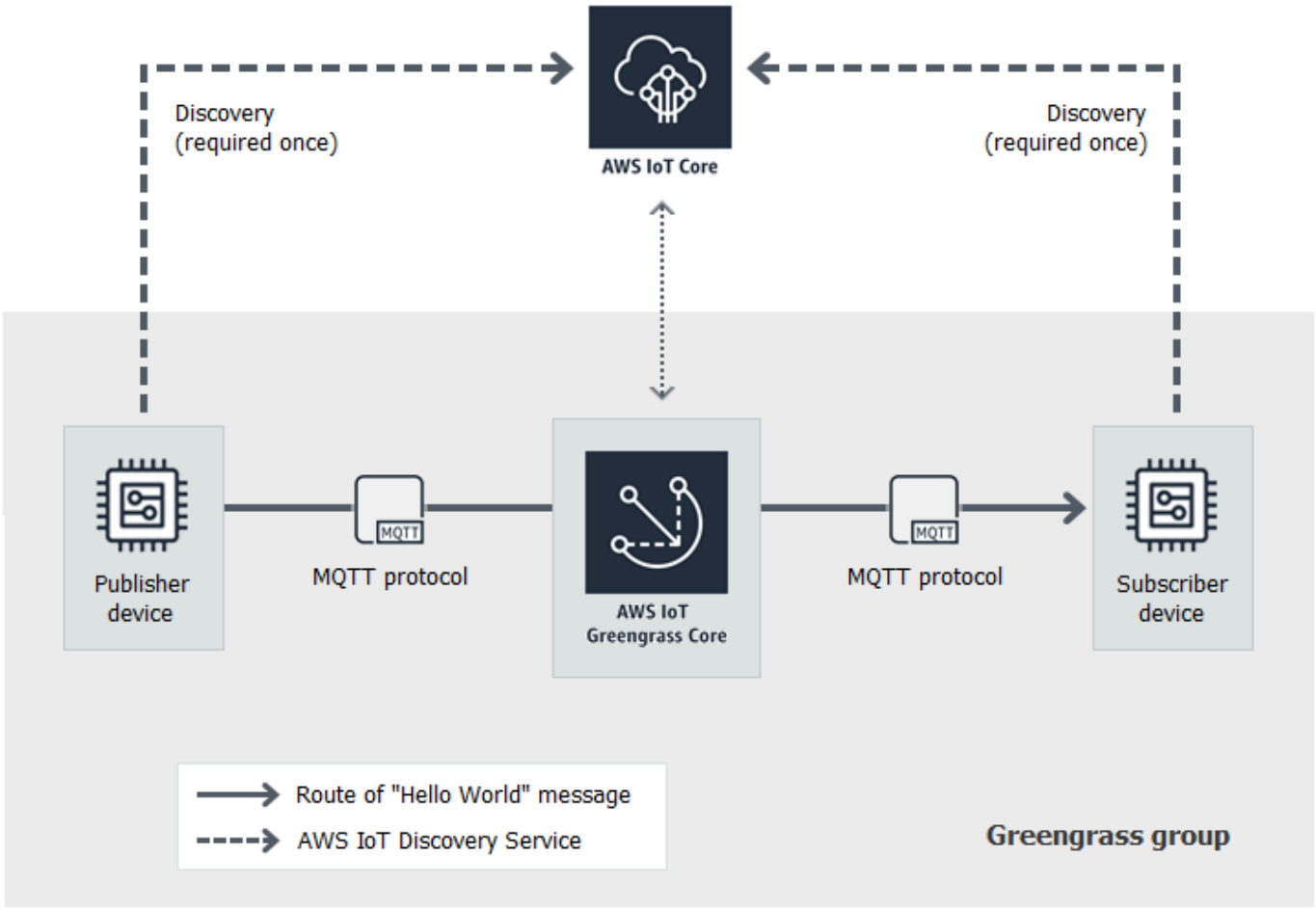
b. Après environ 30 secondes, choisissez Publier dans la rubrique. Le nombre d'appels doit être incrémenté à 2. Cela montre qu'un conteneur créé à partir d'un appel antérieur est en cours d'utilisation, et que les variables de prétraitement en dehors du gestionnaire de fonctions ont été stockées.



Vous devez maintenant comprendre les deux types de fonctions Lambda qui peuvent s'exécuter sur leAWS IoT GreengrassCore. Le module suivant,Module 4, vous montre comment les appareils IoT locaux peuvent interagir au sein d'unAWS IoT Greengrass.

Module 4 : Interaction avec les appareils clients dans unAWS IoT Greengrassgroupe

Ce module vous montre comment les appareils IoT locaux, appelésAppareils de Clientouappareils, peut se connecter et communiquer avec unAWS IoT GreengrassAppareil Core. Appareils clients qui se connectent à unAWS IoT Greengrassfont partie d'unAWS IoT Greengrasset peut participer auAWS IoT GreengrassParadigme de programmation. Dans ce module, un appareil client envoie un message Hello World à un autre appareil du groupe Greengrass.



Avant de commencer, exécutez le script de configuration de l'appareil Greengrass ou complétez le Module 1 et le Module 2. Ce module crée deux appareils de client simulés. Vous n'avez pas besoin d'autres composants ou appareils.

Le module dure approximativement 30 minutes.

Rubriques

- Créez des appareils clients dans unAWS IoT Greengrassgroupe
- Configuration des abonnements
- Installer leKit SDK des appareils AWS IoTpour Python
- Test des communications

Créez des appareils clients dans unAWS IoT Greengrassgroupe

Au cours de cette étape, vous ajoutez deux appareils clients à votre groupe Greengrass. Ce processus inclut l'enregistrement des appareils en tant queAWS IoTet configuration des certificats et des clés pour leur permettre de se connecter àAWS IoT Greengrass.

1. DansAWS IoTVolet de navigation de la console sousGérer, DéveloppezAppareils Greengrass, puisGroupes (V1).
2. Choisissez le groupe cible.
3. Sur la page de configuration de groupe, choisissezAppareils clients, puisAssocié.
4. DansAssocier un appareil client à ce groupemodal, choisissezCréation de nouveauxAWS IoTchose.

   LeCréation d'objetss'ouvre dans un nouvel onglet.
5. Dans la pageCréation d'objetspage, choisissezCréation d'objet unique, puisSuivant.
6. Dans la pageSpécifier les propriétés de, enregistrez cet appareil client en tant que**`HelloWorld_Publisher`**, puisSuivant.
7. Dans la pageConfiguration du certificat d'appareilpage, choisissezSuivant.
8. Dans la pageAttacher les stratégies au certificat, procédez de l'une des manières suivantes :
   - Sélectionnez une stratégie existante qui accorde les autorisations requises par les appareils clients, puis choisissezCréation d'objet.

     Une fenêtre modale s'ouvre dans laquelle vous pouvez télécharger les certificats et les clés que l'appareil utilise pour se connecter auAWS Cloudet.
   - Créez et joignez une nouvelle stratégie qui accorde des autorisations aux appareils clients. Procédez comme suit :

     a. Sélectionnez Create policy (Créer la stratégie).

        La page Créer une stratégie s'ouvre dans un nouvel onglet.

     b. Sur la page Créer une stratégie, procédez comme suit :

        i. PourNom de la stratégie, saisissez un nom qui décrit la stratégie, par exemple**`GreengrassV1ClientDevicePolicy`**.

        ii. Dans la pageDéclarations de stratégieonglet, sousDocument de stratégie, choisissezJASON.

iii. Saisissez le document de stratégie suivant. Cette stratégie permet à l'appareil client de découvrir les cœurs Greengrass et de communiquer sur tous les sujets MQTT. Pour découvrir comment restreindre l'accès à cette stratégie, consultezAuthentification et autorisation d'appareil pour 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. Choisissez Create (Créer) pour créer la politique.

c. Retournez à l'onglet de navigateur avecAttacher les stratégies au certificatpage ouverte. Procédez comme suit :

i. DansStratégies, sélectionnez la stratégie que vous avez créée, par exempleGreengrassV1ClientDevicePolicy.

Si vous ne voyez pas cette stratégie, sélectionnez le bouton d'actualisation.

ii. ChoisissezCréation d'objet.

Une fenêtre modale s'ouvre dans laquelle vous pouvez télécharger les certificats et les clés que l'appareil utilise pour se connecter auAWS Cloudet.

9. DansTélécharger des certificats et des clésmodal, téléchargez les certificats de l'appareil.

   > **Important**
   >
   > Avant de choisirTerminé, téléchargez les ressources de sécurité.

   Procédez comme suit :

   a. PourCertificat de l'appareil, choisissezTéléchargementpour télécharger le certificat d'appareil.

   b. PourFichier de clé publique, choisissezTéléchargementpour télécharger la clé publique du certificat.

   c. PourFichier de clé privée, choisissezTéléchargementpour télécharger le fichier de clé privée pour le certificat.

   d. VérificationAuthentification du serveurdans leAWS IoTManuel du développeuret choisissez le certificat d'autorité de certification racine approprié. Nous vous recommandons d'utiliser les points de terminaison Amazon Trust Services (ATS) et les certificats d'autorité de certification racines ATS. UnderCertificats CA racines, choisissezTéléchargementpour un certificat d'autorité de certification racine.

   e. Sélectionnez Done (Exécuté).

   Notez l'ID de certificat commun dans les noms de fichiers pour les clés et le certificat d'appareil. Vous en aurez besoin ultérieurement.

10. Retournez à l'onglet de navigateur avecAssocier un appareil client à ce groupeouvert. Procédez comme suit :

    a. PourAWS IoTnom de l'objet, choisissez leHelloWorld_Publisherquelque chose que tu as créé.

       Si vous ne voyez pas ce message, sélectionnez le bouton d'actualisation.

    b. Choisissez Associate.

11. Répétez les étapes 3 à 10 pour ajouter un deuxième appareil client au groupe.

Nommez cet appareil client**`HelloWorld_Subscriber`**. Téléchargez les certificats et les clés pour cet appareil client sur votre ordinateur. Là encore, notez l'ID du certificat commun dans les noms de fichiers pour HelloWorldAppareil _abonné.

Vous devez maintenant deux appareils clients dans votre groupe Greengrass :

- HelloWorld_Éditeur
- HelloWorld_Abonné

12. Créez un dossier sur votre ordinateur pour les informations d'identification de sécurité de ces appareils clients. Copiez les certificats et les clés dans ce dossier.

Configuration des abonnements

Au cours de cette étape, vous autorisez le HelloWorldAppareil client _Publisher pour envoyer des messages MQTT au HelloWorldAppareil client _abonné.

1. Sur la page de configuration de groupe, choisissez l'optionSubscriptions, puis choisissez, puis choisissez.Addition.
2. Dans la pageCréer un abonnement, procédez comme suit pour configurer l'abonnement :

   a. PourType de source, choisissezAppareil client, puis choisissez, puisHelloWorld_Éditeur.

   b. UnderTarget type (Type de cible), choisissezAppareil client, puis choisissez, puisHelloWorld_Abonné.

   c. Pour Filtre de rubrique, tapez **`hello/world/pubsub`**.

   > **Note**
   >
   > Vous pouvez supprimer les abonnements des modules précédents. Sur la page du groupeSubscriptions, sélectionnez les abonnements à supprimer, puis choisissezSupprimer.

   d. Choisissez Create subscription (Créer un abonnement).
3. Assurez-vous que la détection automatique est activée afin que le noyau Greengrass puisse publier une liste de ses adresses IP. Les appareils clients utilisent ces informations pour découvrir le noyau. Procédez comme suit :

   a. Sur la page de configuration de groupe, choisissez l'optionFonctions LambdaOnglet.

   b. UnderFonctions Lambda du système, choisissezDétecteur IP, puis choisissez, puisModifier.

   c. DansModifier les paramètres du détecteur IP, choisissezDétecter et remplacer automatiquement les points de terminaison des courtiers MQTT, puis choisissez, puisEnregistrer.

4. Assurez-vous que le démon Greengrass est en cours d'exécution comme décrit dansDéploiement des configurations cloud sur un appareil Core.

5. Sur la page de configuration du groupe, choisissez.Déploiement.

L'état du déploiement s'affiche sous le nom du groupe dans l'en-tête de la page. Pour consulter les détails du déploiement, choisissez l'optionDéploiementsOnglet.

Installer leKit SDK des appareils AWS IoTpour Python

Les appareils clients peuvent utiliserKit SDK des appareils AWS IoTpour Python pour communiquer avecAWS IoTetAWS IoT Greengrassappareils principaux (à l'aide du langage de programmation Python). Pour en savoir plus, notamment les exigences, consultez le kitKit SDK des appareils AWS IoTpour PythonReadmesur GitHub.

Dans cette étape, vous installez le kit SDK et obtenez le kit`basicDiscovery.py`exemple de fonction utilisée par les appareils clients simulés sur votre ordinateur.

1. Pour installer le kit SDK sur votre ordinateur, avec tous les composants nécessaires, choisissez votre système d'exploitation :

   Windows

   1. Ouvrez une invite de commande de niveau élevé et exécutez la commande suivante :

      ```
      python --version
      ```

      Si aucune information de version n'est renvoyée ou si le numéro de version est inférieur à 2.7 pour Python 2 ou à 3.3 pour Python 3, suivez les instructions indiquées à la page Downloading Python pour installer Python 2.7+ ou Python 3.3+. Pour plus d'informations, consultez la page Using Python on Windows.

   2. Télécharger leKit SDK des appareils AWS IoTpour Pythonen tant que`zip`et extrayez-le sur un emplacement approprié de votre ordinateur.

Notez le chemin d'accès au dossier `aws-iot-device-sdk-python-master` extrait qui contient le fichier `setup.py`. À l'étape suivante, ce chemin d'accès sera indiqué par *`path-to-SDK-folder`*.

3. À partir d'une invite de commande de niveau élevé, exécutez la commande suivante :

```
cd path-to-SDK-folder
python setup.py install
```

macOS

1. Ouvrez une fenêtre de terminal et exécutez la commande suivante :

```
python --version
```

Si aucune information de version n'est renvoyée ou si le numéro de version est inférieur à 2.7 pour Python 2 ou à 3.3 pour Python 3, suivez les instructions indiquées à la page Downloading Python pour installer Python 2.7+ ou Python 3.3+. Pour plus d'informations, consultez la page Using Python on a Macintosh.

2. Dans la fenêtre de terminal, exécutez les commandes suivantes pour déterminer la version OpenSSL :

```
python
>>>import ssl
>>>print ssl.OPENSSL_VERSION
```

Notez la valeur de la version OpenSSL.

> **Note**
>
> Si vous exécutez Python 3, utilisez print(ssl.OPENSSL_VERSION).

Pour fermer le shell Python, exécutez la commande suivante :

```
>>>exit()
```

Si la version OpenSSL est postérieure à 1.0.1, passez directement à l'étape 3. Sinon, suivez ces étapes :

- Depuis la fenêtre de terminal, exécutez la commande suivante afin de déterminer si l'ordinateur utilise Simple Python Version Management :

```
which pyenv
```

Si un chemin d'accès est renvoyé, choisissez l'onglet Using (Utilise) **pyenv**. Si aucun chemin n'est renvoyé, choisissez l'onglet Not using (N'utilise pas) **pyenv**.

Using pyenv

1. Consultez la page des versions Python pour Mac OS X (ou similaire) pour déterminer la version stable de Python la plus récente. Dans l'exemple suivant, cette valeur est indiquée par *latest-Python-version*.
2. À partir de la fenêtre de terminal, exécutez les commandes suivantes :

   ```
   pyenv install latest-Python-version
   pyenv global latest-Python-version
   ```

   Par exemple, si la version la plus récente de Python 2 est 2.7.14, ces commandes seront les suivantes :

   ```
   pyenv install 2.7.14
   pyenv global 2.7.14
   ```

3. Fermez, puis rouvrez une fenêtre de terminal et exécutez les commandes suivantes :

   ```
   python
   >>>import ssl
   >>>print ssl.OPENSSL_VERSION
   ```

   La version OpenSSL doit être au minimum 1.0.1. Si la version est inférieure à 1.0.1, la mise à jour a échoué. Vérifiez la version de Python utilisée dans les commandes pyenv install et pyenv global, puis réessayez.

4. Pour quitter le shell Python, exécutez la commande suivante :

```
exit()
```

Not using pyenv

1. À partir d'une fenêtre de terminal, exécutez la commande suivante pour déterminer si brew est installé :

```
which brew
```

Si aucun chemin d'accès n'est renvoyé, installez `brew` comme suit :

```
/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/
Homebrew/install/master/install)"
```

> **Note**
>
> Suivez les invites d'installation. Le téléchargement des outils de ligne de commande Xcode peut prendre un certain temps.

2. Exécutez les commandes suivantes :

```
brew update
brew install openssl
brew install python@2
```

LeKit SDK des appareils AWS IoTpour Python nécessite la version OpenSSL 1.0.1 (ou ultérieure) compilée avec le fichier exécutable Python. La commande brew install python installe un fichier exécutable `python2` qui répond à cette exigence. Le fichier exécutable `python2` est installé dans le répertoire `/usr/local/bin`, qui doit faire partie de la variable d'environnement `PATH`. Pour confirmer cela, exécutez la commande suivante :

```
python2 --version
```

Si des informations sur la version `python2` sont fournies, passez directement à l'étape suivante. Dans le cas contraire, ajoutez de façon définitive le chemin `/usr/local/bin` à votre variable d'environnement `PATH` en ajoutant la ligne suivante à votre profil shell :

```
export PATH="/usr/local/bin:$PATH"
```

Par exemple, si vous utilisez `.bash_profile` ou si vous ne possédez pas encore de profil shell, exécutez la commande suivante à partir d'une fenêtre de terminal :

```
echo 'export PATH="/usr/local/bin:$PATH"' >> ~/.bash_profile
```

Ensuite, définissez la source de votre profil shell et confirmez que les informations de version sont fournies par `python2 --version`. Par exemple, si vous utilisez `.bash_profile`, exécutez les commandes suivantes :

```
source ~/.bash_profile
python2 --version
```

Les informations de version `python2` doivent être renvoyées.

3. Ajoutez la ligne suivante à votre profil shell :

   ```
   alias python="python2"
   ```

   Par exemple, si vous utilisez `.bash_profile` ou si vous ne possédez pas encore de profil shell, exécutez la commande suivante :

   ```
   echo 'alias python="python2"' >> ~/.bash_profile
   ```

4. Ensuite, définissez la source de votre profil shell. Par exemple, si vous utilisez `.bash_profile`, exécutez la commande suivante :

   ```
   source ~/.bash_profile
   ```

   L'appel de la commande python a pour effet de lancer le fichier exécutable Python contenant la version OpenSSL requise (`python2`).

5. Exécutez les commandes suivantes :

```
python
 import ssl
 print ssl.OPENSSL_VERSION
```

La version OpenSSL doit être 1.0.1 ou une version ultérieure.

6. Pour quitter le shell Python, exécutez la commande suivante :

```
 exit()
```

3. Exécutez les commandes suivantes pour installer le kitKit SDK des appareils AWS IoTpour 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. À partir d'une fenêtre de terminal , exécutez la commande suivante :

```
python --version
```

Si aucune information de version n'est renvoyée ou si le numéro de version est inférieur à 2.7 pour Python 2 ou à 3.3 pour Python 3, suivez les instructions indiquées à la page Downloading Python pour installer Python 2.7+ ou Python 3.3+. Pour plus d'informations, consultez la page Using Python on Unix platforms.

2. Dans la fenêtre de terminal, exécutez les commandes suivantes pour déterminer la version OpenSSL :

```
python
>>>import ssl
>>>print ssl.OPENSSL_VERSION
```

Notez la valeur de la version OpenSSL.

> **Note**
>
> Si vous exécutez Python 3, utilisez print(ssl.OPENSSL_VERSION).

Pour fermer le shell Python, exécutez la commande suivante :

```
exit()
```

Si la version OpenSSL est postérieure à 1.0.1, passez directement à l'étape suivante. Dans le cas contraire, exécutez la ou les commandes pour mettre à jour OpenSSL pour votre distribution (par exemple, `sudo yum update openssl`, `sudo apt-get update`, etc.).

Confirmez que la version OpenSSL est bien 1.0.1 ou une version ultérieure en exécutant les commandes suivantes :

```
python
>>>import ssl
>>>print ssl.OPENSSL_VERSION
>>>exit()
```

3. Exécutez les commandes suivantes pour installer le kitKit SDK des appareils AWS IoTpour 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. Après leKit SDK des appareils AWS IoTpour Python est installé, accédez à`sample`set ouvrez le dossier`greengrass`folder.

   Pour ce didacticiel, vous copiez l'exemple de fonction `basicDiscovery.py`, qui utilise les certificats et les clés que vous avez téléchargés dans the section called “Créez des appareils clients dans unAWS IoT Greengrassgroupe”.

3. Copier`basicDiscovery.py`dans le dossier qui contient le fichier HelloWorld_Publisher et HelloWorldCertificats et clés de l'appareil _Abonné.

Test des communications

1. Assurez-vous que votre ordinateur et le périphérique AWS IoT Greengrass principal sont connectés à Internet via le même réseau.

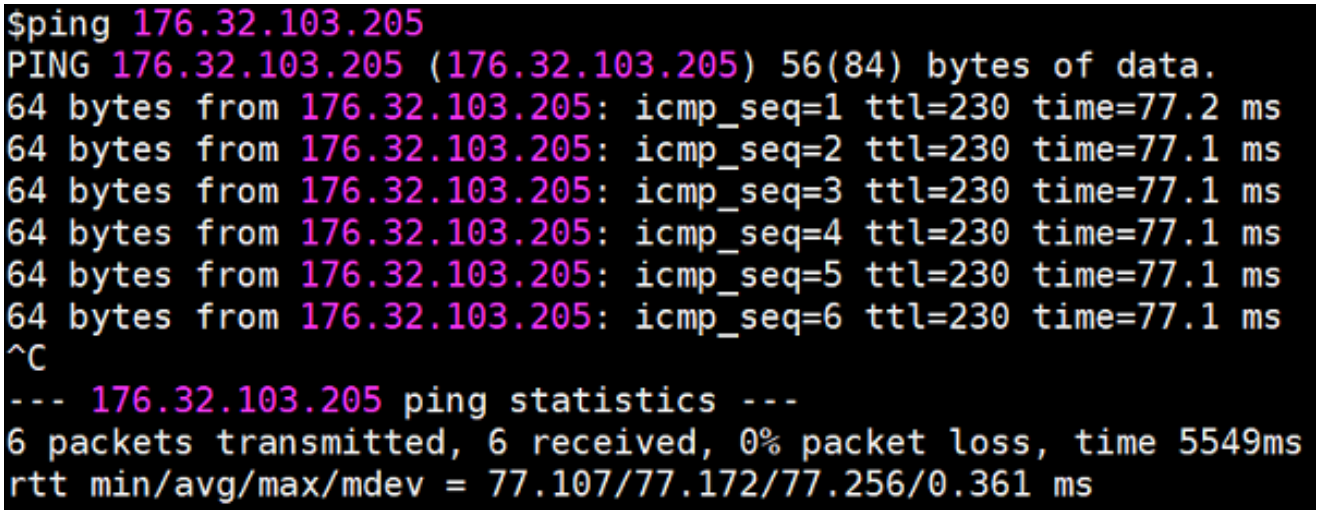
   a. Sur le périphérique AWS IoT Greengrass principal, exécutez la commande suivante pour trouver son adresse IP.

   ```
   hostname -I
   ```

   b. Sur votre ordinateur, exécutez la commande suivante à l'aide de l'adresse IP du noyau. Vous pouvez utilisez Ctrl + C pour arrêter la commande ping.

   ```
   ping IP-address
   ```

   Une sortie similaire à la suivante indique une communication réussie entre l'ordinateur et le périphérique AWS IoT Greengrass principal (0 % de perte de paquets) :

   > **Note**
   >
   > Si vous ne parvenez pas à envoyer un ping à une instance EC2 en cours d'exécution AWS IoT Greengrass, assurez-vous que les règles du groupe de sécurité entrant pour l'instance autorisent le trafic ICMP pour les messages de demande Echo. Pour plus d'informations, consultez la section Ajout de règles à un groupe de sécurité dans le guide de l'utilisateur Amazon EC2.
   > Sur les ordinateurs hôtes Windows, dans l'application Pare-feu Windows avec fonctions avancées de sécurité, vous pouvez devoir activer une règle entrante

qui autorise les demandes d'écho entrantes (par exemple, Partage de fichiers et d'imprimante (demande d'écho - ICMPv4-In)) ou en créer une.

2. Obtenez votre AWS IoT point de terminaison.

   a. Dans le volet de navigation de la AWS IoT console, sélectionnez Paramètres.

   b. Sous Point de terminaison des données de l'appareil, notez la valeur du point de terminaison. Vous allez utiliser cette valeur pour remplacer l'espace réservé *AWS_IOT_ENDPOINT* dans les commandes des étapes suivantes.

   > **Note**
   >
   > Assurez-vous que vos points de terminaison correspondent à votre type de certificat.

3. Sur votre ordinateur (et non sur le périphérique AWS IoT Greengrass principal), ouvrez deux fenêtres de ligne de commande (terminal ou invite de commande). Une fenêtre représente l'appareil client HelloWorld _Publisher et l'autre le périphérique client HelloWorld _Subscriber.

   Lors de l'exécution, `basicDiscovery.py` tente de collecter des informations sur l'emplacement du AWS IoT Greengrass noyau à ses extrémités. Ces informations sont stockées une fois que le périphérique client a découvert le cœur et s'y est connecté avec succès. Cela permet d'exécuter localement la messagerie et les opérations à venir (sans avoir besoin d'une connexion Internet).

   > **Note**
   >
   > Les ID client utilisés pour les connexions MQTT doivent correspondre au nom de l'objet du périphérique client. Le `basicDiscovery.py` script définit l'ID client pour les connexions MQTT sur le nom d'objet que vous spécifiez lorsque vous exécutez le script. Exécutez la commande suivante depuis le dossier qui contient le `basicDiscovery.py` fichier pour obtenir des informations détaillées sur l'utilisation du script :
   >
   > ```
   > python basicDiscovery.py --help
   > ```

4. Dans la fenêtre de l'appareil client HelloWorld _Publisher, exécutez les commandes suivantes.

   - Remplacez *path-to-certs-folder* par le chemin d'accès au dossier qui contient les certificats, les clés et `basicDiscovery.py`.
   - Remplacez *AWS_IOT_ENDPOINT* par votre point de terminaison.

- Remplacez les deux CertId instances d'*éditeur* par l'ID de certificat figurant dans le nom de fichier de votre appareil 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'
```

La sortie obtenue doit être similaire à la suivante, qui inclut des entrées comme `Published topic 'hello/world/pubsub': {"message": "Hello, World! Sent from HelloWorld_Publisher", "sequence": 1}`.

> **Note**
>
> Si le script renvoie un message `error: unrecognized arguments`. Remplacez les guillemets simples par des guillemets doubles pour les paramètres `--topic` et `--message`, puis exécutez à nouveau la commande.
> Pour résoudre un problème de connexion, vous pouvez essayer d'utiliser la détection manuelle des IP.

```
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. Dans la fenêtre de l'appareil client HelloWorld _Subscriber, exécutez les commandes suivantes.

   - Remplacez *path-to-certs-folder* par le chemin d'accès au dossier qui contient les certificats, les clés et `basicDiscovery.py`.
   - Remplacez *AWS_IOT_ENDPOINT* par votre point de terminaison.

- Remplacez les deux CertId instances *d'abonné* par l'ID du certificat dans le nom de fichier de votre appareil 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
```

La sortie suivante doit s'afficher. Elle inclut des entrées comme `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...
```

Fermez la HelloWorld_Publisher fenêtre pour empêcher les messages de s'y accumuler. HelloWorld_Subscriber

Les tests sur un réseau d'entreprise peuvent interférer avec la connexion au noyau. Pour contourner ce problème, vous pouvez entrer le point de terminaison. Cela garantit que le `basicDiscovery.py` script se connecte à l'adresse IP correcte du périphérique AWS IoT Greengrass principal.

Pour entrer manuellement le point de terminaison

1. Dans le volet de navigation de la AWS IoT console, sous Gérer, développez les appareils Greengrass, puis choisissez Groups (V1).
2. Dans la section Groupes Greengrass, choisissez votre groupe.
3. Configurez le noyau pour gérer manuellement les points de terminaison du broker MQTT. Procédez comme suit :

   a. Sur la page de configuration du groupe, choisissez l'onglet Fonctions Lambda.

   b. Sous Fonctions Lambda du système, sélectionnez Détecteur IP, puis Modifier.

c. Dans les paramètres Modifier le détecteur IP, choisissez Gérer manuellement les points de terminaison du broker MQTT, puis sélectionnez Enregistrer.

4. Entrez le point de terminaison du broker MQTT pour le noyau. Procédez comme suit :

   a. Sous Vue d'ensemble, choisissez le noyau Greengrass.

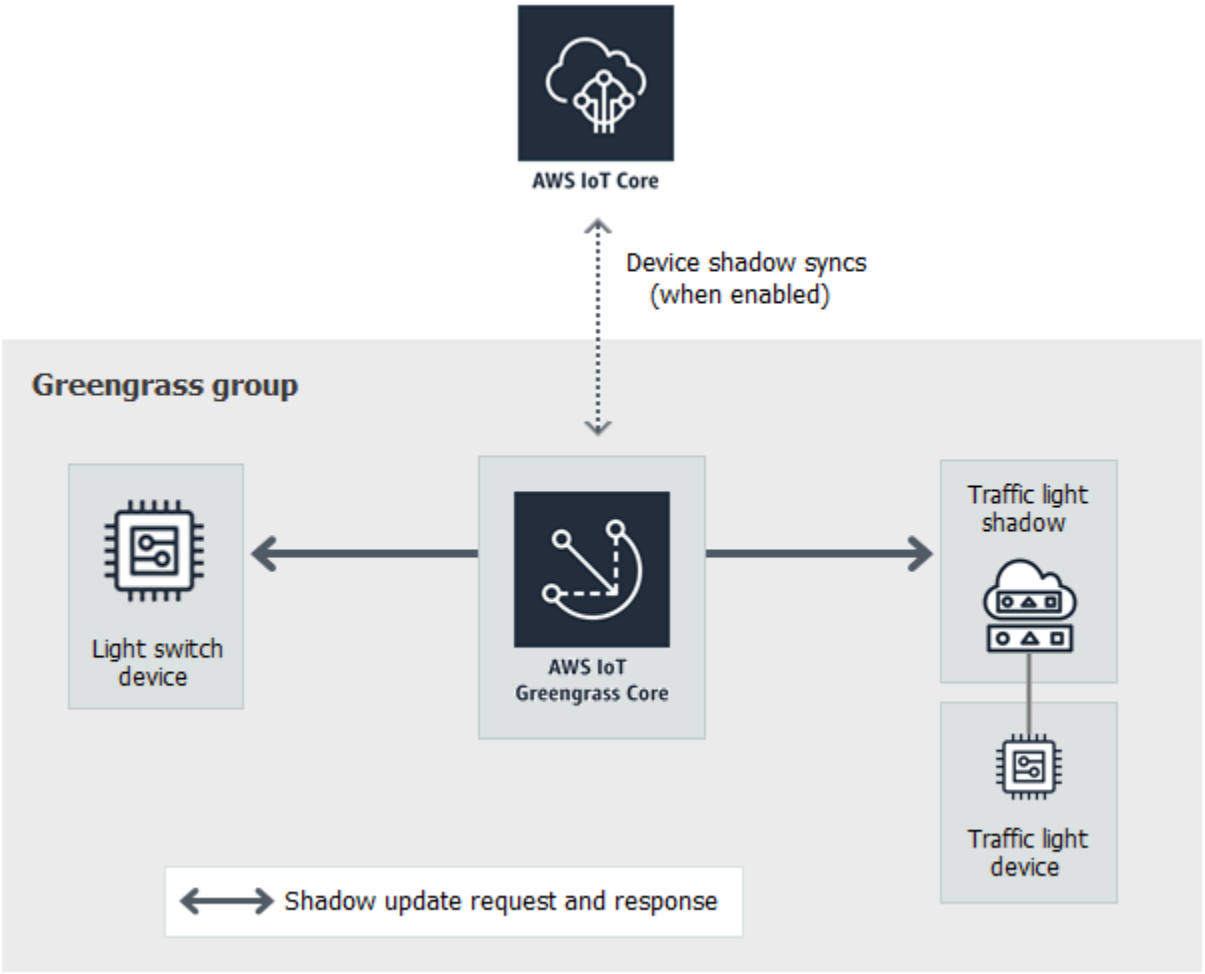
   b. Sous Points de terminaison du broker MQTT, sélectionnez Gérer les points de terminaison.

   c. Choisissez Ajouter un point de terminaison et assurez-vous de n'avoir qu'une seule valeur de point de terminaison. Cette valeur doit être le point de terminaison de l'adresse IP pour le port 8883 de votre appareil AWS IoT Greengrass principal (par exemple,`192.168.1.4`).

   d. Choisissez Mettre à jour.

Module 5 : Interaction avec les device shadows

Ce module avancé vous montre comment les appareils clients peuvent interagir avecAWS IoTDevice Shadowsdans unAWS IoT Greengrass. Un shadow est un document JSON utilisé pour stocker les informations sur l'état actuel ou souhaité d'un objet. Dans ce module, vous découvrirez comment un appareil client (`GG_Switch`) peut modifier l'état d'un autre appareil client (`GG_TrafficLight`) et comment ces états peuvent être synchronisés avec leAWS IoT Greengrasscloud :



Avant de commencer, exécutez le script de configuration de l'appareil Greengrass ou assurez-vous que vous avez effectué le Module 1 et le Module 2. Vous devez également comprendre comment connecter les appareils clients à unAWS IoT Greengrasscore (Module 4). Vous n'avez pas besoin d'autres composants ou appareils.

Ce module dure environ 30 minutes.

Rubriques

- Configuration des appareils et des abonnements
- Téléchargement des fichiers requis
- Test des communications (synchronisation des périphériques désactivée)
- Test des communications (synchronisation des appareils activée)

Configuration des appareils et des abonnements

Les ombres peuvent être synchroniséesAWS IoTlorsque leAWS IoT Greengrasscore est connecté à Internet. Dans ce module, vous allez d'abord utiliser les shadows locaux sans les synchroniser avec le cloud. Vous activerez ensuite la synchronisation du cloud.

Chaque appareil client a son propre shadow. Pour de plus amples informations, veuillez consulterService Device Shadow pourAWS IoTdans leAWS IoTManuel du développeur.

1. Sur la page de configuration du groupe, choisissez l'Appareils de clientOnglet.
2. De laAppareils de client, ajoutez deux nouveaux appareils de client dans votreAWS IoT Greengrass. Pour obtenir les étapes détaillées de ce processus, consultez the section called “Créez des appareils clients dans unAWS IoT Greengrassgroupe”.

   - Nommez les appareils de client**`GG_Switch`**et**`GG_TrafficLight`**.
   - Générez et téléchargez les ressources de sécurité pour les deux appareils de client.
   - Notez l'ID de certificat présent dans les noms de fichiers des ressources de sécurité pour les appareils de client. Vous utiliserez ces valeurs ultérieurement.
3. Créez un dossier sur votre ordinateur pour les informations de sécurité de ces appareils de client. Copiez les certificats et les clés dans ce dossier.
4. Assurez-vous que les appareils de client sont configurés pour utiliser les shadows locaux etAWS Cloud. Si ce n'est pas le cas, sélectionnez l'appareil client, choisissezSynchronisation shadow, puisDésactiver la synchronisation avec le cloud.
5. Ajoutez les abonnements du tableau suivant à votre groupe. Par exemple, pour créer le premier abonnement :

   a. Sur la page de configuration du groupe, choisissez l'Subscriptions, puisAddition.
   b. PourType de source, choisissezAppareil client, puisGG_Switch.
   c. PourTarget type (Type de cible), choisissezService, puisService Shadow local.
   d. Pour Filtre de rubriques, tapez **`$aws/things/GG_TrafficLight/shadow/update`**.
   e. Choisissez Create subscription (Créer un abonnement).

   Les rubriques doivent être entrées exactement comme indiqué dans le tableau. Bien qu'il soit possible d'utiliser des caractères génériques pour consolider certains abonnements, nous vous déconseillons cette pratique. Pour de plus amples informations, veuillez consulterRubriques shadow MQTTdans leAWS IoTManuel du développeur.

Source	Cible	Sujet	Remarques
GG_Switch	Service Shadow local	$aws/things/things/things/GG_TrafficLight/shadow/mise à jour	GG_Switch envoie une demande de mise à jour pour mettre à jour la rubrique.
Service Shadow local	GG_Switch	$aws/things/things/things/GG_TrafficLight/shadow/update/accepted	GG_Switch doit savoir si la demande de mise à jour a été acceptée.
Service Shadow local	GG_Switch	$aws/things/things/things/GG_TrafficLight/shadow/update/rejected	GG_Switch doit savoir si la demande de mise à jour a été rejetée.
GG_TrafficLight	Service Shadow local	$aws/things/things/things/GG_TrafficLight/shadow/mise à jour	Le GG_TrafficLight envoie une mise à jour de son état à la rubrique de mise à jour.
Service Shadow local	GG_TrafficLight	$aws/things/things/things/GG_TrafficLight/shadow/update/delta	Le service shadow local envoie une mise à jour reçue à GG_TrafficLight via le sujet Delta.
Service Shadow local	GG_TrafficLight	$aws/things/things/things/GG_TrafficLight/shadow/update/accepted	Le GG_TrafficLight doit savoir si la mise à jour de l'état a été acceptée.

Source	Cible	Sujet	Remarques
Service Shadow local	GG_TrafficLight	$aws/things/things/things/GG_TrafficLight/shadow/update/rejected	Le GG_TrafficLight doit savoir si la mise à jour de l'état a été refusée.

Les nouveaux abonnements s'affichent sur la pageSubscriptionsOnglet.

> **Note**
>
> Pour de plus amples informations sur le caractère $, veuillez consulter Rubriques réservées.

6. Assurez-vous que la détection automatique est activée afin que le noyau Greengrass puisse publier une liste de ses adresses IP. Les appareils de client utilisent ces informations pour découvrir le noyau. Procédez comme suit :

   a. Sur la page de configuration du groupe, choisissez l'Fonctions LambdaOnglet.

   b. UnderFonctions Lambda du système, choisissezDétecteur IP, puisModifier.

   c. DansModifier les paramètres du détecteur IP, choisissezDétecter et remplacer automatiquement les points de terminaison des courtiers MQTT, puisEnregistrer.

7. Assurez-vous que le démon Greengrass est en cours d'exécution comme décrit dansDéploiement des configurations cloud sur un appareil Core.

8. Sur la page de configuration du groupe, choisissezDéploiement.

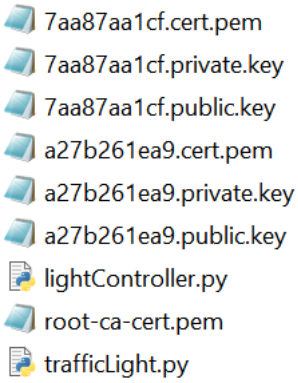
Téléchargement des fichiers requis

1. Si vous ne ne ne ne ne ne ne ne ne ne ne ne neKit SDK des appareils AWS IoTpour Python. Pour obtenir des instructions, consultez l'étape 1 dans the section called “Installer leKit SDK des appareils AWS IoTpour Python”.

   Le kit SDK est utilisé par les appareils de client pour communiquer avecAWS IoTet avecAWS IoT GreengrassAppareils Core.

2. à partir des TrafficLightdossier d'exemples sur GitHub, téléchargez `lelightController.pyettrafficLight.py`sur votre ordinateur. Enregistrez-les dans le dossier qui contient GG_Switch et GG_TrafficLight les certificats et clés des appareils clients.

   `LelightController.py`correspond à l'appareil client GG_Switch, tandis que `letrafficLight.py`le script correspond à la GG_TrafficLight appareil client.

   > **Note**
   >
   > Les exemples de fichiers Python sont stockés dans leAWS IoT GreengrassPour plus de commodités, mais ils n'utilisent pas le kit SDK Core pour plus de commodités, mais ils n'AWS IoT GreengrassKit SDK Core.

Test des communications (synchronisation des périphériques désactivée)

1. Assurez-vous que votre ordinateur et le périphérique AWS IoT Greengrass principal sont connectés à Internet via le même réseau.

   a. Sur le périphérique AWS IoT Greengrass principal, exécutez la commande suivante pour trouver son adresse IP.

   ```
   hostname -I
   ```

   b. Sur votre ordinateur, exécutez la commande suivante à l'aide de l'adresse IP du noyau. Vous pouvez utilisez Ctrl + C pour arrêter la commande ping.

   ```
   ping IP-address
   ```

Un résultat similaire à ce qui suit indique une communication réussie entre l'ordinateur et le périphérique AWS IoT Greengrass principal (0 % de perte de paquets) :

```
$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**
>
> Si vous ne parvenez pas à envoyer un ping à une instance EC2 en cours d'exécution AWS IoT Greengrass, assurez-vous que les règles du groupe de sécurité entrant pour l'instance autorisent le trafic ICMP pour les messages de demande Echo. Pour plus d'informations, consultez la section Ajout de règles à un groupe de sécurité dans le guide de l'utilisateur Amazon EC2.
> Sur les ordinateurs hôtes Windows, dans l'application Pare-feu Windows avec fonctions avancées de sécurité, vous pouvez devoir activer une règle entrante qui autorise les demandes d'écho entrantes (par exemple, Partage de fichiers et d'imprimante (demande d'écho - ICMPv4-In)) ou en créer une.

2. Obtenez votre AWS IoT point de terminaison.

   a. Dans le volet de navigation de la AWS IoT console, sélectionnez Paramètres.

   b. Sous Point de terminaison des données de l'appareil, notez la valeur du point de terminaison. Vous allez utiliser cette valeur pour remplacer l'espace réservé *AWS_IOT_ENDPOINT* dans les commandes des étapes suivantes.

> **Note**
>
> Assurez-vous que vos points de terminaison correspondent à votre type de certificat.

3. Sur votre ordinateur (et non sur le périphérique AWS IoT Greengrass principal), ouvrez deux fenêtres de ligne de commande (terminal ou invite de commande). Une fenêtre représente le périphérique client GG_Switch et l'autre représente le périphérique client GG_TrafficLight .

   a. Dans la fenêtre de l'appareil client GG_Switch, exécutez les commandes suivantes.

   - Remplacez *path-to-certs-folder* par le chemin d'accès au dossier qui contient les certificats, clés et fichiers Python.
   - Remplacez *AWS_IOT_ENDPOINT* par votre point de terminaison.
   - Remplacez les deux CertId instances de *commutateur* par l'ID du certificat dans le nom de fichier de votre appareil 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. Dans la fenêtre de l'appareil TrafficLight client GG_, exécutez les commandes suivantes.

   - Remplacez *path-to-certs-folder* par le chemin d'accès au dossier qui contient les certificats, clés et fichiers Python.
   - Remplacez *AWS_IOT_ENDPOINT* par votre point de terminaison.
   - Remplacez les deux CertId instances *light* par l'ID du certificat dans le nom de fichier de votre appareil TrafficLight client GG_.

```
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
```

   Toutes les 20 secondes, le commutateur met à jour l'état du shadow sur V, J et R, et le feu affiche son nouvel état, comme illustré ci-après.

   Sortie 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
~~~~~~~~~~~~~~~~~~~~~~~~~
```

TrafficLight Sortie GG_ :

```
++++++++ Received Shadow Delta ++++++++++
{u'state': {u'property': u'R'}, u'metadata': {u'property': {u'timestamp': 1545337381}}, u'version': 33, u'clientToken':
u'3b22e27c-930d-4c6a-8562-9f86088249f4'}
property: R
version: 33
++++++++++++++++++++++


Light changed to: R
{"state":{"reported":{"property":"R"}}}
2018-12-20 12:23:01,539 - AWSIoTPythonSDK.core.protocol.mqtt_core - INFO - Performing sync publish...
~~~~~~~~~~ Shadow Update Accepted ~~~~~~~~~~~~~~
Update request with token: f552109f-c1c2-4ae6-a841-8443506eefcb accepted!
property: R
~~~~~~~~~~~~~~~~~~~~~~~~~
```

Lorsqu'il est exécuté pour la première fois, chaque script de périphérique client exécute le service de AWS IoT Greengrass découverte pour se connecter au AWS IoT Greengrass cœur (via Internet). Une fois qu'un appareil client a découvert et s'est connecté avec succès au AWS IoT Greengrass cœur, les futures opérations peuvent être exécutées localement.

> **Note**
>
> Les scripts `lightController.py` et `trafficLight.py` stockent les informations de connexion dans le dossier `groupCA`, qui est créé dans le même dossier que les scripts. Si vous recevez des erreurs de connexion, assurez-vous que l'adresse IP du `ggc-host` fichier correspond au point de terminaison de l'adresse IP de votre cœur.

4. Dans la AWS IoT console, choisissez votre AWS IoT Greengrass groupe, choisissez l'onglet Appareils clients, puis choisissez GG_ TrafficLight pour ouvrir la page des détails de l' AWS IoT appareil client.
5. Choisissez l'onglet Device Shadows. Une fois que le GG_Switch a changé d'état, aucune mise à jour ne devrait être apportée à cette ombre. C'est parce que le GG_ TrafficLight est configuré pour désactiver la synchronisation des ombres avec le cloud.
6. Appuyez sur Ctrl + C dans la fenêtre de l'appareil client GG_Switch (`lightController.py`). Vous devriez voir que la fenêtre GG_ TrafficLight (`trafficLight.py`) cesse de recevoir des messages de changement d'état.

Ces fenêtres doivent rester ouvertes, afin que vous puissiez exécuter les commandes de la section suivante.

Test des communications (synchronisation des appareils activée)

Pour ce test, vous configurez GG_TrafficLight ombre de l'appareil avec laquelle se synchroniserAWS IoT. Vous exécutez les mêmes commandes que celles du test précédent, mais cette fois l'état du shadow dans le cloud est mis à jour lorsque GG_Switch envoie une demande de mise à jour.

1. DansAWS IoTconsole, choisissez votreAWS IoT Greengrass, puis choisissez laAppareils de clientOnglet.
2. Sélectionnez la GG_TrafficLight appareil, choisissezshadow, puis choisissezActiver la synchronisation des ombres avec le cloud.

   Vous devez recevoir une notification selon laquelle l'état du Synchronisation shadow de l'appareil a été mis à jour.
3. Sur la page de configuration du groupe, choisissezDéploiement.
4. Dans vos deux fenêtres de ligne de commande, exécutez les commandes du test précédent pour laGG_SwitchetGG_TrafficLightappareils de client
5. Vérifiez maintenant l'état du shadow dans laAWS IoTconsole Choisissez votreAWS IoT Greengrass, choisissez laAppareils de clientonglet, choisissezGG_TrafficLight, choisissez leDevice Shadows, puis choisissezShadow classique.

   Parce que vous avez activé la synchronisation du GG_TrafficLight shadowAWS IoT, l'état du shadow dans le cloud doit être mis à jour chaque fois que GG_Switch envoie une mise à jour. Cette fonctionnalité peut être utilisée pour exposer l'état d'un appareil clientAWS IoT.

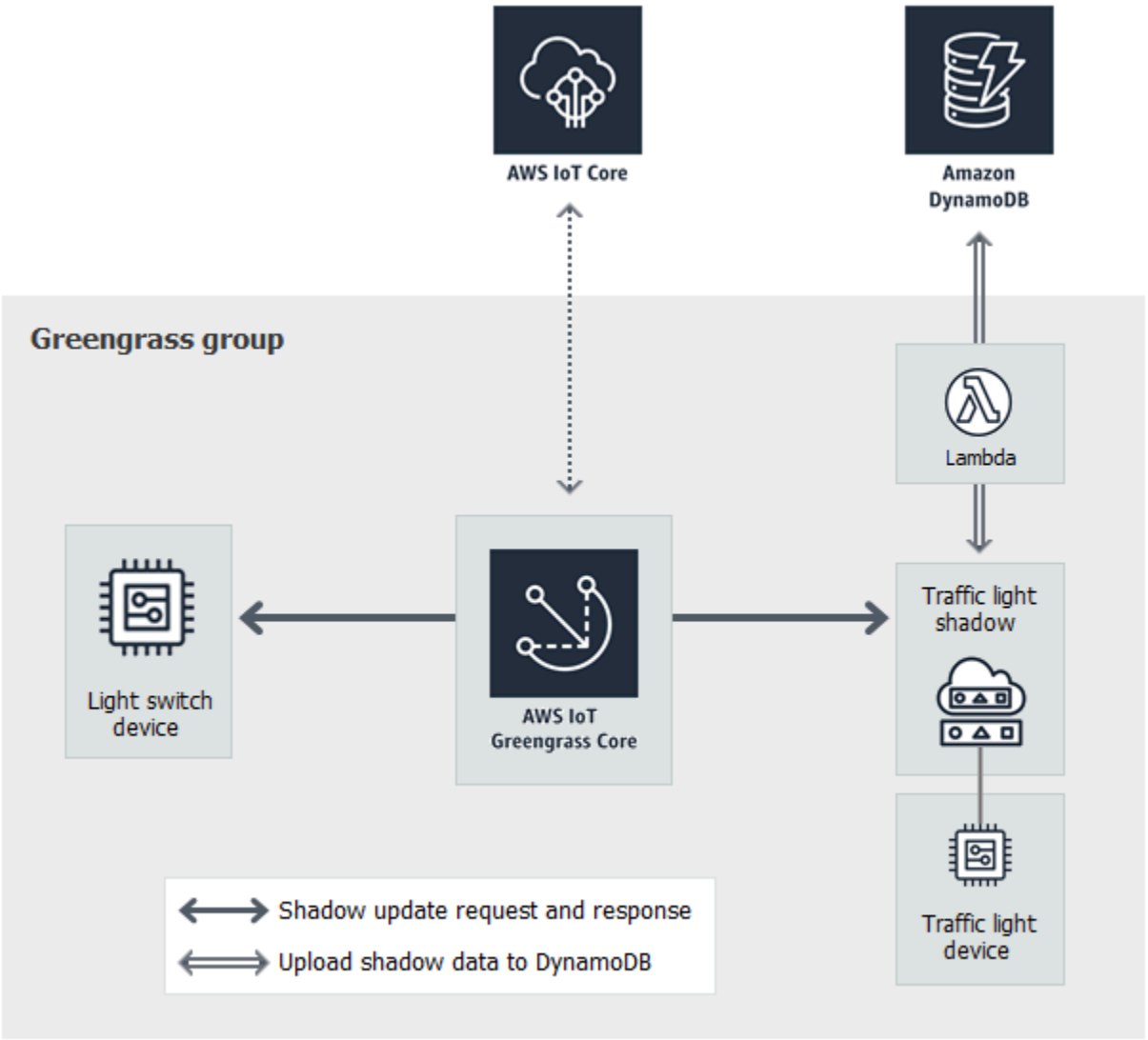
   > **Note**
   >
   > Si nécessaire, vous pouvez résoudre les problèmes en affichant laAWS IoT Greengrassjournaux de base, en particulier`runtime.log` :
   >
   > ```
   > cd /greengrass/ggc/var/log
   > sudo cat system/runtime.log | more
   > ```

Vous pouvez également consulter les fichiers `GGShadowSyncManager.log` et `GGShadowService.log`. Pour plus d'informations, consultez Résolution des problèmes.

Conservez la configuration des appareils de client et des abonnements. Vous les utiliserez dans le module suivant. Vous exécutez également les mêmes commandes.

Module 6 : Accès à d'autresAWSservices

Ce module avancé vous montre comment le faireAWS IoT Greengrassles cœurs peuvent interagir avec d'autresAWSservices dans le cloud. Il s'appuie sur l'exemple de feux de signalisation deModule 5et ajoute une fonction Lambda qui traite les états du shadow et télécharge un résumé sur la table Amazon DynamoDB.



Avant de commencer, exécutez le script de configuration de l'appareil Greengrass ou assurez-vous que vous avez effectué le Module 1 et le Module 2. Vous devriez également effectuer le module 5. Vous n'avez pas besoin d'autres composants ou appareils.

Ce module dure environ 30 minutes.

> **Note**
>
> Ce module crée et met à jour une table dans DynamoDB. Bien que la plupart des opérations soient de petite envergure et relèvent du niveau gratuit Amazon Web Services, l'exécution de

certaines des étapes décrites dans ce module peut entraîner des frais sur votre compte. Pour plus d'informations sur la tarification, consultezDocumentation sur la tarification DynamoDB.

Rubriques

Configuration du rôle de groupe

Le rôle de groupe est unRôle IAMque vous créez et attachez à votre groupe Greengrass.
Ce rôle contient les autorisations qui ont déployé des fonctions Lambda (et d'autresAWS IoT Greengrassfonctionnalités) à utiliser pour accéderAWSServices . Pour plus d'informations, consultez the section called “Rôle de groupe Greengrass”.

Vous utilisez les étapes générales suivantes pour créer un rôle de groupe dans la console IAM.

1. Créez une stratégie qui autorise ou refuse des actions sur une ou plusieurs ressources.
2. Créez un rôle qui utilise le service Greengrass en tant qu'entité approuvée.
3. Attachez votre stratégie au rôle.

Ensuite, dans leAWS IoTconsole, vous ajoutez le rôle au groupe Greengrass.

> **Note**
>
> Un groupe Greengrass possède un rôle de groupe. Si vous souhaitez ajouter des autorisations, vous pouvez modifier les stratégies attachées ou joindre d'autres stratégies.

Dans ce didacticiel, vous créez une stratégie d'autorisations qui permet de décrire, créer et mettre à jour des actions sur une table Amazon DynamoDB. Ensuite, vous attachez la stratégie à un nouveau rôle et associez le rôle à votre groupe Greengrass.

Tout d'abord, créez une stratégie gérée par le client qui accorde les autorisations requises par la fonction Lambda dans ce module.

1. Dans le panneau de navigation de la console IAM, sélectionnezStratégies, puisCréer une stratégie.
2. Sur l'onglet JSON, remplacez le contenu de l'espace réservé par la stratégie suivante. La fonction Lambda de ce module utilise ces autorisations pour créer et mettre à jour une table DynamoDB nomméeCarStats.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "PermissionsForModule6",
            "Effect": "Allow",
            "Action": [
                "dynamodb:DescribeTable",
                "dynamodb:CreateTable",
                "dynamodb:PutItem"
            ],
            "Resource": "arn:aws:dynamodb:*:*:table/CarStats"
        }
    ]
}
```

3. Choisissez Next (Suivant) Étiquettes, puisSuivant: Review (Examiner). Les balises ne sont pas utilisés dans le cadre de ce didacticiel.
4. Pour Name (Nom), saisissez **`greengrass_CarStats_Table`**, puis choisissez Create policy (Créer une stratégie).

Ensuite, créez un rôle qui utilise la nouvelle stratégie.

5. Dans le volet de navigation, sélectionnez Rôles, puis Créer un rôle.
6. UNDERType d'entité de confiance, choisissezAWSservice.
7. UNDERCas d'utilisation,Cas d'utilisation de d'autresAWSserviceschoisirGreengrass, SelectGreengrass, puisSuivant.
8. UNDERStratégies d'autorisations, sélectionnez le nouveau**`greengrass_CarStats_Table`**policy (stratégie), puisSuivant.

9. Pour le Nom du rôle, saisissez **`Greengrass_Group_Role`**.
10. Pour Description, saisissez **`Greengrass group role for connectors and user-defined Lambda functions`**.
11. Sélectionnez Create role (Créer un rôle).

    Maintenant, ajoutez le rôle à votre groupe Greengrass.

12. DansAWS IoTVolet de navigation de la console sousGérer, DéveloppezAppareils Greengrass, puisGroups (V1).
13. UNDERGroups Greengrass, choisissez votre groupe.
14. ChoisissezParamètres, puisRôle d'associé.
15. ChoisissezGreengrass_Group_Roledans votre liste de rôles, puisRôle d'associé.

Création et configuration de la fonction Lambda

Dans cette étape, vous allez créer une fonction Lambda qui suit le nombre de voitures qui passent le feu de signalisation. Chaque fois que l'état de l'`GG_TrafficLight`ombre passe à`G`, la fonction Lambda simule le passage d'un nombre aléatoire de voitures (de 1 à 20). À chaque troisième changement de`G` lumière, la fonction Lambda envoie des statistiques de base, telles que min et max, à une table DynamoDB.

1. Sur votre ordinateur, créez un dossier nommé `car_aggregator`.
2. À partir du dossier d'TrafficLight exemples GitHub, téléchargez le`carAggregator.py` fichier`car_aggregator` dans le dossier. Il s'agit du code de votre fonction Lambda.

   > **Note**
   >
   > Cet exemple de fichier Python est stocké dans le référentielAWS IoT Greengrass Core SDK pour des raisons de commodité, mais il n'utilise pas le SDKAWS IoT Greengrass Core.

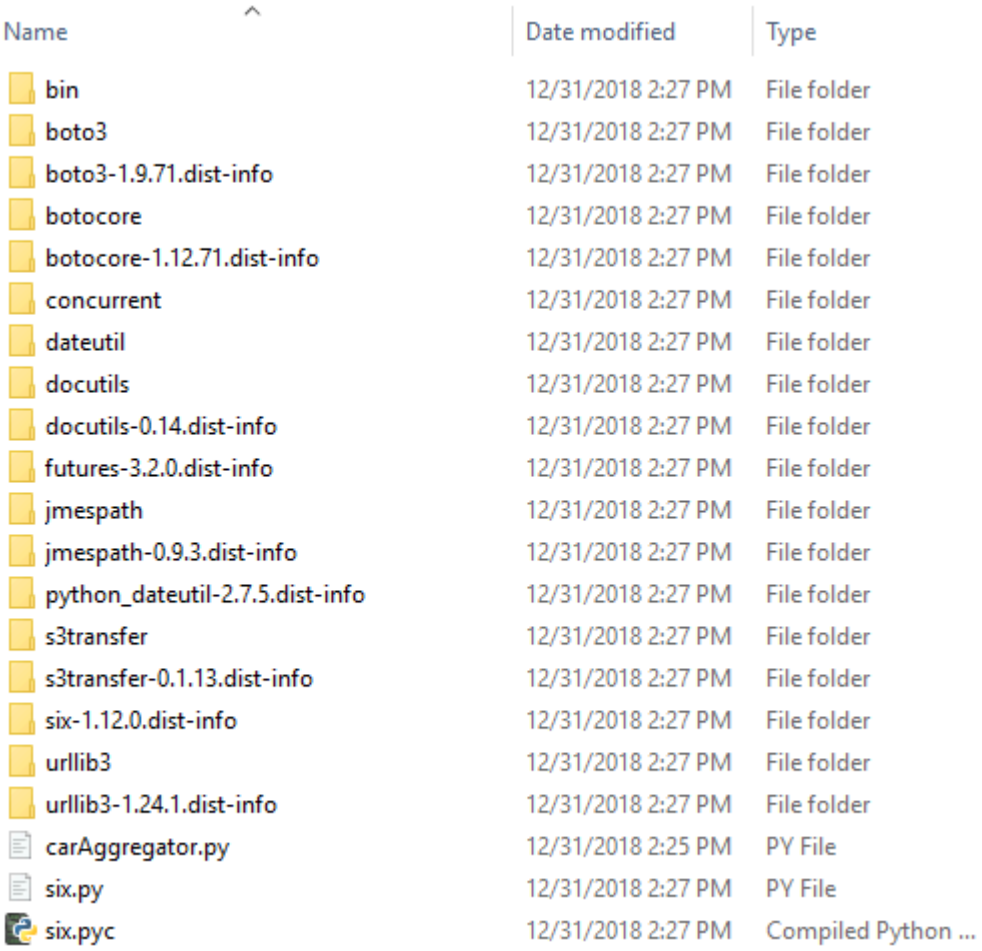
3. Si vous ne travaillez pas dans la région Est des États-Unis (Virginie du Nord), ouvrez la ligne suivante`carAggregator.py` et`region_name` remplacez-la parRégion AWS celle actuellement sélectionnée dans laAWS IoT console. Pour obtenir la liste des versions prises AWS IoT Greengrassen chargeRégion AWS, reportez-vous au Référence générale d'Amazon Web Services.

```
dynamodb = boto3.resource('dynamodb', region_name='us-east-1')
```

4. Exécutez la commande suivante dans une fenêtre de ligne de commande pour installer le AWS SDK for Python (Boto3)package et ses dépendances dans le`car_aggregator` dossier. Les fonctions Lambda de Greengrass utilisent leAWS SDK pour accéder à d'autresAWS services. (Pour Windows, utilisez une invite de commande élevée.)

   ```
   pip install boto3 -t path-to-car_aggregator-folder
   ```

   Vous obtenez une liste de répertoires similaire à la suivante :

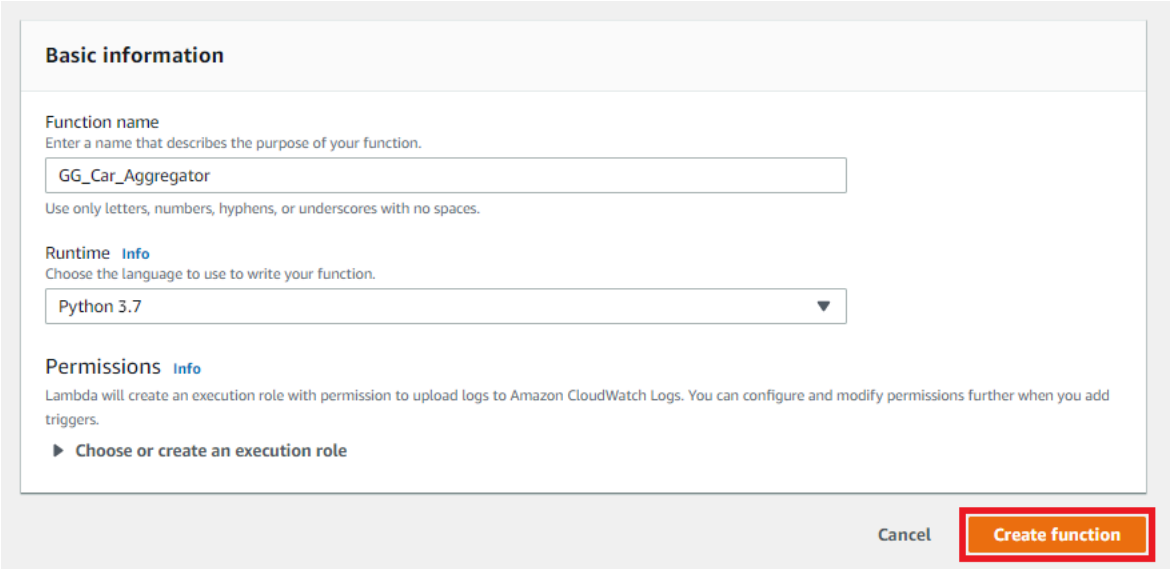
   

5. Compressez le contenu du dossier `car_aggregator` en un fichier `.zip` nommé `car_aggregator.zip`. (Compressez les contenus du dossier, et non pas le dossier.) Il s'agit du package de déploiement de votre fonction Lambda.

6. Dans la console Lambda, créez une fonction nomméeGG_Car_Aggregator et définissez les champs restants comme suit :

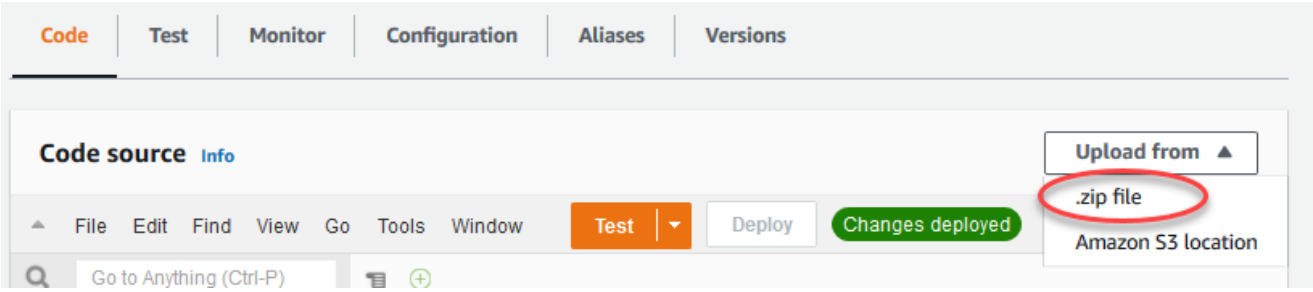
- Pour Runtime, sélectionnez Python 3.7.
- Pour les autorisations, conservez le paramètre par défaut. Cela crée un rôle d'exécution qui accorde des autorisations Lambda de base. Ce rôle n'est pas utilisé parAWS IoT Greengrass.

Sélectionnez Create function (Créer une fonction).



7. Chargez le package de déploiement de votre fonction Lambda :

   a. Dans l'onglet Code, sous Source du code, choisissez Charger depuis. Dans la liste déroulante, choisissez le fichier .zip.



   b. Choisissez charger, puis choisissez votre package`car_aggregator.zip` de déploiement. Ensuite, choisissez Enregistrer.

   c. Dans l'onglet Code de la fonction, sous Paramètres d'exécution, choisissez Modifier, puis entrez les valeurs suivantes.

      - Pour Runtime, sélectionnez Python 3.7.
      - Pour Handler (Gestionnaire), entrez **`carAggregator.function_handler`**.

d. Choisissez Save (Enregistrer).

8. Publiez la fonction Lambda, puis créez un alias nommé**`GG_CarAggregator`**. Pour step-by-step obtenir des instructions, consultez les étapes de publication de la fonction Lambda et de création d'un alias dans le module 3 (partie 1).
9. Dans laAWS IoT console, ajoutez la fonction Lambda que vous venez de créer à votreAWS IoT Greengrass groupe :

   a. Sur la page de configuration du groupe, choisissez Fonctions Lambda, puis sous Mes fonctions Lambda, choisissez Ajouter.

   b. Pour la fonction Lambda, choisissez GG_Car_Aggregator.

   c. Pour la version de la fonction Lambda, choisissez l'alias de la version que vous avez publiée.

   d. Pour Limite de mémoire, entrez **`64 MB`**.

   e. Pour Pinned, choisissez True.

   f. Choisissez Ajouter une fonction Lambda.

> **Note**
>
> Vous pouvez supprimer d'autres fonctions Lambda des modules précédents.

Configuration des abonnements

Dans cette étape, vous créez un abonnement qui permet au GG_TrafficLight shadow pour envoyer les informations sur l'état mises à jour à la fonction Lambda GG_Car_Aggregator. Cet abonnement est ajouté aux abonnements que vous avez créés dans le Module 5, et qui sont tous nécessaires pour ce module.

1. Sur la page de configuration de groupe, choisissez leSubscriptionsonglet, puisAddition.
2. Dans la pageCréer un abonnement, procédez comme suit :

   a. PourType de source, choisissezServicepuisService Shadow local.

   b. PourTarget type (Type de cible), choisissezFonction LambdapuisGG_Car_Aggregator.

   c. Pour Filtre de rubriques, tapez **`$aws/things/GG_TrafficLight/shadow/update/documents`**.

d. Choisissez Create subscription (Créer un abonnement).

Pour ce module, vous avez besoin du nouvel abonnement et des abonnements que vous avez créés dans le Module 5.

3. Assurez-vous que le démon Greengrass est en cours d'exécution comme décrit dansDéploiement des configurations cloud sur un appareil Core.
4. Sur la page de configuration de groupe, choisissezDéploiement.

Test des communications

1. Sur votre ordinateur, ouvrez deux fenêtres de ligne de commande. Comme dansModule 5, une fenêtre correspond à l'appareil client GG_Switch et l'autre à l'appareil de client GG_TrafficLight appareil client. Vous les utilisez pour exécuter les mêmes commandes que celles exécutées dans le Module 5.

   Exécutez les commandes suivantes pour l'appareil 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
   ```

   Exécutez les commandes suivantes pour l'appareil GG_TrafficLight Appareil client :

   ```
   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
   ```

   Toutes les 20 secondes, le commutateur met à jour l'état du shadow sur G, J et R, et le feu affiche son nouvel état.
2. À chaque troisième feu vert (toutes les 3 minutes), le gestionnaire de fonctions de la fonction Lambda est déclenché et un nouvel enregistrement DynamoDB est créé. Après`lightController.py`et`trafficLight.py`ont été exécutés pendant trois minutes, accédez àAWS Management Console, puis ouvrez la console DynamoDB.

3. ChoisissezUSA Est (Virginie du Nord)dans leRégion AWSmenu. Il s'agit de la région où la fonction `GG_Car_Aggregator` crée la table.
4. Dans le panneau de navigation, choisissezTables, puisCarStatstable.
5. ChoisissezView (Affichage)pour afficher les entrées du tableau.

   Vous devez voir les entrées avec les statistiques élémentaires sur les voitures passées (une entrée toutes les trois minutes). Vous devrez peut-être choisir le bouton d'actualisation pour consulter les mises à jour de la table.
6. Si le test échoue, vous pouvez rechercher des informations de dépannage dans les journaux Greengrass.

   a. Connectez-vous en tant qu'utilisateur racine et accédez au répertoire `log`. L'accès aux journaux AWS IoT Greengrass nécessite les autorisations racines.

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

   b. Recherchez les erreurs dans `runtime.log`.

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

   c. Consultez le journal généré par la fonction Lambda.

   ```
   cat user/region/account-id/GG_Car_Aggregator.log
   ```

   Les scripts `lightController.py` et `trafficLight.py` stockent les informations de connexion dans le dossier `groupCA`, qui est créé dans le même dossier que les scripts. Si vous recevez des erreurs de connexion, assurez-vous que l'adresse IP indiquée dans la fenêtre`ggc-host`correspond au point de terminaison de l'adresse IP de votre cœur.

   Pour plus d'informations, consultez Résolution des problèmes.

Il s'agit de la fin du didacticiel de base. Vous devez maintenant comprendreAWS IoT Greengrassmodèle de programmation et ses concepts fondamentaux, y comprisAWS IoT Greengrasscœurs, groupes, abonnements, appareils de client et processus de déploiement des fonctions Lambda s'exécutant à la périphérie.

Vous pouvez supprimer la table DynamoDB, ainsi que les fonctions et abonnements Greengrass Lambda. Pour arrêter les communications entreAWS IoT Greengrasset le périphérique principalAWS IoTcloud, ouvrez une fenêtre de terminal sur l'appareil principal et exécutez l'une des commandes suivantes :

- Pour arrêterAWS IoT GreengrassAppareil Core :

```
sudo halt
```

- Pour arrêter le démon AWS IoT Greengrass :

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

Module 7 : Simulation d'intégration de sécurité matérielle

Cette fonction est disponible pourAWS IoT GreengrassCore v1.7 et versions ultérieures.

Ce module avancé montre comment configurer un module de sécurité matérielle (HSM) simulé à utiliser avec un noyau Greengrass. La configuration utilise SoftHSM, qui est une implémentation logicielle pure utilisant l'API PKCS#11. L'objectif de ce module est de vous permettre de configurer un environnement où vous pouvez apprendre et effectuer des tests initiaux sur une implémentation uniquement logicielle de l'API PKCS#11. Il est fourni uniquement à des fins d'apprentissage et de tests initiaux, et non à des fins de production de quelque nature que ce soit.

Vous pouvez utiliser cette configuration pour tester l'utilisation d'un service compatible avec PKCS # 11 pour stocker vos clés privées. Pour plus d'informations sur l'implémentation uniquement logicielle, consultez SoftHSM. Pour plus d'informations sur l'intégration d'une sécurité matérielle sur unAWS IoT Greengrasscore, y compris les exigences générales, voirthe section called “Intégration de sécurité matérielle”.

> **Important**
>
> Ce module est destiné à des fins de test uniquement. Nous déconseillons vivement l'utilisation de SoftHSM dans un environnement de production, car elle peut donner un faux sentiment de sécurité supplémentaire. La configuration résultante ne fournit pas tous les avantages de sécurité réels. Les clés stockées dans SoftHSM ne sont pas stockées de

façon plus sécurisée que dans n'importe quel autre moyen de stockage de secrets dans l'environnement Greengrass.
L'objectif de ce module est de vous permettre de découvrir la spécification PKCS#11 et d'effectuer des tests initiaux sur votre logiciel si vous prévoyez d'utiliser un HSM matériel à l'avenir.
Vous devez tester séparément et complètement votre implémentation matérielle future avant toute utilisation en production, car il peut y avoir des différences entre l'implémentation PKCS#11 fournie dans SoftHSM par rapport à une implémentation basée sur le matériel.

Si vous avez besoin d'aide pour intégrer unmodule de sécurité matérielle prise en charge, contactez votreAWSReprésentant du support technique.

Avant de commencer, exécutez le script de configuration de périphérique Greengrass ou assurez-vous d'avoir effectué le module 1 et le module 2 du didacticiel de mise en route. Dans ce module, nous supposons que votre noyau est configuré et communique déjà avecAWS. Ce module dure environ 30 minutes.

Installer le logiciel SoftHSM

Dans cette étape, vous installez SoftHSM et les outils pkcs11, qui sont utilisés pour gérer votre instance SoftHSM.

- Dans un terminal sur votreAWS IoT Greengrass, exécutez la commande suivante :

  ```
  sudo apt-get install softhsm2 libsofthsm2-dev pkcs11-dump
  ```

  Pour plus d'informations sur ces packages, consultez Install softhsm2, Install libsofthsm2-dev et Install pkcs11-dump.

  > **Note**
  >
  > Si vous rencontrez des problèmes lors de l'utilisation de cette commande sur votre système, consultez SoftHSM version 2 sur GitHub. Ce site fournit plus d'informations d'installation, notamment sur la façon de générer depuis la source.

Configurer SoftHSM

Au cours de cette étape, vous configurez SoftHSM.

1. Basculez vers l'utilisateur racine.

```
sudo su
```

2. Utilisez la page du manuel pour trouver l'emplacement `softhsm2.conf` dans le système. `/etc/softhsm/softhsm2.conf` est un emplacement courant. Mais l'emplacement peut être différent sur certains systèmes.

```
man softhsm2.conf
```

3. Créez le répertoire pour le fichier de configuration softhsm2 dans l'emplacement sur le système. Dans cet exemple, nous supposons que l'emplacement est `/etc/softhsm/softhsm2.conf`.

```
mkdir -p /etc/softhsm
```

4. Créez le répertoire du jeton dans le répertoire `/greengrass`.

> **Note**
>
> Si vous ignorez cette étape, softhsm2-util indique `ERROR: Could not initialize the library`.

```
mkdir -p /greengrass/softhsm2/tokens
```

5. Configurez le répertoire du jeton.

```
echo "directories.tokendir = /greengrass/softhsm2/tokens" > /etc/softhsm/
softhsm2.conf
```

6. Configurez un backend basé sur des fichiers.

```
echo "objectstore.backend = file" >> /etc/softhsm/softhsm2.conf
```

> **Note**
>
> Ces paramètres de configuration sont proposés à des fins d'expérimentation uniquement. Pour connaître toutes les options de configuration, consultez la page de manuel du fichier de configuration.
>
> ```
> man softhsm2.conf
> ```

Importer la clé privée dans SoftHSM

Au cours de cette étape, vous allez initialiser le jeton SoftHSM, convertir le format de clé privée, puis importer la clé privée.

1. Initialiser le jeton SoftHSM.

   ```
   softhsm2-util --init-token --slot 0 --label greengrass --so-pin 12345 --pin 1234
   ```

   > **Note**
   >
   > Si vous y êtes invité, saisissez le PIN superviseur `12345` et le PIN utilisateur `1234`. AWS IoT Greengrass n'utilise pas le PIN superviseur. Vous pouvez donc utiliser n'importe quelle valeur.
   > Si vous recevez l'erreur `CKR_SLOT_ID_INVALID: Slot 0 does not exist`, essayez plutôt la commande suivante :
   >
   > ```
   > softhsm2-util --init-token --free --label greengrass --so-pin 12345 --pin
   >  1234
   > ```

2. Convertir la clé privée dans un format qui peut être utilisé par l'outil d'importation SoftHSM. Dans le cadre de ce didacticiel, vous devez convertir la clé privée que vous avez obtenue à partir de l'option Création de groupes par défaut dans le Module 2 du didacticiel Démarrez.

   ```
   openssl pkcs8 -topk8 -inform PEM -outform PEM -nocrypt -in hash.private.key -
   out hash.private.pem
   ```

3. Importer la clé privée dans SoftHSM. Parmi les commandes suivantes, exécutez celle qui correspond à votre version de softhsm2-util.

   Syntaxe Raspbian softhsm2-util v2.2.0

   ```
   softhsm2-util --import hash.private.pem --token greengrass --label iotkey --id
    0000 --pin 12340
   ```

   Syntaxe Ubuntu softhsm2-util v2.0.0

   ```
   softhsm2-util --import hash.private.pem --slot 0 --label iotkey --id 0000 --pin
    1234
   ```

   Cette commande identifie l'emplacement comme `0` et définit l'étiquette de clé comme `iotkey`. Vous utiliserez ces valeurs dans la section suivante.

Une fois la clé privée importée, vous pouvez la supprimer (optionnel) du répertoire `/greengrass/certs`. Assurez-vous de conserver la CA racine et les certificats de l'appareil dans le répertoire.

Configurer le noyau Greengrass pour utiliser SoftHSM

Au cours de cette étape, vous allez modifier le fichier de configuration du noyau Greengrass pour utiliser SoftHSM.

1. Trouvez le chemin d'accès à la bibliothèque du fournisseur SoftHSM (`libsofthsm2.so`) sur votre système :

   a. Obtenez la liste des packages installés de la bibliothèque.

      ```
      sudo dpkg -L libsofthsm2
      ```

      Le fichier `libsofthsm2.so` est situé dans le répertoire `softhsm`.

   b. Copiez le chemin complet dans le fichier, par exemple, `/usr/lib/x86_64-linux-gnu/softhsm/libsofthsm2.so`. Vous utiliserez cette valeur plus tard.

2. Arrêtez le démon Greengrass.

   ```
   cd /greengrass/ggc/core/
   ```

```
sudo ./greengrassd stop
```

3. Ouvrez le fichier de configuration Greengrass. Il s'agit du fichier `config.json` dans le répertoire `/greengrass/config`.

> **Note**
>
> Les exemples de cette procédure sont rédigées en supposant que le fichier `config.json` utilise le format qui est généré à partir de l'option Création de groupes par défaut dans le Module 2 du didacticiel Démarrez.

4. Dans l'objet `crypto.principals`, insérez l'objet de certificat de serveur MQTT suivant. Ajoutez une virgule si nécessaire pour créer un fichier JSON.

```
  "MQTTServerCertificate": {
    "privateKeyPath": "path-to-private-key"
  }
```

5. Dans l'objet `crypto`, insérez l'objet PKCS11 suivant. Ajoutez une virgule si nécessaire pour créer un fichier JSON.

```
  "PKCS11": {
    "P11Provider": "/path-to-pkcs11-provider-so",
    "slotLabel": "crypto-token-name",
    "slotUserPin": "crypto-token-user-pin"
  }
```

Votre fichier doit se présenter comme suit :

```
{
  "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**
>
> Pour utiliser over-the-air (OTA) avec sécurité matérielle,`PKCS11`doit également contenir l'objet`OpenSSLEngine`propriété. Pour plus d'informations, consultez the section called “Configurer les mises à jour OTA”.

6. Modifiez l'objet `crypto` :

   a. Configurez l'objet `PKCS11`.

      - Pour `P11Provider`, entrez le chemin d'accès complet à `libsofthsm2.so`.
      - Pour `slotLabel`, saisissez `greengrass`.
      - Pour `slotUserPin`, saisissez `1234`.

   b. Configurez les chemins de la clé privée dans l'objet `principals`. Ne modifiez pas la propriété `certificatePath`.

- Pour les propriétés `privateKeyPath`, saisissez le chemin RFC 7512 PKCS # 11 suivant (qui spécifie l'étiquette de clé). Faites de même pour le `IoTCertificate`, le `SecretsManager` et les mandataires `MQTTServerCertificate`.

```
pkcs11:object=iotkey;type=private
```

c. Vérifiez l'objet `crypto`. Il doit ressembler à l'exemple ci-dessous.

```
"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. Supprimez les valeurs `caPath`, `certPath` et `keyPath` de l'objet `coreThing`. Il doit ressembler à l'exemple ci-dessous.

```
"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**
>
> Dans le cadre de ce didacticiel, vous spécifiez la même clé privée pour tous les mandataires. Pour plus d'informations sur le choix de la clé privée du serveur MQTT local, consultez Performance. Pour plus d'informations sur le secrets manager local, consultez Déployer des secrets sur Core.

Tester la configuration

- Démarrez le démon Greengrass.

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

  Si le démon démarre correctement, cela indique que votre noyau est configuré correctement.

  Vous êtes maintenant prêt à en savoir plus sur la spécification PKCS#11 et à effectuer les tests initiaux avec l'API PKCS#11 qui est fournie par l'implémentation SoftHSM.

  > **Important**
  >
  > Là encore, il est extrêmement important d'avoir conscience que ce module est conçu pour l'apprentissage et le test uniquement. Il ne renforce pas réellement le niveau de sécurité de votre environnement Greengrass.
  > Au lieu de cela, le module a pour objectif de vous permettre de commencer à apprendre et à tester en vue d'une utilisation future d'un véritable HSM basé sur le matériel. À ce moment-là, vous devrez tester séparément et complètement votre logiciel sur le module HSM basé sur le matériel avant toute utilisation en production, car il peut y avoir des différences entre l'implémentation PKCS # 11 fournie dans SoftHSM et une implémentation basée sur le matériel.

Consulter aussi

- PKCS #11 Cryptographic Token Interface Usage Guide Version 2.40. Révisé par John Leiseboer et Robert Griffin. 16 novembre 2014. OASIS Committee Remarque 02. http://docs.oasis-open.org/

pkcs11/pkcs11-ug/v2.40/cn02/pkcs11-ug-v2.40-cn02.html. Dernière version : http://docs.oasis-open.org/pkcs11/pkcs11-ug/v2.40/pkcs11-ug-v2.40.html.

- RFC 7512

Mises à jour OTA du logiciel AWS IoT Greengrass Core

Le package logiciel AWS IoT Greengrass Core inclut un agent de mise à jour qui peut effectuer des mises à jour over-the-air (OTA) du AWS IoT Greengrass logiciel. Vous pouvez utiliser les mises à jour OTA pour installer la dernière version du logiciel AWS IoT Greengrass Core ou du logiciel de l'agent de mise à jour OTA sur un ou plusieurs cœurs. Avec les mises à jour OTA, vos appareils principaux n'ont pas besoin d'être physiquement présents.

Nous vous recommandons d'utiliser les mises à jour OTA dès que possible. Elles fournissent un mécanisme que vous pouvez utiliser pour suivre l'état et l'historique des mises à jour. En cas d'échec de la mise à jour, l'agent de mise à jour OTA revient à la version logicielle précédente.

> **Note**
>
> Les mises à jour OTA ne sont pas prises en charge lorsque vous utilisez `apt` pour installer le logiciel AWS IoT Greengrass Core. Pour ces installations, nous vous recommandons d'utiliser `apt` pour mettre à niveau le logiciel. Pour plus d'informations, veuillez consulter the section called “Installer à partir d'un référentiel APT”.

Les mises à jour OTA permettent de mieux :

- Corriger les vulnérabilités de sécurité.
- Prendre en charge des problèmes de stabilité logicielle.
- Déployer de nouvelles fonctionnalités ou des fonctionnalités améliorées.

Cette fonction s'intègre aux tâches AWS IoT.

Prérequis

Les conditions suivantes s'appliquent aux mises à jour OTA du logiciel AWS IoT Greengrass.

- Le cœur de Greengrass doit disposer d'au moins 400 Mo d'espace disque disponible dans le stockage local. L'agent de mise à jour OTA nécessite environ trois fois plus d'espace d'exécution que le logiciel AWS IoT Greengrass Core. Pour de plus amples informations, veuillez consulter Quotas de service pour le noyau Greengrass dans le Référence générale d'Amazon Web Services.
- Le noyau Greengrass doit être connecté auAWS Cloud.

- Le noyau Greengrass doit être correctement configuré et provisionné avec des certificats et des clés pour l'authentification avec AWS IoT Core et AWS IoT Greengrass. Pour plus d'informations, veuillez consulter the section called “Certificats X.509”.
- Le noyau Greengrass ne peut pas être configuré pour utiliser un proxy réseau.

> **Note**
>
> À partir de la version AWS IoT Greengrass 1.9.3, les mises à jour OTA sont prises en charge sur les noyaux qui configurent le trafic MQTT sur le port 443 au lieu du port par défaut 8883. Toutefois, l'agent de mise à jour OTA ne prend pas en charge les mises à jour via un proxy réseau. Pour plus d'informations, veuillez consulter the section called “Connexion au port 443 ou via un proxy réseau”.

- Le démarrage de confiance ne peut pas être activé dans la partition qui contient le logiciel AWS IoT Greengrass Core.

> **Note**
>
> Vous pouvez installer et exécuter le logiciel AWS IoT Greengrass Core sur une partition dont le démarrage de confiance est activé, mais les mises à jour OTA ne sont pas prises en charge.

- AWS IoT Greengrass doit avoir des autorisations de lecture/écriture sur la partition qui contient le logiciel AWS IoT Greengrass Core.
- Si vous utilisez un système d'initialisation pour gérer votre noyau Greengrass, vous devez configurer les mises à jour OTA pour qu'elles s'intègrent au système d'initialisation. Pour plus d'informations, veuillez consulter the section called “Intégration à des systèmes d'initialisation”.
- Vous devez créer un rôle utilisé pour présigner les URL Amazon S3 vers les artefacts de mise à jour AWS IoT Greengrass logicielle. Ce rôle de signataire vous permet d'accéder AWS IoT Core aux artefacts de mise à jour logicielle stockés dans Amazon S3 en votre nom. Pour plus d'informations, veuillez consulter the section called “Autorisations IAM pour les mises à jour OTA”.

Autorisations IAM pour les mises à jour OTA

Lors AWS IoT Greengrass de la publication d'une nouvelle version du logiciel AWS IoT Greengrass Core, AWS IoT Greengrass met à jour les artefacts logiciels stockés dans Amazon S3 qui sont utilisés pour la mise à jour OTA.

Vous Compte AWS devez inclure un rôle de signataire d'URL Amazon S3 qui peut être utilisé pour accéder à ces artefacts. Le rôle doit disposer d'une politique d'autorisations qui autorise l'`s3:GetObject`action sur les compartiments des ciblesRégion AWS. Le rôle doit également avoir une stratégie d'approbation qui permet à `iot.amazonaws.com` d'assumer le rôle en tant qu'entité approuvée.

Politique d'autorisations

Pour les autorisations de rôle, vous pouvez utiliser la stratégie gérée par AWS ou créer une stratégie personnalisée.

- Utiliser la stratégie gérée par AWS

  La politique UpdateArtifactAccess gérée de GreenGrassota est fournie par. AWS IoT Greengrass Utilisez cette politique si vous souhaitez autoriser l'accès à toutes les régions Amazon Web Services prises en charge parAWS IoT Greengrass, actuelles et futures.

- Création d'une politique personnalisée

  Vous devez créer une politique personnalisée si vous souhaitez spécifier explicitement les régions Amazon Web Services dans lesquelles vos cœurs sont déployés. L'exemple de stratégie suivant autorise l'accès aux mises à jour du logiciel AWS IoT Greengrass dans six régions.

```
{
    "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/*"
            ]
        }
    ]
```

```
}
```

Politique d'approbation

La stratégie d'approbation attachée au rôle doit autoriser l'action `sts:AssumeRole` et définir `iot.amazonaws.com` comme élément principal. Cela autorise AWS IoT Core à assumer le rôle en tant qu'entité de confiance. Voici un exemple de document de stratégie :

```
{
     "Version": "2012-10-17",
     "Statement": [
         {
             "Sid": "AllowIotToAssumeRole",
             "Action": "sts:AssumeRole",
             "Principal": {
                  "Service": "iot.amazonaws.com"
             },
             "Effect": "Allow"
        }
    ]
}
```

En outre, l'utilisateur qui initie une mise à jour OTA doit être autorisé à utiliser `greengrass:CreateSoftwareUpdateJob` et `iot:CreateJob`, et à utiliser `iam:PassRole` pour transmettre les autorisations du rôle de signataire. Voici un exemple de politique 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"
        }
    ]
}
```

Considérations

Avant de lancer une mise à jour OTA du logiciel Greengrass Core, vous devez connaître l'impact sur les appareils de votre groupe Greengrass, tant sur l'appareil principal que sur les appareils clients connectés localement à ce noyau :

- Le cœur s'arrête au cours de la mise à jour.
- Toutes les fonctions Lambda s'exécutant sur le noyau seront arrêtées. Si ces fonctions procèdent à des écritures dans des ressources locales, elles peuvent laisser ces ressources dans un état incorrect si elles ne sont pas fermées correctement.
- Pendant les temps d'arrêt du cœur, toutes ses connexions avec le cœur AWS Cloud sont perdues. Les messages qui lui sont envoyés par des appareils clients sont perdus.
- Les caches des informations d'identification sont perdus.
- Les files d'attente qui fonctionnent en attente des fonctions Lambda sont perdues.
- Les fonctions Lambda à longue durée de vie perdent leurs informations d'état dynamique et toutes les tâches en attente sont abandonnées.

Les informations d'état suivantes sont conservées pendant une mise à jour OTA :

- Configuration de Core
- Configuration du groupe Greengrass
- Instances shadows locales
- Journaux Greengrass
- Journaux des agents de mise à jour OTA

Agent de mise à jour OTA Greengrass

L'agent de mise à jour Greengrass OTA est le composant logiciel de l'appareil qui gère les tâches de mise à jour créées et déployées dans le cloud. L'agent de mise à jour OTA est distribué dans le même progiciel que le logiciel AWS IoT Greengrass Core. Cet agent est situé dans */greengrass-root*`/ota/ota_agent/ggc-ota`. Il écrit les journaux dans `/var/log/greengrass/ota/ggc_ota.txt`.

> **Note**
>
> */greengrass-root* indique le chemin d'installation du logiciel AWS IoT Greengrass Core sur votre appareil. Généralement, il s'agit du répertoire `/greengrass`.

Vous pouvez démarrer l'agent de mise à jour OTA en exécutant le fichier binaire manuellement ou en l'intégrant dans un script d'initialisation, tel qu'un fichier de service systemd. Si vous exécutez le binaire manuellement, il doit être exécuté en tant que root. Au démarrage, l'agent de mise à jour OTA écoute les tâches de mise à jour AWS IoT Greengrass logicielle AWS IoT Core et les exécute de manière séquentielle. L'agent de mise à jour OTA ignore tous les autres types de AWS IoT tâches.

L'extrait suivant montre un exemple de fichier de service systemd permettant de démarrer, d'arrêter et de redémarrer l'agent de mise à jour 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 cœur qui est la cible d'une mise à jour ne doit pas exécuter deux instances de l'agent de mise à jour OTA. Cela conduirait les deux agents à traiter les mêmes tâches, ce qui créerait des conflits.

Intégration à des systèmes d'initialisation

Lors d'une mise à jour OTA, l'agent de mise à jour OTA redémarre les fichiers binaires sur le périphérique principal. Si les fichiers binaires sont en cours d'exécution, cela peut entraîner des conflits lorsqu'un système d'initialisation surveille l'état du logiciel AWS IoT Greengrass Core ou l'agent pendant la mise à jour. Pour vous aider à intégrer le mécanisme de mise à jour OTA à vos stratégies de surveillance d'initialisation, vous pouvez écrire des scripts shell qui s'exécutent avant et après une mise à jour. Par exemple, vous pouvez utiliser le `ggc_pre_update.sh` script pour sauvegarder des données ou arrêter des processus avant que l'appareil ne s'arrête.

Pour demander à l'agent de mise à jour OTA d'exécuter ces scripts, vous devez inclure l'`"managedRespawn" : true`indicateur dans le fichier config.json. Ce paramètre est illustré dans l'extrait suivant :

```
{
   "coreThing": {
       …
   },
   "runtime": {
       …
   },
   "managedRespawn": true
       …
}
```

Commande managedRespawn avec mises à jour OTA

Les exigences suivantes s'appliquent aux mises à jour OTA `managedRespawn` définies sur `true` :

- Les scripts shell suivants doivent être présents dans le */greengrass-root*`/usr/scripts` répertoire :
  - `ggc_pre_update.sh`
  - `ggc_post_update.sh`
  - `ota_pre_update.sh`
  - `ota_post_update.sh`
- Les scripts doivent renvoyer un code de retour indiquant la réussite.
- Les scripts doivent être détenus par la racine et être exécutables par la racine uniquement.
- Le `ggc_pre_update.sh` script doit arrêter le démon Greengrass.

- Le `ggc_post_update.sh` script doit démarrer le démon Greengrass.

> **Note**
>
> Comme l'agent de mise à jour OTA gère son propre processus, les `ota_post_update.sh` scripts `ota_pre_update.sh` et n'ont pas besoin d'arrêter ou de démarrer le service OTA.

L'agent de mise à jour OTA exécute les scripts du/*greengrass-root*/usr/scripts. L'arborescence doit être similaire à ce qui suit :

```
<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
```

Lorsque `managedRespawn` ce paramètre est défini sur`true`, l'agent de mise à jour OTA vérifie la présence de ces scripts dans le */greengrass-root*/usr/scripts répertoire avant et après la mise à jour logicielle. Si les scripts n'existent pas, la mise à jour échoue. AWS IoT Greengrassne valide pas le contenu de ces scripts. La meilleure pratique consiste à vérifier que vos scripts fonctionnent correctement et à émettre les codes de sortie appropriés en cas d'erreur.

Pour les mises à jour OTA du logiciel AWS IoT Greengrass Core :

- Avant de démarrer la mise à jour, l'agent exécute le script `ggc_pre_update.sh`. Utilisez ce script pour les commandes qui doivent être exécutées avant que l'agent de mise à jour OTA ne lance la mise à jour logicielle AWS IoT Greengrass Core, par exemple pour sauvegarder des données ou arrêter tout processus en cours d'exécution. L'exemple suivant montre un script simple pour arrêter le démon Greengrass.

  ```
  #!/bin/bash
  set -euo pipefail
  ```

```
systemctl stop greengrass
```

- Après avoir terminé la mise à jour, l'agent exécute le script `ggc_post_update.sh`. Utilisez ce script pour les commandes qui doivent être exécutées après que l'agent de mise à jour OTA a lancé la mise à jour logicielle AWS IoT Greengrass Core, par exemple pour redémarrer les processus. L'exemple suivant montre un script simple pour démarrer le démon Greengrass.

```
#!/bin/bash
set -euo pipefail
systemctl start greengrass
```

Pour les mises à jour OTA de l'agent de mise à jour OTA :

- Avant de démarrer la mise à jour, l'agent exécute le script `ota_pre_update.sh`. Utilisez ce script pour les commandes qui doivent être exécutées avant que l'agent de mise à jour OTA ne se mette à jour, par exemple pour sauvegarder des données ou arrêter tout processus en cours d'exécution.
- Après avoir terminé la mise à jour, l'agent exécute le script `ota_post_update.sh`. Utilisez ce script pour les commandes qui doivent être exécutées après la mise à jour de l'agent de mise à jour OTA, par exemple pour redémarrer des processus.

> **Note**
>
> S'il `managedRespawn` est défini sur`false`, l'agent de mise à jour OTA n'exécute pas les scripts.

Création d'une mise à jour OTA

Procédez comme suit pour effectuer une mise à jour OTA du logiciel AWS IoT Greengrass sur un ou plusieurs noyaux :

1. Assurez-vous que vos noyaux répondent aux conditions requises pour les mises à jour OTA.

> **Note**
>
> Si vous avez configuré un système d'initialisation pour gérer le logiciel AWS IoT Greengrass Core ou l'agent de mise à jour OTA, vérifiez les points suivants sur vos cœurs :
>
> - Le fichier config.json spécifie `"managedRespawn" : true`.
> - Le répertoire/*greengrass-root* /usr/scripts contient les scripts suivants :
>   - ggc_pre_update.sh
>   - ggc_post_update.sh
>   - ota_pre_update.sh
>   - ota_post_update.sh
>
> Pour plus d'informations, veuillez consulter the section called “Intégration à des systèmes d'initialisation”.

2. Dans le terminal d'un appareil principal, démarrez l'agent de mise à jour OTA.

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

> **Note**
>
> *greengrass-root* indique le chemin d'installation du logiciel AWS IoT Greengrass Core sur votre appareil. Généralement, il s'agit du répertoire `/greengrass`.

Ne démarrez pas plusieurs instances de l'agent de mise à jour OTA sur un noyau car cela pourrait provoquer des conflits.

3. Utilisez l'AWS IoT GreengrassAPI pour créer une tâche de mise à jour logicielle.

   a. Appelez l'API CreateSoftwareUpdateJob. Dans cet exemple de procédure, nous utilisons les commandes de l'AWS CLI.

      La commande suivante crée une tâche qui met à jour le logiciel AWS IoT Greengrass Core sur un seul noyau. Remplacez les exemples de valeur, puis exécutez la commande.

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
```

Cette commande renvoie la réponse suivante.

```
{
    "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. Copiez l'élément `IoTJobId` de la réponse.

c. Appelez DescribeJobl'AWS IoT CoreAPI pour voir l'état de la tâche. Remplacez les exemples de valeur par l'ID de votre tâche, puis exécutez la commande.

```
aws iot describe-job --job-id GreengrassUpdateJob_c3bd7f36-ee80-4d42-8321-
a1da0EXAMPLE
```

La commande renvoie un objet de réponse qui contient des informations sur la tâche, y compris `status` et `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": {}
    }
}
```

Pour bénéficier d'une aide à la résolution des problèmes, consultez Résolution des problèmes.

API CreateSoftwareUpdateJob

Vous pouvez utiliser l'`CreateSoftwareUpdateJob`API pour mettre à jour le logiciel AWS IoT Greengrass Core ou le logiciel de l'agent de mise à jour OTA sur vos appareils principaux. Cette API crée une tâche d'instantané AWS IoT qui avertit les périphériques lorsqu'une mise à jour est disponible. Après avoir appelé `CreateSoftwareUpdateJob`, vous pouvez utiliser d'autres commandes de tâche AWS IoT pour suivre la mise à jour logicielle. Pour plus d'informations, consultez la section Jobs du Guide du AWS IoT développeur.

L'exemple suivant montre comment utiliser l'AWS CLI pour créer une tâche de mise à jour du logiciel AWS IoT Greengrass Core sur un appareil principal :

```
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
```

La commande `create-software-update-job` renvoie une réponse JSON qui contient l'ID de tâche, l'ARN de la tâche et la version du logiciel qui a été installée par la mise à jour :

```
{
    "IotJobId": "GreengrassUpdateJob_c3bd7f36-ee80-4d42-8321-a1da0EXAMPLE",
    "IotJobArn": "arn:aws:iot:region:123456789012:job/GreengrassUpdateJob_c3bd7f36-
ee80-4d42-8321-a1da0EXAMPLE",
    "PlatformSoftwareVersion": "1.9.2"
}
```

Pour consulter les étapes pour utiliser `create-software-update-job` afin de mettre à jour un appareil principal, veuillez consulter the section called “Création d'une mise à jour OTA”.

La commande `create-software-update-job` est paramétrée comme suit :

`--update-targets-architecture`

Architecture de l'appareil du noyau.

Valeurs valides : `armv7l`, `armv6l`, `x86_64` ou `aarch64`

`--update-targets`

Noyaux à mettre à jour. La liste peut contenir les ARN de noyaux individuels et les ARN de groupes d'objets dont les membres sont des noyaux. Pour plus d'informations sur les groupes d'objets, consultez la section Groupes d'objets statiques dans le Guide du AWS IoT développeur.

`--update-targets-operating-system`

Système d'exploitation de l'appareil du noyau.

Valeurs valides : `ubuntu`, `amazon_linux`, `raspbian` ou `openwrt`

`--software-to-update`

Spécifie si le logiciel principal ou le logiciel de l'agent de mise à jour OTA doit être mis à jour.

Valeurs valides : `core` ou `ota_agent`

`--s3-url-signer-role`

L'ARN du rôle IAM utilisé pour présigner l'URL Amazon S3 qui renvoie aux artefacts de mise à jour AWS IoT Greengrass logicielle. La politique d'autorisations attachée au rôle doit autoriser l'`s3:GetObject`action sur les compartiments des ciblesRégion AWS. Le rôle doit également autoriser `iot.amazonaws.com` à assumer le rôle en tant qu'entité approuvée. Pour plus d'informations, veuillez consulter the section called “Autorisations IAM pour les mises à jour OTA”.

`--amzn-client-token`

(Facultatif) Jeton client utilisé pour effectuer des requêtes idempotentes. Fournissez un jeton unique pour empêcher la création de mises à jour en double à la suite de tentatives internes.

`--update-agent-log-level`

(Facultatif) Le niveau de journalisation des instructions de journal générées par l'agent de mise à jour OTA. La valeur par défaut est `ERROR`.

Valeurs valides : `NONE`, `TRACE`, `DEBUG`, `VERBOSE`, `INFO`, `WARN`, `ERROR` ou `FATAL`

> **Note**
>
> `CreateSoftwareUpdateJob` accepte les demandes uniquement pour les combinaisons d'architecture et de système d'exploitation prises en charge suivantes :
>
> - ubuntu/x86_64

- ubuntu/aarch64
- amazon_linux/x86_64
- raspbian/armv7l
- raspbian/armv6l
- openwrt/aarch64
- openwrt/armv7l

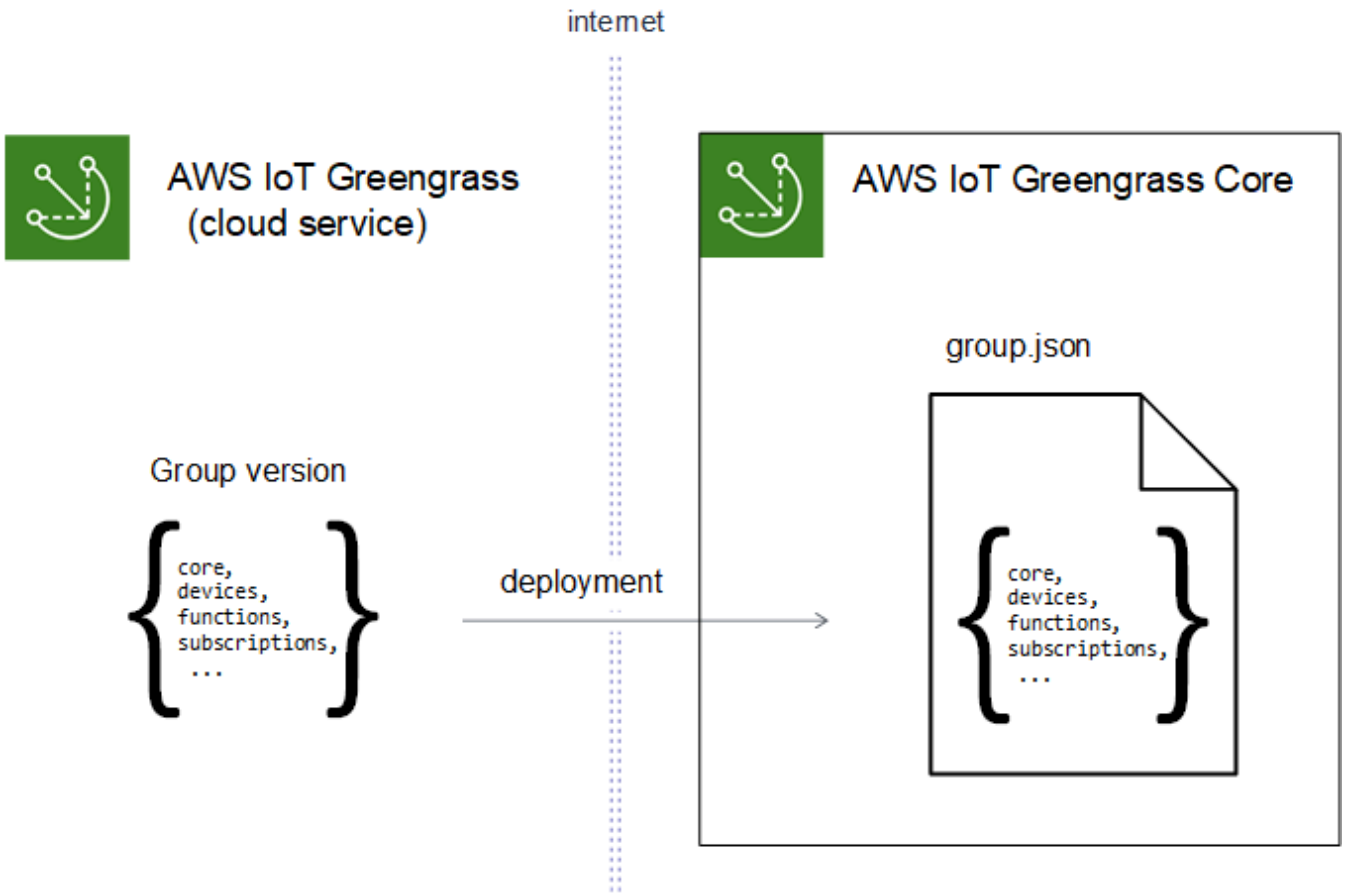
Déploiement de groupes AWS IoT Greengrass sur un noyau AWS IoT Greengrass Core

Utilisez AWS IoT Greengrass des groupes pour organiser les entités dans votre environnement périphérique. Vous pouvez également utiliser des groupes pour contrôler la manière dont les entités du groupe interagissent entre elles et avec leAWS Cloud. Par exemple, seules les fonctions Lambda du groupe sont déployées pour être exécutées localement, et seuls les appareils du groupe peuvent communiquer via le serveur MQTT local.

Un groupe doit inclure un noyau, appareil AWS IoT qui exécute le logiciel AWS IoT Greengrass Core. Le noyau agit comme une passerelle périphérique et fournit des fonctionnalités AWS IoT Core dans l'environnement périphérique. En fonction des besoins de votre entreprise, vous pouvez également ajouter les entités suivantes à un groupe :

- Appareils clients. Représenté en tant qu'objets dans le registre AWS IoT. Ces appareils doivent exécuter FreeRTOS ou utiliser AWS IoTle Device SDK AWS IoT Greengrass ou l'API Discovery pour obtenir des informations de connexion pour le noyau. Seuls les appareils clients membres du groupe peuvent se connecter au cœur.
- Fonctions Lambda. Applications sans serveur définies par l'utilisateur qui exécutent du code sur le cœur. Les fonctions Lambda sont créées AWS Lambda et référencées à partir d'un groupe Greengrass. Pour plus d’informations, consultez Exécuter des fonctions Lambda locales.
- Connecteurs. Applications sans serveur prédéfinies qui exécutent du code sur le cœur. Les connecteurs peuvent fournir une intégration intégrée à l'infrastructure localeAWS, aux protocoles des appareils et à d'autres services cloud. Pour plus d’informations, consultez Intégrer à des services et protocoles à l'aide de connecteurs.
- Abonnements. Définit les éditeurs, les abonnés et les rubriques MQTT (ou sujets) qui sont autorisés pour la communication MQTT.
- Ressources. Références aux appareils et volumes locaux, aux modèles d'apprentissage automatique et aux secrets, utilisés pour le contrôle d'accès par les fonctions et connecteurs Greengrass Lambda.
- Journaux. Configurations de journalisation pour les composants AWS IoT Greengrass du système et les fonctions Lambda. Pour plus d’informations, consultez the section called “Surveillance avec les journaux AWS IoT Greengrass”.

Vous gérez votre groupe Greengrass dans le, AWS Cloud puis vous le déployez sur un noyau. Le déploiement copie la configuration du groupe dans le fichier `group.json` sur l'appareil principal. Ce fichier se trouve dans le dossier *`greengrass-root`*`/ggc/deployments/group`.



> **Note**
>
> Au cours d'un déploiement, le processus de démon Greengrass sur l'appareil principal s'arrête, puis redémarre.

Déploiement de groupes depuis la AWS IoT console

Vous pouvez déployer un groupe et gérer ses déploiements depuis la page de configuration du groupe dans la AWS IoT console.

> **Note**
>
> Pour ouvrir cette page dans la console, choisissez Greengrass devices, Groups (V1), puis sous Greengrass groups, choisissez votre groupe.

Pour déployer la version actuelle du groupe

- Sur la page de configuration du groupe, choisissez Deploy.

Pour afficher l'historique de déploiement du groupe

L'historique de déploiement d'un groupe inclut la date et l'heure, la version du groupe et l'état de chaque tentative de déploiement.

1. Sur la page de configuration du groupe, choisissez l'onglet Déploiements.
2. Pour obtenir plus d'informations sur un déploiement, y compris les messages d'erreur, choisissez Déploiements depuis la AWS IoT console, sous Appareils Greengrass.

Pour redéployer un déploiement de groupe

Vous pouvez souhaiter redéployer un déploiement si le déploiement actuel échoue ou revenir à une autre version de groupe.

1. Sur la AWS IoT console, choisissez Greengrass devices, puis Groups (V1).
2. Choisissez l'onglet Déploiements.
3. Choisissez le déploiement que vous souhaitez redéployer, puis Redéployer.

Pour réinitialiser les déploiements de groupe

Vous pouvez souhaiter réinitialiser les déploiements de groupe pour déplacer ou supprimer un groupe ou supprimer des informations de déploiement. Pour plus d'informations, consultez the section called “Réinitialiser les déploiements”.

1. Sur la AWS IoT console, choisissez Greengrass devices, puis Groups (V1).
2. Choisissez l'onglet Déploiements.

3. Choisissez le déploiement que vous souhaitez réinitialiser, puis sélectionnez Réinitialiser les déploiements.

Déploiement de groupes avec l'API AWS IoT Greengrass

L'API AWS IoT Greengrass fournit les actions suivantes pour déployer des groupes AWS IoT Greengrass et gérer des déploiements de groupe. Vous pouvez appeler ces actions à partir de l'AWS CLI, de l'API AWS IoT Greengrass ou du kit de développement SDK AWS.

Action	Description
CreateDeployment	Crée un déploiement `Redeployment` ou `NewDeployment` . Vous pouvez souhaiter redéployer un déploiement si le déploiement actuel échoue. Ou vous pouvez souhaiter redéployer pour revenir à une autre version de groupe.
GetDeploymentStatus	Renvoie le statut d'un déploiement : `Building`, `InProgress` , `Success` ou `Failure`. Vous pouvez configurer les EventBridge événements Amazon pour recevoir des notifications de déploiement. Pour plus d'informations, consultez the section called "Obtention des notifications de déploiement".
ListDeployments	Renvoie l'historique de déploiement pour le groupe.
ResetDeployments	Réinitialise les déploiements pour le groupe. Vous pouvez souhaiter réinitialiser les déploiements de groupe pour déplacer ou supprimer un groupe ou supprimer des informations de déploiement. Pour plus

Action	Description
	d'informations, consultez the section called "Réinitialiser les déploiements".

> **Note**
>
> Pour plus d'informations sur les opérations de déploiement en bloc, consultez the section called "Créer des déploiements en bloc".

Obtention de l'ID de groupe

L'ID de groupe est couramment utilisé dans les actions API. Vous pouvez utiliser cette ListGroupsaction pour trouver l'identifiant du groupe cible dans votre liste de groupes. Par exemple, dans le AWS CLI, utilisez la commande `list-groups`.

```
aws greengrass list-groups
```

Vous pouvez également inclure l'option `query` pour filtrer les résultats. Par exemple :

- Pour obtenir le groupe créé le plus récemment :

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

- Pour obtenir un groupe par son nom :

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

  Les noms de groupe ne devant pas nécessairement être uniques, plusieurs groupes peuvent être renvoyés.

Voici un exemple de réponse `list-groups` : Les informations relatives à chaque groupe comprennent l'ID du groupe (dans la propriété `Id`) et l'ID de la version de groupe la plus récente (dans la propriété `LatestVersion`). Pour obtenir d'autres identifiants de version pour un groupe, utilisez l'ID de groupe avec ListGroupVersions.

> **Note**
>
> Vous pouvez également trouver ces valeurs dans la AWS IoT console. L'ID du groupe s'affiche sur la page Paramètres du groupe. Les identifiants de version du groupe sont affichés dans l'onglet Déploiements du groupe.

```
{
    "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"
        },
        ...
    ]
}
```

Si vous ne spécifiez pas deRégion AWS, AWS CLI les commandes utilisent la région par défaut de votre profil. Pour renvoyer des groupes dans une autre région, incluez l'option *`region`*. Par exemple :

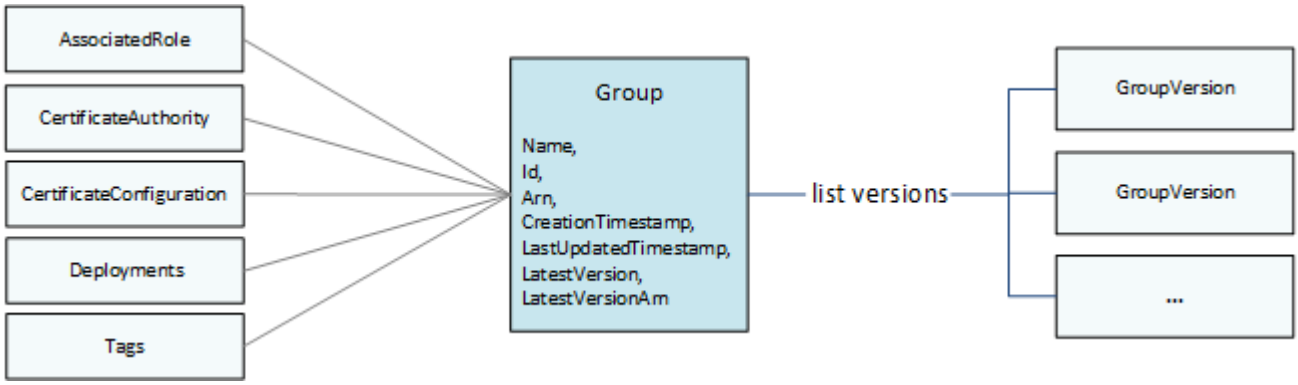
```
aws greengrass list-groups --region us-east-1
```

Présentation du modèle d'objet de groupe AWS IoT Greengrass

Lors de la programmation avec l'API AWS IoT Greengrass, il est utile de comprendre le modèle d'objet de groupe Greengrass.

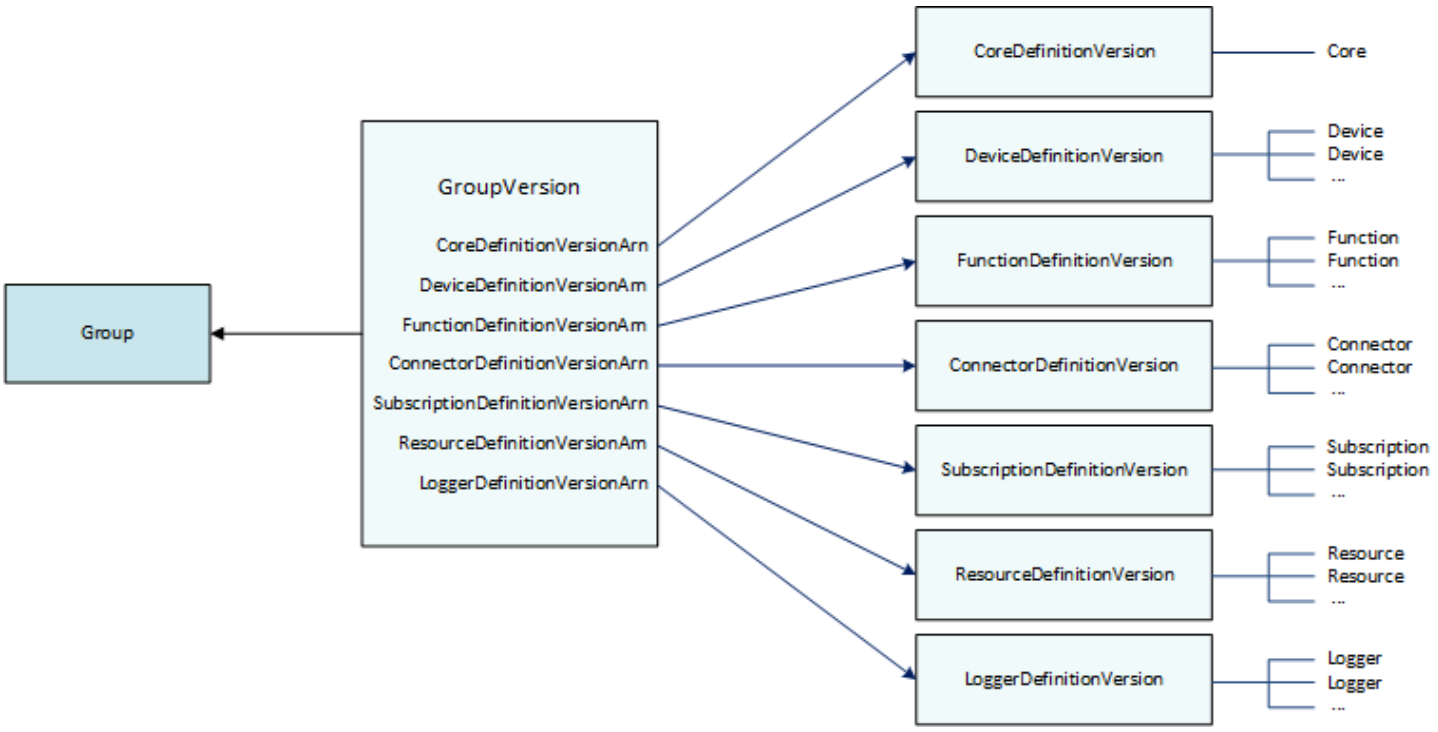
Groups

Dans l'API AWS IoT Greengrass, l'objet de niveau supérieur `Group` se compose de métadonnées et d'une liste d'objets `GroupVersion`. Les objets `GroupVersion` sont associés à un `Group` par ID.



Versions de groupe

Les objets `GroupVersion` définissent l'appartenance au groupe. Chaque objet `GroupVersion` fait référence à un objet `CoreDefinitionVersion` et d'autres versions de composant par ARN. Ces références déterminent les entités à inclure dans le groupe.



Par exemple, pour inclure trois fonctions Lambda, un appareil et deux abonnements dans le groupe, les `GroupVersion` références :

- `CoreDefinitionVersion` contenant le noyau requis.
- `FunctionDefinitionVersion`, qui contient les trois fonctions.
- Celui `DeviceDefinitionVersion` qui contient l'appareil client.
- `SubscriptionDefinitionVersion`, qui contient les deux abonnements.

L'objet `GroupVersion` déployé sur un périphérique noyau détermine les entités disponibles dans l'environnement local et la manière dont elles peuvent interagir.

Composants de groupe

Les composants que vous ajoutez aux groupes ont une hiérarchie à trois niveaux :

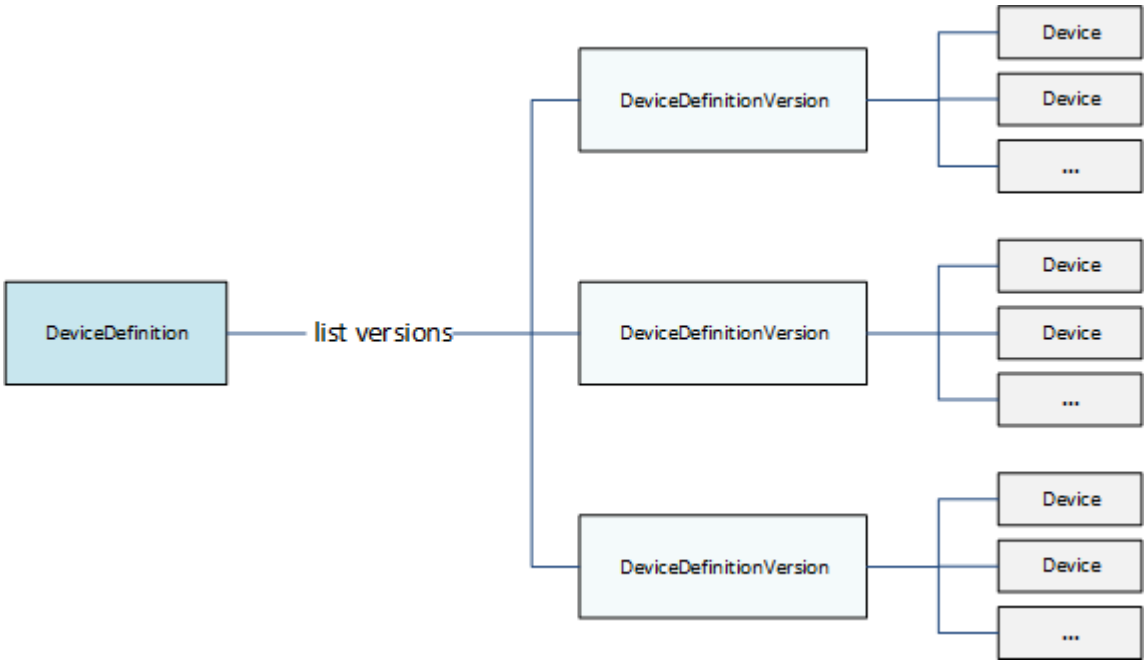
- Définition qui fait référence à une liste d'DefinitionVersionobjets d'un type donné. Par exemple, un objet `DeviceDefinition` fait référence à une liste d'objets `DeviceDefinitionVersion`.
- Un DefinitionVersionqui contient un ensemble d'entités d'un type donné. Par exemple, un objet `DeviceDefinitionVersion` contient une liste d'objets `Device`.

- Entités individuelles qui définissent leurs propriétés et leur comportement. Par exemple, a `Device` définit l'ARN de l'appareil client correspondant dans le AWS IoT registre, l'ARN de son certificat d'appareil et indique si son shadow local se synchronise automatiquement avec le cloud.

  Vous pouvez ajouter les types d'entité suivants à un groupe :

  - Connecteur
  - Principal
  - Device
  - Fonction
  - Logger
  - Ressource
  - Abonnement

L'exemple suivant `DeviceDefinition` fait référence à trois objets `DeviceDefinitionVersion` qui contiennent chacun plusieurs objets `Device`. Un seul objet `DeviceDefinitionVersion` à la fois est utilisé dans un groupe.



Mise à jour de groupes

Dans l'API AWS IoT Greengrass, vous utilisez des versions pour mettre à jour la configuration d'un groupe. Les versions étant immuables, pour ajouter, supprimer ou modifier des composants de

groupe, vous devez créer DefinitionVersiondes objets contenant des entités nouvelles ou mises à jour.

Vous pouvez associer de nouveaux DefinitionVersionsobjets à des objets de définition nouveaux ou existants. Par exemple, vous pouvez utiliser l'action `CreateFunctionDefinition` pour créer un objet `FunctionDefinition` qui inclut l'objet `FunctionDefinitionVersion` comme version initiale, ou vous pouvez utiliser l'action `CreateFunctionDefinitionVersion` et faire référence à un objet `FunctionDefinition` existant.

Après avoir créé les composants de votre groupe, vous en créez un `GroupVersion` qui contient tous les DefinitionVersionobjets que vous souhaitez inclure dans le groupe. Vous déployez ensuite l'objet `GroupVersion`.

Pour déployer un objet `GroupVersion`, celui-ci doit faire référence à un objet `CoreDefinitionVersion` qui contient exactement un objet `Core`. Toutes les entités référencées doivent être membres du groupe. En outre, un rôle de service Greengrass doit être associé à votre rôle Compte AWS dans l'Région AWSendroit où vous déployez le. `GroupVersion`

> **Note**
>
> Les actions `Update` dans l'API sont utilisées pour modifier le nom d'un objet `Group` ou de composant Definition.

Mettre à jour les entités qui font référence à AWS des ressources

Les fonctions Lambda et les ressources secrètes de Greengrass définissent les propriétés spécifiques à Greengrass et font également référence aux ressources correspondantes. AWS Pour mettre à jour ces entités, vous pouvez apporter des modifications à la AWS ressource correspondante plutôt qu'à vos objets Greengrass. Par exemple, les fonctions Lambda font référence à une fonction dans AWS Lambda et définissent également le cycle de vie et d'autres propriétés spécifiques au groupe Greengrass.

- Pour mettre à jour le code de fonction Lambda ou les dépendances packagées, apportez vos modifications dans. AWS Lambda Lors du déploiement de groupe suivant, ces modifications sont extraites de AWS Lambda et copiées dans votre environnement local.
- Pour mettre à jour les propriétés spécifiques à Greengrass, vous créez un objet `FunctionDefinitionVersion` qui contient les propriétés `Function` mises à jour.

> **Note**
>
> Les fonctions Lambda Greengrass peuvent référencer une fonction Lambda par un alias ARN ou une version ARN. Si vous référencez l'ARN d'alias (recommandé), vous n'avez pas besoin de mettre à jour votre `FunctionDefinitionVersion` (ou `SubscriptionDefinitionVersion`) lorsque vous publiez une nouvelle version de fonction dans AWS Lambda. Pour plus d'informations, consultez the section called "Référence de fonctions par alias ou version".

Consultez aussi

- the section called "Obtention des notifications de déploiement"
- the section called "Réinitialiser les déploiements"
- the section called "Créer des déploiements en bloc"
- Dépannage des problèmes de déploiement

- Référence d'API AWS IoT Greengrass Version 1
- AWS IoT Greengrasscommandes dans la référence des AWS CLI commandes

Obtention des notifications de déploiement

Amazon EventBridge Les règles d'événement vous fournissent des notifications sur les changements d'état pour vos déploiements de groupe Greengrass. EventBridge fournit un flux d'événements système en temps quasi réel qui décrivent les modifications apportées àAWSAWS.AWS IoT Greengrassenvoie ces événements à EventBridge sur unau moins une foisbase. Cela signifie queAWS IoT Greengrasspeut envoyer plusieurs copies d'un événement donné pour garantir la livraison. En outre, les écouteurs d'événements peuvent ne pas recevoir les événements dans l'ordre dans lequel ces derniers se produisent.

> **Note**
>
> Amazon EventBridge est un service de bus d'événement que vous pouvez utiliser pour connecter vos applications à des données provenant de diverses sources, telles queAppareils noyau Greengrasset les notifications de déploiement. Pour de plus amples

informations, veuillez consulterQu'est-ce qu'Amazon EventBridge?dans leAmazon EventBridge Guide de l'utilisateur.

AWS IoT Greengrass émet un événement lorsque des déploiements de groupe changent d'état. Vous pouvez créer une EventBridge règle qui s'exécute pour toutes les transitions d'état ou les transitions vers les états que vous spécifiez. Lorsqu'un déploiement passe à un état qui déclenche une règle, EventBridge appelle les actions cibles définies dans la règle. Cela vous permet d'envoyer des notifications, de capturer des informations sur les événements, de prendre des mesures correctives ou de déclencher d'autres événements en réponse à un changement d'état. Par exemple, vous pouvez créer des règles pour les cas d'utilisation suivants :

- Mise en route des opérations post-déploiement, telles que le téléchargement d'assets et la notification au personnel.
- Envoyer des notifications en cas de réussite ou de l'échec d'un déploiement.
- Publier des métriques personnalisées sur les événements de déploiement.

AWS IoT Greengrass émet un événement lorsqu'un déploiement passe aux états suivants : `Building`, `InProgress`, `Success` et `Failure`.

> **Note**
>
> La surveillance du statut d'une opération de déploiement en bloc n'est pas prise en charge actuellement. Cependant, AWS IoT Greengrass émet des événements de changement d'état pour les déploiements de groupe individuels qui font partie d'un déploiement en bloc.

Événement de changement de statut de déploiement de groupe

L' événement pour un changement d'état de déploiement utilise le format suivant :

```
{
    "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"
    }
}
```

Vous pouvez créer des règles qui s'appliquent à un ou plusieurs groupes. Vous pouvez filtrer les règles selon un ou plusieurs des types et états de déploiement suivants :

Types de déploiement

- `NewDeployment`. Premier déploiement d'une version de groupe.
- `ReDeployment`. Redéploiement d'une version de groupe.
- `ResetDeployment`. Supprime les informations de déploiement stockées dans laAWS Cloudet sur leAWS IoT GreengrassCœurs. Pour plus d'informations, consultez the section called “Réinitialiser les déploiements”.
- `ForceResetDeployment`. Supprime les informations de déploiement stockées dans laAWS Cloudet signale la réussite sans attendre que le noyau réponde. Supprime également des informations de déploiement stockées sur le noyau si le noyau est connecté ou lors de sa prochaine connexion.

États de déploiement

- `Building`. AWS IoT Greengrass valide la configuration du groupe et construit les artefacts de déploiement.
- `InProgress`. Le déploiement est en cours sur leAWS IoT GreengrassCœurs.
- `Success`. Le déploiement a réussi.
- `Failure`. Le déploiement a échoué.

Il est possible que les événements soient dupliqués ou hors service. Pour déterminer l'ordre des événements, utilisez la propriété `time`.

> **Note**
>
> AWS IoT Greengrass n'utilise pas la propriété `resources`, qui est donc toujours vide.

Conditions préalables à la création d' EventBridge règles

Avant de créer une EventBridge règle pourAWS IoT Greengrass, procédez comme suit :

- Familiarisez-vous avec les événements, les règles et les cibles de EventBridge.
- Créez et configurez les cibles appelées par votre EventBridge règles. Les règles peuvent appeler de nombreux types de cibles, notamment :
  - Amazon Simple Notification Service (Amazon SNS)
  - Fonctions AWS Lambda
  - Amazon Kinesis Video Streams
  - Files d'attente Amazon Simple Queue Service (Amazon SQS)

Pour de plus amples informations, veuillez consulterQu'est-ce qu'Amazon EventBridge?etMise en route avec Amazon EventBridgedans leAmazon EventBridge Guide de l'utilisateur.

Configurer les notifications de déploiement (console)

Pour créer une EventBridge règle qui publie une rubrique Amazon SNS lorsque l'état du déploiement change pour un groupe. Cela permet aux serveurs web, aux adresses e-mail et aux autres abonnés à la rubrique de répondre à l'événement. Pour de plus amples informations, veuillez consulterCréation d'une EventBridge règle qui se déclenche sur un événement à partir d'unAWSressourcedans leAmazon EventBridge Guide de l'utilisateur.

1. Ouverture d'Amazon EventBridgeconsole.
2. Dans le volet de navigation, choisissez Rules.
3. Choisissez Create rule.
4. Saisissez un nom et une description pour la règle.

   Une règle ne peut pas avoir le même nom qu'une autre règle de la même région et sur le même bus d'événement.

5. Pour Event bus (Bus d'événement), sélectionnez le bus d'événement que vous souhaitez associer à cette règle. Si vous souhaitez que cette règle corresponde aux événements provenant de votre compte, sélectionnezAWSBus d'événement par défaut. Lorsqu'un service AWS de votre compte émet un événement, il accède toujours au bus d'événement par défaut de votre compte.
6. Pour Rule type (Type de règle), choisissez Rule with an event pattern (Règle avec un modèle d'événement).
7. Choisissez Next (Suivant).
8. Pour Event source (Origine de l'événement), choisissez AWSservices (Services).
9. PourModèle d'événement, choisissezAWSservices.
10. PourAWSservice, choisissez Greengrass.
11. Dans Event type (Type d'événement), choisissez Greengrass Deployment Status Change - Modifier l'état du déploiement Greengrass).

> **Note**
>
> LeAWSAppel d'API via CloudTraille type d'événement est basé surAWS IoT GreengrassIntégration de àAWS CloudTrail. Vous pouvez utiliser cette option pour créer des règles initiées par des appels en lecture ou en écriture vers laAWS IoT GreengrassAPI. Pour plus d'informations, consultez the section called “Journalisation des appels d'API AWS IoT Greengrass avec AWS CloudTrail”.

12. Choisissez les états de déploiement qui déclenchent une notification.
    - Pour recevoir des notifications pour tous les événements de modification d'état, choisissez Any state (N'importe quel état).
    - Pour recevoir des notifications pour certains événements de modification d'état uniquement, choisissez Specific state(s) (État(s) spécifique(s)), puis choisissez les états cibles.
13. Choisissez les types de déploiement qui déclenchent une notification.
    - Pour recevoir des notifications pour tous les types de déploiement, choisissez Any state (N'importe quel état).
    - Pour recevoir des notifications pour certains types de déploiement uniquement, choisissez Specific state(s) (État(s) spécifique(s)), puis choisissez les types de déploiement cibles.
14. Choisissez Next (Suivant).
15. PourTypes de cible, choisissezAWSservice.

16. PourSélectionnez une cible, configurez votre cible. Cet exemple utilise une rubrique Amazon SNS, mais vous pouvez configurer d'autres types de cible pour envoyer des notifications.

    a. Pour Target (Cible), choisissez SNS topic (Rubrique SNS).

    b. Pour Topic (Rubrique), choisissez votre rubrique cible.

    c. Choisissez Next (Suivant).

17. UNDERÉtiquettes, définissez les balises pour la règle ou laissez les champs vides.
18. Choisissez Next (Suivant).
19. Consultez les détails de la règle et choisissez Create rule (Créer une règle).

Configurer les notifications de déploiement (interface de ligne de commande)

Pour créer une EventBridge règle qui publie une rubrique Amazon SNS lorsque l'état du déploiement change pour un groupe. Cela permet aux serveurs web, aux adresses e-mail et aux autres abonnés à la rubrique de répondre à l'événement.

1. Créez la règle .

   • Remplacez*group-id*avec l'ID de votreAWS IoT Greengrass.

   ```
   aws events put-rule \
     --name TestRule \
     --event-pattern "{\"source\": [\"aws.greengrass\"], \"detail\": {\"group-id\":
    [\"group-id\"]}}"
   ```

   Les propriétés qui sont omises dans le modèle sont ignorées.

2. Ajoutez la rubrique en tant que cible de règle.

   • Remplacez*fil de sujet*avec l'ARN de votre rubrique Amazon SNS.

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

> **Note**
>
> Pour autoriser Amazon EventBridge pour appeler votre rubrique cible, vous devez ajouter une stratégie basée sur les ressources à votre rubrique. Pour de plus amples informations, veuillez consulterAutorisations Amazon SNSdans leAmazon EventBridge Guide de l'utilisateur.

Pour de plus amples informations, veuillez consulterÉvénements et modèles d'événements dans EventBridgedans leAmazon EventBridge Guide de l'utilisateur.

Configurer les notifications de déploiement (AWS CloudFormation)

UtiliserAWS CloudFormationmodèles à créer EventBridge Règles qui envoient des notifications sur les changements d'état pour vos déploiements de groupe Greengrass. Pour de plus amples informations, veuillez consulterAmazon EventBridge Référence de type de ressourcedans leAWS CloudFormationGuide de l'utilisateur.

Consulter aussi

- Déploiement de groupes AWS IoT Greengrass
- Qu'est-ce qu'Amazon EventBridge?dans leAmazon EventBridge Guide de l'utilisateur

Réinitialiser les déploiements

Cette fonctionnalité est disponible pour AWS IoT Greengrass Core v1.1 et versions ultérieures.

Vous pouvez souhaiter réinitialiser les déploiements d'un groupe pour :

- Supprimez le groupe, par exemple lorsque vous souhaitez déplacer le noyau du groupe vers un autre groupe ou lorsque le noyau du groupe a été redessiné. Avant de supprimer un groupe, vous devez réinitialiser les déploiements du groupe pour utiliser le noyau avec un autre groupe Greengrass.
- Déplacer le noyau du groupe vers un autre groupe.
- Restaurer l'état antérieur d’un groupe avant tous les déploiements.
- Supprimer la configuration de déploiement de l'appareil principal.

- Supprimer des données sensibles de l'appareil principal ou du cloud.
- Déployer une nouvelle configuration sur un noyau sans avoir à remplacer le noyau par un autre dans le groupe actuel.

> **Note**
>
> La fonctionnalité de réinitialisation des déploiements n'est pas disponible dans la version 1.0.0 du logiciel AWS IoT Greengrass Core. Vous ne pouvez pas supprimer un groupe qui a été déployé à l'aide de la version 1.0.0.

L'opération de réinitialisation des déploiements nettoie d'abord toutes les informations de déploiement stockées dans le cloud pour un groupe donné. Il demande ensuite au périphérique principal du groupe de nettoyer également toutes les informations relatives au déploiement (fonctions Lambda, journaux des utilisateurs, base de données fantôme et certificat de serveur, mais pas les certificats `config.json` définis par l'utilisateur ou les certificats principaux de Greengrass). Vous ne pouvez pas lancer une réinitialisation des déploiements pour un groupe si ce dernier a un déploiement dont l'état est `In Progress` ou `Building`.

Réinitialisez les déploiements depuis la console AWS IoT

Vous pouvez réinitialiser les déploiements de groupe à partir de la page de configuration de groupe de la AWS IoT console.

1. Dans le volet de navigation de la AWS IoT console, sous Gérer, développez les appareils Greengrass, puis choisissez Groups (V1).
2. Choisissez le groupe cible.
3. Dans l'onglet Déploiements, choisissez Réinitialiser les déploiements.
4. Dans la boîte de dialogue Réinitialiser les déploiements pour ce groupe Greengrass, **`confirm`** tapez pour accepter, puis choisissez Réinitialiser le déploiement.

Réinitialiser les déploiements avec l'API AWS IoT Greengrass

Vous pouvez utiliser l'action `ResetDeployments` dans l'AWS CLI, l'API AWS IoT Greengrass ou le kit de développement SDK AWS pour réinitialiser les déploiements. Les exemples de cette rubrique utilisent l’interface de ligne de commande.

```
aws greengrass reset-deployments --group-id GroupId [--force]
```

Arguments destinés à la commande **`reset-deployments`** de l'interface de ligne de commande :

`--group-id`

ID du groupe Utilisez la commande `list-groups` pour obtenir cette valeur.

`--force`

Facultatif. Utilisez ce paramètre si l'appareil principal du groupe a été perdu, volé ou détruit. Si cette option est activée, le processus de déploiement de la réinitialisation signale la réussite une fois que toutes les informations de déploiement dans le cloud ont été nettoyées, sans attendre la réponse d'un appareil principal. Toutefois, si l'appareil principal est ou devient actif, il effectue également des opérations de nettoyage.

La sortie de la commande `reset-deployments` de l'interface de ligne de commande ressemble à ceci :

```
{
    "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"
}
```

Vous pouvez vérifier le statut du déploiement de la réinitialisation avec la commande `get-deployment-status` de l'interface de ligne de commande :

```
aws greengrass get-deployment-status --deployment-id DeploymentId --group-id GroupId
```

Arguments destinés à la commande **`get-deployment-status`** de l'interface de ligne de commande :

`--deployment-id`

ID de déploiement

`--group-id`

ID du groupe

La sortie de la commande `get-deployment-status` de l'interface de ligne de commande ressemble à ceci :

```
{
    "DeploymentStatus": "Success",
    "UpdatedAt": "2017-04-04T00:00:00.000Z"
}
```

`DeploymentStatus` est défini sur `Building` lorsque le déploiement de la réinitialisation est en cours de préparation. Lorsque le déploiement de réinitialisation est prêt mais que le AWS IoT Greengrass noyau n'a pas pris en charge le déploiement de réinitialisation, `DeploymentStatus` c'est le cas`InProgress`.

Si l'opération de réinitialisation échoue, les informations relatives à l'erreur sont renvoyées dans la réponse.

Consultez aussi

- Déploiement de groupes AWS IoT Greengrass
- ResetDeployments dans la référence de AWS IoT Greengrass Version 1 l'API
- GetDeploymentStatusdans la référence de AWS IoT Greengrass Version 1 l'API

Créer des déploiements en bloc pour des groupes

Vous pouvez utiliser de simples appels d'API pour déployer un grand nombre de groupes Greengrass à la fois. Ces déploiements sont déclenchés avec une vitesse de transmission adaptative qui possède une limite supérieure fixe.

Ce didacticiel explique comment utiliser la AWS CLI pour créer et surveiller un déploiement de groupes en bloc dans AWS IoT Greengrass. L'exemple de déploiement en bloc dans ce tutoriel contient plusieurs groupes. Vous pouvez utiliser l'exemple dans votre implémentation pour ajouter autant de groupes que nécessaire.

Le didacticiel contient les étapes détaillées suivantes :

1. Créer et charger le fichier d'entrée du déploiement en bloc
2. Créer et configurer un rôle d'exécution IAM pour les déploiements en bloc

3. Autoriser votre rôle d'exécution à accéder à votre compartiment S3
4. Déployer les groupes
5. Test du déploiement

Prérequis

Pour suivre ce didacticiel, vous devez disposer des éléments suivants :

- Un ou plusieurs groupes Greengrass déployables. Pour plus d'informations sur la création des groupes et des noyaux AWS IoT Greengrass, consultez Commencer avec AWS IoT Greengrass.
- La AWS CLI installée et configurée sur votre machine. Pour plus d'informations, consultez le Guide de l'utilisateur AWS CLI.
- Un compartiment S3 créé dans le mêmeRégion AWScommeAWS IoT Greengrass. Pour plus d'informations, consultezCréation et configuration d'un compartiment S3dans leManuel de l'utilisateur Amazon Simple Service.

> **Note**
>
> Actuellement, les compartiments activés SSE KMS ne sont pas pris en charge.

Étape 1 : Créer et charger le fichier d'entrée du déploiement en bloc

Au cours de cette étape, vous créez un fichier d'entrée de déploiement et vous le chargez dans votre compartiment Amazon S3. Ce fichier est sérialisé, séparé par des lignes d'un fichier JSON qui contient des informations sur chaque groupe de votre déploiement en bloc. AWS IoT Greengrass utilise ces informations pour déployer chaque groupe en votre nom lorsque vous initialisez votre groupe de déploiement en bloc.

1. Exécutez la commande suivante pour obtenir l'`groupId` de chaque groupe que vous souhaitez déployer. Vous entrez l'`groupId` dans votre fichier d'entrée de déploiement en bloc de telle sorte que AWS IoT Greengrass puisse identifier chaque groupe à déployer.

> **Note**
>
> Vous pouvez également trouver ces valeurs dansAWS IoTconsole L'ID du groupe s'affiche sur la page Paramètres du groupe. Les ID de version de groupe sont affichés dans leDéploiementsonglet.

```
aws greengrass list-groups
```

La réponse contient des informations sur chaque groupe de votre compte AWS IoT Greengrass :

```
{
  "Groups": [
    {
      "Name": "string",
      "Id": "string",
      "Arn": "string",
      "LastUpdatedTimestamp": "string",
      "CreationTimestamp": "string",
      "LatestVersion": "string",
      "LatestVersionArn": "string"
    }
  ],
  "NextToken": "string"
}
```

Exécutez la commande suivante pour obtenir l'`groupVersionId` de chaque groupe que vous souhaitez déployer.

```
list-group-versions --group-id groupId
```

La réponse contient des informations sur toutes les versions du groupe. Prenez en compte les informations de`Version`pour la version de groupe que vous souhaitez utiliser.

```
{
  "Versions": [
    {
      "Arn": "string",
      "Id": "string",
      "Version": "string",
      "CreationTimestamp": "string"
    }
  ],
  "NextToken": "string"
}
```

2. Dans le terminal de votre ordinateur ou l'éditeur de votre choix, créez un fichier,*MyBulkDeploymentInputFile*, à partir de l'exemple suivant. Ce fichier contient des informations sur chaque groupe AWS IoT Greengrass à inclure dans un déploiement en bloc. Bien que cet exemple définisse plusieurs groupes, pour ce didacticiel, votre fichier peut en contenir un seul.

> **Note**
>
> La taille de ce fichier doit être inférieure à 100 Mo.

```
{"GroupId":"groupId1", "GroupVersionId":"groupVersionId1",
 "DeploymentType":"NewDeployment"}
{"GroupId":"groupId2", "GroupVersionId":"groupVersionId2",
 "DeploymentType":"NewDeployment"}
{"GroupId":"groupId3", "GroupVersionId":"groupVersionId3",
 "DeploymentType":"NewDeployment"}
...
```

Chaque enregistrement (ou ligne) contient un objet de groupe. Chaque objet de groupe contient son `GroupId` et `GroupVersionId` correspondants et un `DeploymentType`. Actuellement, AWS IoT Greengrass prend uniquement en charge les types de déploiement en bloc `NewDeployment`.

Enregistrez et fermez votre fichier. Prenez note de l'emplacement du fichier.

3. Utilisez la commande suivante dans votre terminal pour charger votre fichier d'entrée dans votre compartiment Amazon S3. Remplacez le chemin de fichier par l'emplacement et le nom de votre fichier. Pour de plus amples informations, veuillez consulter Ajouter un objet à un compartiment.

```
aws s3 cp path/MyBulkDeploymentInputFile s3://my-bucket/
```

Étape 2 : Créer et configurer un rôle d'exécution IAM

Au cours de cette étape, vous utilisez la console IAM pour créer un rôle d'exécution autonome. Vous pouvez alors établir une relation d'approbation entre le rôle etAWS IoT Greengrasset assurez-vous que votre utilisateur IAM possède`PassRole`privilèges pour votre rôle d'exécution. Cela permet à AWS IoT Greengrass d'assumer votre rôle d'exécution et de créer les déploiements en votre nom.

1. Utilisez la stratégie suivante pour créer un rôle d'exécution. Ce document de stratégie permet à AWS IoT Greengrass d'accéder à votre fichier d'entrée de déploiement en bloc lorsqu'il crée chaque déploiement en votre nom.

   Pour de plus amples informations sur la création d'un rôle IAM et la délégation des autorisations, veuillez consulter Création de rôles 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**
>
> Cette stratégie doit avoir une ressource pour chaque groupe ou version de groupe dans votre fichier d'entrée de déploiement en bloc pour être déployée par AWS IoT Greengrass. Pour autoriser l'accès à tous les groupes, spécifiez un astérisque pour `Resource` :
>
> ```
> "Resource": ["*"]
> ```

2. Modifiez la relation d'approbation pour votre rôle d'exécution pour inclure AWS IoT Greengrass. Cela permet à AWS IoT Greengrass d'utiliser votre rôle d'exécution et les autorisations attachées à celui-ci. Pour de plus amples informations, veuillez consulter Modification de la relation d'approbation pour un rôle existant.

   Nous vous recommandons également d'inclure`aws:SourceArn`et`aws:SourceAccount`Clés de contexte de condition globale dans votre stratégie d'approbation pour empêcherdéputé confusproblème de sécurité. Les clés contextuelles de condition limitent l'accès afin d'autoriser uniquement les demandes provenant du compte spécifié et de l'espace de travail Greengrass. Pour plus d'informations sur le problème du député confus, veuillez consulterPrévention du problème de l'adjoint confus entre services.

   ```
   {
     "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. Donnez IAM`PassRole`autorisations de votre rôle d'exécution à votre utilisateur IAM. Cet utilisateur IAM est celui utilisé pour initier le déploiement en bloc.`PassRole`Les autorisations permettent à votre utilisateur IAM de transmettre votre rôle d'exécution àAWS IoT Greengrasspour une utilisation. Pour plus d'informations, consultez la section Octroi d'autorisations à un utilisateur pour transférer un rôle à un service AWS.

   Utilisez l'exemple suivant pour mettre à jour la stratégie IAM attachée à votre rôle d'exécution. Modifiez cet exemple, si nécessaire.

```
{
   "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"
               }
           }
       }
   ]
}
```

Étape 3 : Autoriser votre rôle d'exécution à accéder à votre compartiment S3

Pour démarrer votre déploiement en bloc, votre rôle d'exécution doit être en mesure de lire votre fichier d'entrée de déploiement en bloc à partir de votre compartiment Amazon S3. Attachez l'exemple de stratégie suivant à votre compartiment Amazon S3 pour`GetObject`Les autorisations sont accessibles à votre rôle d'exécution.

Pour de plus amples informations, veuillez consulter Comment ajouter une stratégie de compartiment S3 ?

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

Vous pouvez utiliser la commande suivante dans votre terminal pour vérifier votre stratégie de compartiment :

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

> **Note**
>
> Vous pouvez modifier directement votre rôle d'exécution pour accorder l'autorisation à votre compartiment Amazon S3`GetObject`autorisations à la place. Pour ce faire, attachez l'exemple de stratégie suivant à votre rôle d'exécution.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::my-bucket/objectKey"
        }
    ]
}
```

Étape 4 : Déployer les groupes

Au cours de cette étape, vous allez lancer une opération de déploiement en bloc pour toutes les versions de groupe configurées dans votre fichier d'entrée de déploiement en bloc. L'action de déploiement pour chacune de vos versions de groupe est de type `NewDeploymentType`.

> **Note**
>
> Il n'est pas possible d'appeler StartBulkDeployment pendant qu'un autre déploiement en bloc à partir du même compte est encore en cours d'exécution. La demande est rejetée.

1. Pour démarrer le déploiement en bloc, utilisez la commande suivante :

   Nous vous recommandons d'inclure un jeton `X-Amzn-Client-Token` dans chaque demande StartBulkDeployment. Ces demandes sont idempotentes en ce qui concerne le jeton et les paramètres de demande. Ce jeton peut être n'importe quel caractère unique, sensible à la casse et peut contenir un maximum de 64 caractères 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"
          }"
```

La commande doit entraîner la réussite de code de statut `200`, ainsi que la réponse suivante :

```
{
  "bulkDeploymentId": UUID
}
```

Prenez note de l'ID de déploiement en bloc. Il peut être utilisé pour vérifier le statut de votre déploiement en bloc.

> **Note**
>
> Bien que les opérations de déploiement en bloc ne soient pas prises en charge actuellement, vous pouvez créer Amazon EventBridge pour recevoir des notifications sur les changements de statut de déploiement pour des groupes individuels. Pour plus d'informations, consultez the section called “Obtention des notifications de déploiement”.

2. Utilisez la commande suivante pour vérifier le statut de votre déploiement en bloc.

```
aws greengrass get-bulk-deployment-status --bulk-deployment-id 1234567
```

La commande doit retourner un code de statut réussi `200` en plus d'une charge utile JSON d'informations :

```
 {
  "BulkDeploymentStatus": Running,
  "Statistics": {
     "RecordsProcessed": integer,
     "InvalidInputRecords": integer,
     "RetryAttempts": integer
```

```
  },
  "CreatedAt": "string",
  "ErrorMessage": "string",
  "ErrorDetails": [
    {
      "DetailedErrorCode": "string",
      "DetailedErrorMessage": "string"
    }
  ]
}
```

`BulkDeploymentStatus` contient le statut actuel de l'exécution en bloc. L'exécution peut avoir l'un des six statuts suivants :

- `Initializing`. La requête de déploiement en bloc a été reçue et l'exécution se prépare à démarrer.
- `Running`. L'exécution du déploiement en bloc a commencé.
- `Completed`. L'exécution du déploiement en bloc a fini de traiter tous les enregistrements.
- `Stopping`. L'exécution du déploiement en bloc a reçu une commande pour s'arrêter et sera suspendue d'ici peu. Vous ne pouvez pas démarrer un nouveau déploiement en bloc pendant qu'un déploiement antérieur se trouve dans l'état `Stopping`.
- `Stopped`. L'exécution du déploiement en bloc a été arrêtée manuellement.
- `Failed`. L'exécution du déploiement en bloc a rencontré une erreur et s'est arrêtée. Le champ `ErrorDetails` contient des détails sur l’erreur.

La charge utile JSON inclut également des données statistiques sur la progression du déploiement en bloc. Vous pouvez utiliser ces informations pour déterminer combien de groupes ont été traités et combien ont échoué. Les informations statistiques incluent :

- `RecordsProcessed`: le nombre d'enregistrements de groupe tentés.
- `InvalidInputRecords`: le nombre total d'enregistrements ayant renvoyé une erreur qui ne peut pas être retentée. Par exemple, cela peut se produire si un enregistrement de groupe du fichier d'entrée utilise un format incorrect ou spécifie une version de groupe qui n'existe pas, ou si l'exécution n'accorde pas l'autorisation de déployer un groupe ou une version de groupe.
- `RetryAttempts`: le nombre de tentatives de déploiement ayant renvoyé une erreur qui peut être retentée. Par exemple, une nouvelle tentative est déclenchée si la tentative pour déployer

un groupe renvoie une erreur de limitation. Un déploiement de groupe peut être réessayé jusqu'à cinq fois.

Dans le cas d'un échec de l'exécution du déploiement en bloc, cette charge utile inclut également une section `ErrorDetails` qui peut être utilisée pour le dépannage. Elle contient des informations sur la cause de l'échec de l'exécution.

Vous pouvez vérifier régulièrement le statut du déploiement en bloc pour confirmer qu'il progresse comme prévu. Une fois que le déploiement est terminé, `RecordsProcessed` doit être égal au nombre de groupes de déploiement dans votre fichier d'entrée de déploiement en bloc. Cela indique que chaque enregistrement a été traité.

Étape 5 : Test du déploiement

Utilisez la commande ListBulkDeployments pour rechercher l'ID de votre déploiement en bloc.

```
aws greengrass list-bulk-deployments
```

Cette commande renvoie une liste de tous vos déploiements en bloc, du plus récent au moins récent, y compris votre `BulkDeploymentId`.

```
{
  "BulkDeployments": [
    {
      "BulkDeploymentId": 1234567,
      "BulkDeploymentArn": "string",
      "CreatedAt": "string"
    }
  ],
  "NextToken": "string"
}
```

Maintenant, appelez la commande ListBulkDeploymentDetailedReports pour rassembler des informations détaillées sur chaque déploiement.

```
aws greengrass list-bulk-deployment-detailed-reports --bulk-deployment-id 1234567
```

La commande doit retourner un code de statut réussi `200` en plus d'une charge utile JSON d'informations :

```
{
  "BulkDeploymentResults": [
    {
      "DeploymentId": "string",
      "GroupVersionedArn": "string",
      "CreatedAt": "string",
      "DeploymentStatus": "string",
      "ErrorMessage": "string",
      "ErrorDetails": [
        {
          "DetailedErrorCode": "string",
          "DetailedErrorMessage": "string"
        }
      ]
    }
  ],
  "NextToken": "string"
}
```

Cette charge utile contient généralement une liste paginée de chaque déploiement et son statut de déploiement du plus récent au moins récent. Elle contient également plus d'informations en cas d'échec de l'exécution du déploiement en bloc. Là encore, le nombre total de déploiements répertoriés doit être égal au nombre de groupes que vous avez identifiés dans votre fichier d'entrée de déploiement en bloc.

Les informations renvoyées peuvent changer jusqu'à ce que les déploiements se trouvent dans un état de mise hors service (réussite ou échec). Vous pouvez appeler cette commande périodiquement d'ici là.

Résolution des problèmes de déploiements en bloc

Si le déploiement en bloc n'est pas réussi, vous pouvez essayer les étapes de dépannage suivantes. Exécutez les commandes dans votre terminal.

Résolvez les erreurs de fichier d'entrée

Le déploiement en bloc peut échouer en cas d'erreurs de syntaxe dans le fichier d'entrée de déploiement en bloc. Cela renvoie un statut de déploiement en bloc `Failed` avec un message d'erreur indiquant le numéro de la ligne de la première erreur de validation. Il existe quatre erreurs possibles :

- ```
  InvalidInputFile: Missing GroupId at line number: line number
  ```

  Cette erreur indique que la ligne de fichier d'entrée donnée n'est pas en mesure d'enregistrer le paramètre spécifié. Les paramètres manquants possibles sont l'`GroupId` et l'`GroupVersionId`.

- ```
  InvalidInputFile: Invalid deployment type at line number : line number. Only valid
   type is 'NewDeployment'.
  ```

  Cette erreur indique que la ligne de fichier d'entrée donnée répertorie un type de déploiement non valide. Pour l'instant, le seul type pris en charge est un déploiement `NewDeployment`.

- ```
  Line %s is too long in S3 File. Valid line is less than 256 chars.
  ```

  Cette erreur indique que la ligne de fichier d'entrée donnée est trop longue et doit être raccourcie.

- ```
  Failed to parse input file at line number: line number
  ```

  Cette erreur indique que la ligne de fichier d'entrée donnée n'est pas considérée comme un format json valide.

Vérifier les déploiements en bloc simultanés

Vous ne pouvez pas démarrer un nouveau déploiement en bloc pendant qu'un autre est toujours en cours d'exécution ou dans un état de mise hors service. Cela peut entraîner une `Concurrent Deployment Error`. Vous pouvez utiliser la commande ListBulkDeployments pour vérifier qu'un déploiement en bloc n'est pas en cours d'exécution. Cette commande répertorie vos déploiements en bloc du plus récent au moins récent.

```
{
  "BulkDeployments": [
    {
      "BulkDeploymentId": BulkDeploymentId,
      "BulkDeploymentArn": "string",
      "CreatedAt": "string"
    }
  ],
  "NextToken": "string"
}
```

Utilisez le déploiement `BulkDeploymentId` du premier déploiement en bloc répertorié pour exécuter la commande GetBulkDeploymentStatus. Si votre déploiement en bloc le plus récent est en cours d'exécution (`Initializing` ou `Running`), utilisez la commande suivante pour arrêter le déploiement en bloc.

```
aws greengrass stop-bulk-deployment --bulk-deployment-id BulkDeploymentId
```

Cette action entraîne le statut `Stopping` jusqu'à ce que le déploiement soit `Stopped`. Une fois que le déploiement a atteint un statut `Stopped`, vous pouvez commencer un nouveau déploiement en bloc.

Check ErrorDetails

Exécutez la commande `GetBulkDeploymentStatus` pour renvoyer une charge utile JSON qui contient des informations sur n'importe quel échec d'exécution du déploiement en bloc.

```
  "Message": "string",
  "ErrorDetails": [
```

```
    {
      "DetailedErrorCode": "string",
      "DetailedErrorMessage": "string"
    }
  ]
```

Lorsqu'une erreur sort, la charge utile JSON `ErrorDetails` renvoyée par cet appel contient plus d'informations sur l'échec d'exécution du déploiement en bloc. Un code de statut d'erreur dans la série `400`, par exemple, indique une erreur d'entrée, dans les paramètres d'entrée ou les dépendances du mandataire.

Vérifier le journal du noyau AWS IoT Greengrass

Si nécessaire, vous pouvez résoudre les problèmes en affichant les journaux du noyau AWS IoT Greengrass. Vous pouvez également utiliser les commandes suivantes pour afficher `runtime.log` :

```
cd /greengrass/ggc/var/log
sudo cat system/runtime.log | more
```

Pour plus d'informations sur la journalisation AWS IoT Greengrass, consultez Surveillance avec les journaux AWS IoT Greengrass.

Consulter aussi

Pour plus d'informations, consultez les ressources suivantes :

- Déploiement de groupes AWS IoT Greengrass
- Commandes d'API Amazon S3dans leAWS CLIRéférence des commandes
- AWS IoT Greengrasscommandesdans leAWS CLIRéférence des commandes

Exécuter les fonctions Lambda sur le noyau AWS IoT Greengrass

AWS IoT Greengrassfournit un environnement d'exécution Lambda conteneurisé pour le code défini par l'utilisateur dans lequel vous créez. AWS Lambda Les fonctions Lambda déployées sur un AWS IoT Greengrass cœur s'exécutent dans le moteur d'exécution Lambda local du cœur. Les fonctions Lambda locales peuvent être déclenchées par des événements locaux, des messages provenant du cloud et d'autres sources, ce qui apporte des fonctionnalités de calcul locales aux appareils clients. Par exemple, vous pouvez utiliser les fonctions Lambda de Greengrass pour filtrer les données des appareils avant de les transmettre au cloud.

Pour déployer une fonction Lambda sur un noyau, vous ajoutez la fonction à un groupe Greengrass (en faisant référence à la fonction Lambda existante), vous configurez les paramètres spécifiques au groupe pour la fonction, puis vous déployez le groupe. Si la fonction accède AWS aux services, vous devez également ajouter les autorisations requises au rôle de groupe Greengrass.

Vous pouvez configurer les paramètres qui déterminent le mode d'exécution des fonctions Lambda, notamment les autorisations, l'isolation, les limites de mémoire, etc. Pour plus d'informations, consultez the section called “Contrôle de l'exécution de la fonction Greengrass Lambda”.

> **Note**
>
> Ces paramètres permettent également d'exécuter AWS IoT Greengrass dans un conteneur Docker. Pour plus d'informations, consultez the section called “Exécuter AWS IoT Greengrass dans un conteneur Docker”.

Le tableau suivant répertorie les environnements d'exécution AWS Lambda pris en charge et les versions du logiciel AWS IoT Greengrass Core qu'ils peuvent y exécuter.

Langage ou plateforme	Version de GGC
Python 3.8	1.11
Python 3.7	1.9 ou version ultérieure
Python 2.7 *	1.0 ou version ultérieure

Langage ou plateforme	Version de GGC
Java 8	1.1 ou version ultérieure
Node.js 12,x *	1.10 ou version ultérieure
Node.js 8.10 *	1.9 ou version ultérieure
Node.js 6.10 *	1.1 ou version ultérieure
C, C++	1.6 ou version ultérieure

* Vous pouvez exécuter des fonctions Lambda qui utilisent ces environnements d'exécution sur les versions prises en charge deAWS IoT Greengrass, mais vous ne pouvez pas les créer dans. AWS Lambda Si le runtime de votre appareil est différent du runtime AWS Lambda spécifié pour cette fonction, vous pouvez choisir votre propre environnement d'exécution en utilisant `FunctionRuntimeOverride` in. `FunctionDefintionVersion` Pour plus d'informations, consultez CreateFunctionDefinition. Pour plus d'informations sur les environnements d'exécution pris en charge, consultez la politique de support des environnements d'exécution dans le manuel du AWS Lambda développeur.

SDK pour les fonctions Greengrass Lambda

AWSfournit trois SDK qui peuvent être utilisés par les fonctions Greengrass Lambda exécutées sur un cœur. AWS IoT Greengrass Ces kits SDK se trouvent dans différents packages, de sorte que les fonctions peuvent les utiliser simultanément. Pour utiliser un SDK dans une fonction Lambda Greengrass, incluez-le dans le package de déploiement de la fonction Lambda vers lequel vous le téléchargez. AWS Lambda

Kits SDK AWS IoT Greengrass Core

Permet aux fonctions Lambda locales d'interagir avec le cœur pour :

- Échange de messages MQTT avec AWS IoT Core.
- Échangez des messages MQTT avec des connecteurs, des appareils clients et d'autres fonctions Lambda du groupe Greengrass.
- Interagissez avec le service shadow local.
- Appelez d'autres fonctions Lambda locales.

- Accédez aux ressources de secret.
- Interagissez avec le gestionnaire de flux.

AWS IoT Greengrassfournit le SDK de AWS IoT Greengrass base dans les langues et plateformes suivantes sur GitHub.

- AWS IoT GreengrassSDK de base pour Java
- AWS IoT GreengrassSDK de base pour Node.js
- AWS IoT GreengrassSDK de base pour Python
- AWS IoT GreengrassSDK de base pour C

Pour inclure la dépendance du SDK AWS IoT Greengrass principal dans le package de déploiement de la fonction Lambda :

1. Téléchargez la langue ou la plate-forme du package AWS IoT Greengrass Core SDK correspondant au temps d'exécution de votre fonction Lambda.
2. Décompressez le package téléchargé pour obtenir le kit SDK. Le kit SDK est représenté par le dossier `greengrasssdk`.
3. Incluez-le `greengrasssdk` dans le package de déploiement de la fonction Lambda qui contient votre code de fonction. Il s'agit du package dans lequel vous téléchargez la fonction Lambda AWS Lambda lorsque vous créez la fonction Lambda.

StreamManagerClient

Seuls les kits SDK AWS IoT Greengrass Core suivants peuvent être utilisés pour les opérations de gestionnaire de flux :

- SDK Java (v1.4.0 ou version ultérieure)
- SDK Python (v1.5.0 ou version ultérieure)
- SDK Node.js (v1.6.0 ou version ultérieure)

Pour utiliser le SDK AWS IoT Greengrass Core pour Python afin d'interagir avec le gestionnaire de flux, vous devez installer Python 3.7 ou version ultérieure. Vous devez également installer les dépendances à inclure dans vos packages de déploiement de fonctions Python Lambda :

1. Accédez au répertoire SDK qui contient le fichier `requirements.txt`. Ce fichier répertorie les dépendances.

2. Installez les dépendances du kit SDK. Par exemple, exécutez la commande `pip` suivante pour les installer dans le répertoire en cours :

```
pip install --target . -r requirements.txt
```

Installez le SDK AWS IoT Greengrass Core pour Python sur l'appareil principal

Si vous exécutez des fonctions Lambda en Python, vous pouvez également les utiliser `pip`pour installer le SDK AWS IoT Greengrass Core pour Python sur le périphérique principal. Vous pouvez ensuite déployer vos fonctions sans inclure le SDK dans le package de déploiement des fonctions Lambda. Pour plus d'informations, consultez greengrasssdk.

Cette prise en charge est destinée aux noyaux avec des contraintes de taille. Nous vous recommandons d'inclure le SDK dans vos packages de déploiement de fonctions Lambda lorsque cela est possible.

AWS IoT GreengrassKit de développement logiciel (SDK) pour le Machine Learning

Permet aux fonctions Lambda locales d'utiliser des modèles d'apprentissage automatique (ML) déployés sur le cœur de Greengrass sous forme de ressources ML. Les fonctions Lambda peuvent utiliser le SDK pour appeler et interagir avec un service d'inférence local déployé sur le cœur en tant que connecteur. Les fonctions Lambda et les connecteurs ML peuvent également utiliser le SDK pour envoyer des données au connecteur ML Feedback à des fins de téléchargement et de publication. Pour plus d'informations, y compris des exemples de code qui utilisent le kit SDK, consultez the section called “Classification des images ML”, the section called “Détection d'objets ML” et the section called “Retours de ML”.

Le tableau suivant répertorie chaque plateforme ou langage pris en charge, ainsi que les versions compatibles du logiciel AWS IoT Greengrass Core.

Version de SDK	Langage ou plateforme	Version GGC requise	Journal des modifications
1.1.0	Python 3.7 ou 2.7	1.9.3 ou version ultérieure	Ajout de la prise en charge de Python 3.7 et d'un nouveau client `feedback`.
1.0.0	Python 2.7	1.7 ou version ultérieure	Première version.

Pour obtenir des informations sur le téléchargement, consultez the section called “Logiciel du kit SDK de machine learning AWS IoT Greengrass”.

Kits SDK AWS

Permet aux fonctions Lambda locales d'effectuer des appels directs vers AWS des services tels qu'Amazon S3, DynamoDB et. AWS IoT AWS IoT Greengrass Pour utiliser un AWS SDK dans une fonction Greengrass Lambda, vous devez l'inclure dans votre package de déploiement. Lorsque vous utilisez le AWS SDK dans le même package que le SDK AWS IoT Greengrass principal, assurez-vous que vos fonctions Lambda utilisent les espaces de noms corrects. Les fonctions Greengrass Lambda ne peuvent pas communiquer avec les services cloud lorsque le cœur est hors ligne.

Téléchargez les kits SDK AWS depuis le Centre de ressources du démarrage.

Pour plus d'informations sur la création d'un package de déploiement, reportez-vous the section called “Création et empaquetage d'une fonction Lambda” au didacticiel de démarrage ou à la création d'un package de déploiement dans le guide du AWS Lambda développeur.

Migration de fonctions Lambda basées sur le cloud

Le SDK AWS IoT Greengrass principal suit le modèle de programmation du AWS SDK, qui facilite le portage des fonctions Lambda développées pour le cloud vers des fonctions Lambda exécutées sur un cœur. AWS IoT Greengrass

Par exemple, la fonction Lambda Python suivante utilise le AWS SDK for Python (Boto3) pour publier un message sur le sujet `some/topic` dans le cloud :

```
import boto3

iot_client = boto3.client("iot-data")
response = iot_client.publish(
    topic="some/topic", qos=0, payload="Some payload".encode()
)
```

Pour porter la fonction pour un AWS IoT Greengrass noyau, dans l'`import`instruction et l'`client`initialisation, remplacez le nom du `boto3` module par`greengrasssdk`, comme indiqué dans l'exemple suivant :

```
import greengrasssdk

iot_client = greengrasssdk.client("iot-data")
iot_client.publish(topic="some/topic", qos=0, payload="Some payload".encode())
```

> **Note**
>
> Le kit de développement logiciel (SDK) AWS IoT Greengrass Core ne prend en charge que l'envoi des messages MQTT avec QoS = 0. Pour plus d'informations, consultez the section called “Message de qualité de service”.

La similitude entre les modèles de programmation vous permet également de développer vos fonctions Lambda dans le cloud, puis de les migrer vers celles-ci AWS IoT Greengrass avec un minimum d'effort. Les exécutables Lambda ne s'exécutent pas dans le cloud. Vous ne pouvez donc pas utiliser le AWS SDK pour les développer dans le cloud avant le déploiement.

Référencer les fonctions Lambda par alias ou par version

Les groupes Greengrass peuvent référencer une fonction Lambda par alias (recommandé) ou par version. L'utilisation d'un alias facilite la gestion des mises à jour du code, car vous n'avez pas à modifier votre table d'abonnement ou la définition de groupe lorsque le code de fonction est mis à jour. Au lieu de cela, il vous suffit de pointer l'alias vers la nouvelle version de la fonction. Les alias sont convertis en numéros de version lors du déploiement en groupe. Lorsque vous utilisez des alias, la version résolue est mise à jour dans la version vers laquelle l'alias pointe au moment de déploiement.

AWS IoT Greengrassne prend pas en charge les alias Lambda pour les versions $LATEST. Les versions $LATEST ne sont pas liées à des versions de fonctions immuables publiées et peuvent être modifiées à tout moment, ce qui va à l'encontre du AWS IoT Greengrass principe d'immuabilité des versions.

Une pratique courante pour maintenir vos fonctions Greengrass Lambda à jour en cas de modification du code consiste à utiliser un alias nommé dans votre groupe Greengrass et dans **`PRODUCTION`** vos abonnements. Lorsque vous mettez en production de nouvelles versions de votre fonction Lambda, pointez l'alias vers la dernière version stable, puis redéployez le groupe. Vous pouvez également utiliser cette méthode pour restaurer une version précédente.

Contrôle de l'exécution des fonctions Greengrass Lambda à l'aide d'une configuration spécifique au groupe

AWS IoT Greengrassfournit une gestion basée sur le cloud des fonctions de Greengrass Lambda. Bien que le code et les dépendances d'une fonction Lambda soient gérés à l'aideAWS Lambda, vous pouvez configurer le comportement de la fonction Lambda lorsqu'elle s'exécute dans un groupe Greengrass.

Paramètre de configuration spécifiques au groupe

AWS IoT Greengrassfournit les paramètres de configuration spécifiques au groupe suivants pour les fonctions Greengrass Lambda.

Utilisateur et groupe du système

> Identité d'accès utilisée pour exécuter une fonction Lambda. Par défaut, les fonctions Lambda s'exécutent en tant qu'identité d'accès par défaut du groupe. En général, il s'agit des comptes

AWS IoT Greengrass standard (ggc_user et ggc_group). Vous pouvez modifier le paramètre et choisir l'ID utilisateur et l'ID de groupe disposant des autorisations requises pour exécuter la fonction Lambda. Vous pouvez remplacer les deux champs UID et GID ou un seul des deux si vous laissez l'autre champ vide. Ce paramètre vous donne un contrôle plus précis sur l'accès aux ressources de l'appareil. Nous vous recommandons de configurer votre matériel Greengrass avec des limites de ressources, des autorisations de fichiers et des quotas de disque appropriés pour les utilisateurs et les groupes dont les autorisations sont utilisées pour exécuter les fonctions Lambda.

Cette fonctionnalité est disponible pour AWS IoT Greengrass Core v1.7 et versions ultérieures.

> **Important**
>
> Nous vous recommandons d'éviter d'exécuter des fonctions Lambda en tant qu'utilisateur root, sauf en cas de nécessité absolue. Le fait de s'exécuter en tant que root augmente les risques suivants :
>
> - Le risque de modifications involontaires, telles que la suppression accidentelle d'un fichier critique.
> - Le risque que des personnes mal intentionnées mettent en danger vos données et votre appareil.
> - Le risque lié au conteneur disparaît lorsque les conteneurs Docker fonctionnent avec `--net=host` et. `UID=EUID=0`
>
> Si l'exécution en tant que racine est nécessaire, vous devez mettre à jour la configuration AWS IoT Greengrass pour l'activer. Pour plus d'informations, consultez the section called "Exécution d'une fonction Lambda en tant que root".

ID utilisateur du système (numéro)

L'ID utilisateur de l'utilisateur disposant des autorisations requises pour exécuter la fonction Lambda. Ce paramètre n'est disponible que si vous choisissez de l'exécuter en tant qu'autre ID utilisateur/ID de groupe. Vous pouvez utiliser la getent passwd commande sur votre appareil AWS IoT Greengrass principal pour rechercher l'ID utilisateur que vous souhaitez utiliser pour exécuter la fonction Lambda.

Si vous utilisez le même UID pour exécuter des processus et la fonction Lambda sur un appareil principal de Greengrass, votre rôle de groupe Greengrass peut accorder des

informations d'identification temporaires aux processus. Les processus peuvent utiliser les informations d'identification temporaires dans les déploiements principaux de Greengrass.

ID du groupe de systèmes (numéro)

ID de groupe du groupe disposant des autorisations requises pour exécuter la fonction Lambda. Ce paramètre n'est disponible que si vous choisissez de l'exécuter en tant qu'autre ID utilisateur/ID de groupe. Vous pouvez utiliser la getent group commande sur votre appareil AWS IoT Greengrass principal pour rechercher l'ID de groupe que vous souhaitez utiliser pour exécuter la fonction Lambda.

Conteneurisation de la fonction Lambda

Choisissez si la fonction Lambda s'exécute avec la conteneurisation par défaut pour le groupe ou spécifiez la conteneurisation qui doit toujours être utilisée pour cette fonction Lambda.

Le mode de conteneurisation d'une fonction Lambda détermine son niveau d'isolation.

- Les fonctions Lambda conteneurisées s'exécutent en mode conteneur Greengrass. La fonction Lambda s'exécute dans un environnement d'exécution (ou espace de noms) isolé à l'intérieur du conteneur. AWS IoT Greengrass
- Les fonctions Lambda non conteneurisées s'exécutent en mode Sans conteneur. Les fonctions Lambda s'exécutent comme un processus Linux normal sans aucune isolation.

Cette fonctionnalité est disponible pour AWS IoT Greengrass Core v1.7 et versions ultérieures.

Nous vous recommandons d'exécuter les fonctions Lambda dans un conteneur Greengrass, sauf si votre cas d'utilisation exige qu'elles s'exécutent sans conteneurisation. Lorsque vos fonctions Lambda s'exécutent dans un conteneur Greengrass, vous pouvez utiliser les ressources locales et périphériques associées et bénéficier des avantages de l'isolation et d'une sécurité accrue. Avant de modifier la conteneurisation, consultez the section called “Considérations à prendre en compte lors du choix de la conteneurisation des fonctions Lambda”.

> **Note**
>
> Pour s'exécuter sans activer l'espace de noms et le groupe de noms du noyau de votre appareil, toutes vos fonctions Lambda doivent s'exécuter sans conteneurisation. Vous pouvez effectuer cette opération facilement en définissant la conteneurisation par défaut pour le groupe. Pour plus d’informations, consultez the section called “Configuration de la conteneurisation par défaut pour les fonctions Lambda dans un groupe”.

Limite de mémoire

L'allocation de mémoire pour la fonction. La valeur par défaut est 16 Mo.

> **Note**
>
> Le paramètre de limite de mémoire devient indisponible lorsque vous modifiez la fonction Lambda pour qu'elle s'exécute sans conteneurisation. Les fonctions Lambda qui s'exécutent sans conteneurisation n'ont aucune limite de mémoire. Le paramètre de limite de mémoire est supprimé lorsque vous modifiez le paramètre de conteneurisation par défaut de la fonction Lambda ou du groupe pour qu'il s'exécute sans conteneurisation.

Expiration

La durée avant laquelle la fonction ou la demande est mise hors service. Le durée par défaut est de 3 secondes.

Épinglé

Le cycle de vie d'une fonction Lambda peut être à la demande ou de longue durée. La valeur par défaut est à la demande.

Une fonction Lambda à la demande démarre dans un conteneur neuf ou réutilisé lorsqu'elle est invoquée. Les demandes envoyées à la fonction peuvent être traitées par n'importe quel conteneur disponible. Une fonction Lambda de longue durée (ou épinglée) démarre automatiquement après le démarrage et continue de s'exécuter dans son AWS IoT Greengrass propre conteneur (ou bac à sable). Toutes les demandes envoyées à la fonction sont traitées par le même conteneur. Pour plus d'informations, consultez the section called "Configuration du cycle de vie".

Accès en lecture au répertoire /sys

Indique si la fonction peut accéder au dossier /sys de l'hôte. Utilisez cette option lorsque la fonction doit lire les informations sur les appareils depuis le répertoire /sys. La valeur par défaut est false.

> **Note**
>
> Ce paramètre n'est pas disponible lorsque vous exécutez une fonction Lambda sans conteneurisation. La valeur de ce paramètre est supprimée lorsque vous modifiez la fonction Lambda pour qu'elle s'exécute sans conteneurisation.

Type d'encodage

Le type d'encodage attendu de charge utile d'entrée pour la fonction, JSON ou binaire. La valeur par défaut est JSON.

La prise en charge du type d'encodage binaire est disponible à partir du logiciel AWS IoT Greengrass Core v1.5.0 et du kit SDK AWS IoT Greengrass Core v1.1.0. L'acception des données d'entrée binaires peut être utile pour les fonctions qui interagissent avec les données de l'appareil ; en effet, à cause des capacités matérielles restreintes des appareils, il est souvent difficile, voire impossible, pour eux de construire un type de données JSON.

> **Note**
>
> Les exécutables Lambda ne prennent en charge que le type de codage binaire, et non le JSON.

Arguments du processus

Les arguments de la ligne de commande sont transmis à la fonction Lambda lors de son exécution.

Variables d'environnement

Paires clé-valeur qui peuvent transférer dynamiquement des paramètres au code de fonction et aux bibliothèques. Les variables d'environnement locales fonctionnent de la même manière que les variables d'environnement de fonction AWS Lambda, mais sont disponibles dans l'environnement principal.

Stratégies d'accès aux ressources

Liste d'un maximum de 10 ressources locales, de ressources secrètes et de ressources d'apprentissage automatique auxquelles la fonction Lambda est autorisée à accéder, ainsi que

l'autorisation correspondante`read-only`. `read-write` Dans la console, ces ressources affiliées sont répertoriées sur la page de configuration du groupe dans l'onglet Ressources.

Le mode de conteneurisation affecte la manière dont les fonctions Lambda peuvent accéder aux ressources locales des appareils et des volumes ainsi qu'aux ressources d'apprentissage automatique.

- Les fonctions Lambda non conteneurisées doivent accéder aux ressources locales du périphérique et du volume directement via le système de fichiers du périphérique principal.
- Pour permettre aux fonctions Lambda non conteneurisées d'accéder aux ressources d'apprentissage automatique du groupe Greengrass, vous devez définir le propriétaire de la ressource et les propriétés des autorisations d'accès sur la ressource d'apprentissage automatique. Pour plus d'informations, consultez the section called “Accès aux ressources de machine learning”.

Pour plus d'informations sur l'utilisation de l'AWS IoT GreengrassAPI pour définir des paramètres de configuration spécifiques au groupe pour les fonctions Lambda définies par l'utilisateur, CreateFunctionDefinitionreportez-vous à AWS IoT Greengrass Version 1la référence de l'API create-function-definitionou à la référence des commandes. AWS CLI Pour déployer des fonctions Lambda sur un noyau Greengrass, créez une version de définition de fonction contenant vos fonctions, créez une version de groupe qui fait référence à la version de définition de fonction et à d'autres composants du groupe, puis déployez le groupe.

Exécution d'une fonction Lambda en tant que root

Cette fonctionnalité est disponible pour AWS IoT Greengrass Core v1.7 et versions ultérieures.

Avant de pouvoir exécuter une ou plusieurs fonctions Lambda en tant que root, vous devez d'abord mettre à jour la AWS IoT Greengrass configuration pour activer le support. Support pour l'exécution des fonctions Lambda en tant que root est désactivé par défaut. Le déploiement échoue si vous essayez de déployer une fonction Lambda et de l'exécuter en tant que root (UID et GID de 0) et que vous n'avez pas mis à jour la configuration. AWS IoT Greengrass Une erreur similaire à ce qui suit apparaît dans le journal d'exécution (*greengrass_root*/ggc/var/log/system/runtime.log) :

```
lambda(s)
[list of function arns] are configured to run as root while Greengrass is not
 configured to run lambdas with root permissions
```

Important

Nous vous recommandons d'éviter d'exécuter des fonctions Lambda en tant qu'utilisateur root, sauf en cas de nécessité absolue. Le fait de s'exécuter en tant que root augmente les risques suivants :

- Le risque de modifications involontaires, telles que la suppression accidentelle d'un fichier critique.
- Le risque que des personnes mal intentionnées mettent en danger vos données et votre appareil.
- Le risque lié au conteneur disparaît lorsque les conteneurs Docker fonctionnent avec `--net=host` et. `UID=EUID=0`

Pour autoriser les fonctions Lambda à s'exécuter en tant qu'utilisateur root

1. Sur votre appareil AWS IoT Greengrass, accédez au dossier *greengrass-root*/config.

   Note

   Par défaut, *greengrass-root* est le répertoire /greengrass.

2. Modifiez le fichier config.json pour ajouter `"allowFunctionsToRunAsRoot" : "yes"` au champ `runtime`. Par exemple :

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

3. Utilisez les commandes suivantes pour démarrer AWS IoT Greengrass :

   ```
   cd /greengrass/ggc/core
   ```

```
sudo ./greengrassd restart
```

Vous pouvez désormais définir l'ID utilisateur et l'ID de groupe (UID/GID) des fonctions Lambda sur 0 pour exécuter cette fonction Lambda en tant qu'utilisateur root.

Vous pouvez modifier la valeur de `"allowFunctionsToRunAsRoot"` to `"no"` et restart AWS IoT Greengrass si vous souhaitez interdire aux fonctions Lambda de s'exécuter en tant qu'utilisateur root.

Considérations à prendre en compte lors du choix de la conteneurisation des fonctions Lambda

Cette fonctionnalité est disponible pour AWS IoT Greengrass Core v1.7 et versions ultérieures.

Par défaut, les fonctions Lambda s'exécutent dans un AWS IoT Greengrass conteneur. Ce conteneur assure l'isolement entre vos fonctions et l'hôte, ce qui vous offre davantage de sécurité à la fois pour l'hôte et les fonctions dans le conteneur.

Nous vous recommandons d'exécuter les fonctions Lambda dans un conteneur Greengrass, sauf si votre cas d'utilisation exige qu'elles s'exécutent sans conteneurisation. En exécutant vos fonctions Lambda dans un conteneur Greengrass, vous pouvez mieux contrôler la restriction de l'accès aux ressources.

Voici quelques exemples de cas d'exécution sans conteneurisation :

- Vous souhaitez exécuter AWS IoT Greengrass sur un appareil qui ne prend pas en charge le mode conteneur (par exemple, parce que vous utilisez une distribution Linux spécifique ou une version de noyau qui est trop ancienne).
- Vous souhaitez exécuter votre fonction Lambda dans un autre environnement de conteneurs avec son propre OverlayFS, mais vous rencontrez des conflits OverlayFS lorsque vous l'exécutez dans un conteneur Greengrass.
- Vous devez avoir accès aux ressources locales avec des chemins qui ne peuvent pas être déterminés lors du déploiement ou dont le chemin peut changer après le déploiement, notamment avec certains appareils enfichables.
- Vous avez une ancienne application écrite en tant que processus et vous avez rencontré des problèmes lors de son exécution en tant que fonction Lambda conteneurisée.

Différences de conteneurisation

Conteneurisation	Remarques
Conteneur Greengrass	• Toutes les AWS IoT Greengrass fonctionn alités sont disponibles lorsque vous exécutez une fonction Lambda dans un conteneur Greengrass. • Les fonctions Lambda qui s'exécutent dans un conteneur Greengrass n'ont pas accès au code déployé des autres fonctions Lambda, même si elles s'exécutent avec le même ID de groupe. En d'autres termes, vos fonctions Lambda s'exécutent de manière plus isolée les unes des autres. • Étant donné que les fonctions Lambda exécutées dans un AWS IoT Greengrass conteneur font exécuter tous les processus enfants dans le même conteneur que la fonction Lambda, les processus enfants sont interrompus lorsque la fonction Lambda est arrêtée.

Aucun conteneur	• Les fonctionnalités suivantes ne sont pas disponibles pour les fonctions Lambda non conteneurisées :
	  • Limites de mémoire de la fonction Lambda.
	  • Ressources de volumes et d'appareils locales. Vous devez accéder directement aux ressources de l'appareil principal au lieu d'y accéder en tant que membres du groupe Greengrass.
	• Si votre fonction Lambda non conteneur isée accède à une ressource d'apprent issage automatique, vous devez identifier le propriétaire de la ressource et définir des

Conteneurisation	Remarques
	autorisations d'accès sur la ressource, et non sur la fonction Lambda. Cela nécessite le logiciel AWS IoT Greengrass Core v1.10 ou version ultérieure. Pour plus d'informations, consultez the section called "Accès aux ressources de machine learning". • La fonction Lambda dispose d'un accès en lecture seule au code déployé des autres fonctions Lambda exécutées avec le même ID de groupe. • Les fonctions Lambda qui génèrent des processus enfants dans une autre session de processus ou avec un gestionnaire SIGHUP (signal hangup) remplacé, tel que l'utilitaire nohup, ne sont pas automatiquement interrompues lorsque la fonction Lambda parent est arrêtée. AWS IoT Greengrass

> **Note**
>
> Le paramètre de conteneurisation par défaut pour le groupe Greengrass ne s'applique pas aux connecteurs.

La modification de la conteneurisation d'une fonction Lambda peut entraîner des problèmes lors de son déploiement. Si vous avez affecté à votre fonction Lambda des ressources locales qui ne sont plus disponibles avec vos nouveaux paramètres de conteneurisation, le déploiement échoue.

- Lorsque vous remplacez l'exécution d'une fonction Lambda dans un conteneur Greengrass par une fonction sans conteneurisation, les limites de mémoire de la fonction sont supprimées. Vous devez accéder au système de fichiers directement au lieu d'utiliser les ressources locales attachées. Vous devez supprimer les ressources attachées avant le déploiement.
- Lorsque vous passez d'une fonction Lambda exécutée sans conteneurisation à une fonction Lambda exécutée dans un conteneur, votre fonction Lambda perd l'accès direct au système de

fichiers. Vous devez définir une limite de mémoire pour chaque fonction ou accepter la valeur par défaut de 16 Mo. Vous pouvez configurer ces paramètres pour chaque fonction Lambda avant le déploiement.

Pour modifier les paramètres de conteneurisation d'une fonction Lambda

1. Dans le volet de navigation de la AWS IoT console, sous Gérer, développez les appareils Greengrass, puis choisissez Groups (V1).
2. Choisissez le groupe contenant la fonction Lambda dont vous souhaitez modifier les paramètres.
3. Choisissez l'onglet Fonctions Lambda.
4. Sur la fonction Lambda que vous souhaitez modifier, choisissez les points de suspension (...), puis choisissez Modifier la configuration.
5. Modifiez les paramètres de conteneurisation. Si vous configurez la fonction Lambda pour qu'elle s'exécute dans un conteneur Greengrass, vous devez également définir une limite de mémoire et un accès en lecture au répertoire /sys.
6. Choisissez Enregistrer puis Confirmer pour enregistrer les modifications apportées à votre fonction Lambda.

Les modifications prennent effet lorsque le groupe est déployé.

Vous pouvez également utiliser le CreateFunctionDefinitionet CreateFunctionDefinitionVersiondans la référence AWS IoT Greengrass d'API. Si vous modifiez le paramètre conteneurisation, veillez également à mettre à jour les autres paramètres. Par exemple, si vous passez de l'exécution d'une fonction Lambda dans un conteneur Greengrass à une exécution sans conteneurisation, veillez à effacer le paramètre. `MemorySize`

Déterminer les modes d'isolement pris en charge par votre appareil Greengrass

Vous pouvez utiliser le vérificateur de dépendance AWS IoT Greengrass pour déterminer les modes d'isolement (conteneur Greengrass/aucun conteneur) pris en charge par votre appareil Greengrass.

Pour exécuter le vérificateur de dépendance AWS IoT Greengrass

1. Téléchargez et exécutez le vérificateur de AWS IoT Greengrass dépendance depuis le GitHubréférentiel.

```
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. Là où `more` s'affiche, appuyez sur la touche Spacebar pour afficher une autre page de texte.

Pour en savoir plus sur la commande modprobe, exécutez man modprobe dans le terminal.

Définition de l'identité d'accès par défaut pour les fonctions Lambda dans un groupe

Cette fonctionnalité est disponible pour AWS IoT Greengrass Core v1.8 et versions ultérieures.

Pour mieux contrôler l'accès aux ressources de l'appareil, vous pouvez configurer l'identité d'accès par défaut utilisée pour exécuter les fonctions Lambda dans le groupe. Ce paramètre détermine les autorisations par défaut accordées à vos fonctions Lambda lorsqu'elles s'exécutent sur le périphérique principal. Pour remplacer la configuration pour des fonctions individuelles dans le groupe, vous pouvez utiliser la propriété Run as (Exécuter en tant que) de la fonction. Pour plus d'informations, consultez Run as (Exécuter en tant que).

Cette configuration au niveau du groupe est également utilisée pour exécuter le logiciel AWS IoT Greengrass Core sous-jacent. Il s'agit de fonctions Lambda du système qui gèrent les opérations, telles que le routage des messages, la synchronisation instantanée locale et la détection automatique des adresses IP.

L'identité d'accès par défaut peut être configurée pour s'exécuter comme les comptes système AWS IoT Greengrass standard (ggc_user et ggc_group) ou utiliser les autorisations d'un autre utilisateur ou groupe. Nous vous recommandons de configurer votre matériel Greengrass avec des limites de ressources, des autorisations de fichiers et des quotas de disque appropriés pour tous les utilisateurs et groupes dont les autorisations sont utilisées pour exécuter des fonctions Lambda définies par l'utilisateur ou du système.

Pour modifier l'identité d'accès par défaut pour votre groupe AWS IoT Greengrass

1. Dans le volet de navigation de la AWS IoT console, sous Gérer, développez les appareils Greengrass, puis choisissez Groups (V1).

2. Choisissez le groupe dont vous voulez modifier les paramètres.
3. Choisissez l'onglet Fonctions Lambda et, dans la section Environnement d'exécution des fonctions Lambda par défaut, sélectionnez Modifier.
4. Sur la page Modifier l'environnement d'exécution de la fonction Lambda par défaut, sous Utilisateur et groupe du système par défaut, sélectionnez Autre ID utilisateur/ID de groupe.

   Lorsque vous choisissez cette option, les champs ID utilisateur du système (numéro) et ID de groupe système (numéro) s'affichent.
5. Saisissez un ID d'utilisateur, un ID de groupe, ou les deux. Si vous ne remplissez pas l'un des champs, le numéro correspondant du compte système Greengrass (ggc_user ou ggc_group) est utilisé.

   - Dans le champ ID utilisateur du système (numéro), entrez l'ID utilisateur de l'utilisateur qui dispose des autorisations que vous souhaitez utiliser par défaut pour exécuter les fonctions Lambda dans le groupe. Vous pouvez utiliser la commande getent passwd sur votre appareil AWS IoT Greengrass pour rechercher l'ID d'utilisateur.
   - Dans le champ ID du groupe système (numéro), entrez l'ID du groupe disposant des autorisations que vous souhaitez utiliser par défaut pour exécuter les fonctions Lambda dans le groupe. Vous pouvez utiliser la commande getent group sur votre appareil AWS IoT Greengrass pour rechercher l'ID de groupe.

   > **Important**
   >
   > Une exécution en tant qu'utilisateur racine fait peser plus de risques sur vos données et appareils. Ne procédez pas à une exécution en tant qu'utilisateur racine (UID/GID=0), sauf si cela s'avère nécessaire pour votre scénario. Pour plus d'informations, consultez the section called “Exécution d'une fonction Lambda en tant que root”.

Les modifications prennent effet lorsque le groupe est déployé.

Configuration de la conteneurisation par défaut pour les fonctions Lambda dans un groupe

Cette fonctionnalité est disponible pour AWS IoT Greengrass Core v1.7 et versions ultérieures.

Le paramètre de conteneurisation d'un groupe Greengrass détermine la conteneurisation par défaut pour les fonctions Lambda du groupe.

- En mode conteneur Greengrass, les fonctions Lambda s'exécutent par défaut dans un environnement d'exécution isolé à l'intérieur du AWS IoT Greengrass conteneur.
- En mode sans conteneur, les fonctions Lambda s'exécutent par défaut comme des processus Linux classiques.

Vous pouvez modifier les paramètres du groupe pour spécifier la conteneurisation par défaut pour les fonctions Lambda du groupe. Vous pouvez remplacer ce paramètre pour une ou plusieurs fonctions Lambda du groupe si vous souhaitez que les fonctions Lambda s'exécutent avec une conteneurisation différente de la valeur par défaut du groupe. Avant de modifier les paramètres de conteneurisation, consultez the section called “Considérations à prendre en compte lors du choix de la conteneurisation des fonctions Lambda”.

> **⚠ Important**
>
> Si vous souhaitez modifier la conteneurisation par défaut du groupe, mais qu'une ou plusieurs fonctions utilisent une conteneurisation différente, modifiez les paramètres des fonctions Lambda avant de modifier les paramètres du groupe. Si vous modifiez le paramètre de conteneurisation du groupe en premier, les valeurs des paramètres Limite de mémoire et Accès en lecture au répertoire /sys sont ignorées.

Pour modifier les paramètres de conteneurisation de votre groupe AWS IoT Greengrass

1. Dans le volet de navigation de la AWS IoT console, sous Gérer, développez les appareils Greengrass, puis choisissez Groups (V1).
2. Choisissez le groupe dont vous voulez modifier les paramètres.
3. Choisissez l'onglet Fonctions Lambda.
4. Dans Environnement d'exécution de la fonction Lambda par défaut, sélectionnez Modifier.
5. Dans la page Modifier l'environnement d'exécution de la fonction Lambda par défaut, sous Conteneurisation de la fonction Lambda par défaut, modifiez le paramètre de conteneurisation.
6. Choisissez Enregistrer.

Les modifications prennent effet lorsque le groupe est déployé.

Flux de communication pour les fonctions Greengrass Lambda

Les fonctions Greengrass Lambda prennent en charge plusieurs méthodes de communication avec les autres membres du AWS IoT Greengrass groupe, les services locaux et les services cloud (y compris les services). AWS

Communication à l'aide de messages MQTT

Les fonctions Lambda peuvent envoyer et recevoir des messages MQTT en utilisant un modèle de publication et d'abonnement contrôlé par les abonnements.

Ce flux de communication permet aux fonctions Lambda d'échanger des messages avec les entités suivantes :

- Appareils clients du groupe.
- Connecteurs dans le groupe.
- Autres fonctions Lambda du groupe.
- AWS IoT.
- Le service Device Shadow local.

Un abonnement définit la source d'un message, la cible d'un message et une rubrique (ou sujet) qui est utilisée pour acheminer des messages de la source vers la cible. Les messages publiés dans une fonction Lambda sont transmis au gestionnaire enregistré de la fonction. Les abonnements permettent une sécurité accrue et des interactions prévisibles. Pour plus d'informations, consultez the section called “Abonnements gérés dans le flux de travail de messagerie MQTT”.

> **Note**
>
> Lorsque le noyau est hors ligne, les fonctions Greengrass Lambda peuvent échanger des messages avec des appareils clients, des connecteurs, d'autres fonctions et des ombres locales, mais les messages à destination sont mis en file d'attente. AWS IoT Pour plus d'informations, consultez the section called “File d'attente de messages MQTT”.

Autres flux de communication

- Pour interagir avec les ressources locales des appareils et des volumes ainsi qu'avec les modèles d'apprentissage automatique sur un appareil principal, les fonctions Greengrass Lambda utilisent des interfaces de système d'exploitation spécifiques à la plate-forme. Par exemple, vous pouvez utiliser la méthode `open` dans le module os pour les fonctions Python. Pour qu'une fonction ait l'autorisation d'accéder à une ressource, elle doit être affiliée à la ressource et disposer de l'autorisation `read-only` ou `read-write`. Pour plus d'informations, notamment sur la disponibilité des versions AWS IoT Greengrass principales, consultez Accès aux ressources locales etthe section called “Accès aux ressources d'apprentissage automatique à partir du code de fonction Lambda”.

  > **Note**
  >
  > Si vous exécutez votre fonction Lambda sans conteneurisation, vous ne pouvez pas utiliser les ressources de périphérique et de volume locales connectées et vous devez accéder directement à ces ressources.

- Les fonctions Lambda peuvent utiliser le `Lambda` client du SDK AWS IoT Greengrass principal pour appeler d'autres fonctions Lambda du groupe Greengrass.
- Les fonctions Lambda peuvent utiliser le AWS SDK pour communiquer avec les services. AWS Pour plus d'informations, consultez la section AWSSDK.
- Les fonctions Lambda peuvent utiliser des interfaces tierces pour communiquer avec des services cloud externes, comme les fonctions Lambda basées sur le cloud.

> **Note**
>
> Les fonctions Lambda de Greengrass ne peuvent pas communiquer avec d'autres services cloud AWS ou avec d'autres services lorsque le cœur est hors ligne.

Récupération de la rubrique (ou sujet) MQTT en entrée

AWS IoT Greengrassutilise des abonnements pour contrôler l'échange de messages MQTT entre les appareils clients, les fonctions Lambda et les connecteurs d'un groupe, ainsi qu'AWS IoTavec ou avec le service fantôme local. Les abonnements définissent la source d'un message, la cible d'un

message ainsi qu'une rubrique MQTT utilisée pour acheminer les messages. Lorsque la cible est une fonction Lambda, le gestionnaire de la fonction est invoqué lorsque la source publie un message. Pour plus d'informations, consultez the section called "Communication à l'aide de messages MQTT".

L'exemple suivant montre comment une fonction Lambda peut obtenir le sujet d'entrée à partir du `context` sujet transmis au gestionnaire. Pour cela, la fonction accède à la clé `subject` à partir de la hiérarchie de contexte (`context.client_context.custom['subject']`). L'exemple convertit également le message JSON entrant, puis publie la rubrique et le message analysés.

> **Note**
>
> Dans l'API AWS IoT Greengrass, la rubrique d'un abonnement est représentée par la propriété `subject`.

```
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
```

Pour tester la fonction, ajoutez-la à votre groupe à l'aide des paramètres de configuration par défaut. Ajoutez ensuite les abonnements suivants et déployez le groupe. Pour obtenir des instructions, veuillez consulter the section called “Module 3 (partie 1) : Fonctions Lambda surAWS IoT Greengrass”.

Source	Cible	Sujet
Service de rubriques	Fonction	`test/input_message`
Fonction	Service de rubriques	`test/topic_results`

Une fois le déploiement terminé, appelez la fonction.

1. Dans la AWS IoT console, ouvrez la page du client de test MQTT.
2. Abonnez-vous au `test/topic_results` sujet en sélectionnant l'onglet S'abonner à un sujet.
3. Publiez un message dans le `test/input_message` sujet en sélectionnant l'onglet Publier dans un sujet. Dans le cadre de cet exemple, vous devez inclure la propriété `test-key` dans le message JSON.

   ```
   {
     "test-key": "Some string value"
   ```

```
}
```

En cas de réussite, la fonction publie la rubrique en entrée et la chaîne de message dans la rubrique `test/topic_results`.

Configuration du cycle de vie pour les fonctions Greengrass Lambda

Le cycle de vie de la fonction Greengrass Lambda détermine le moment où une fonction démarre et la manière dont elle crée et utilise les conteneurs. Le cycle de vie détermine également comment les variables et la logique de prétraitement se trouvant à l'extérieur du gestionnaire de fonctions sont conservées.

AWS IoT Greengrass prend en charge les cycles de vie à la demande (par défaut) ou à longue durée de vie :

- Les fonctions à la demande démarrent lorsqu'elles sont appelées et s'arrêtent lorsqu'il n'y a plus de tâche à exécuter. Un appel de la fonction crée un conteneur séparé (ou sandbox) pour traiter les appels, sauf si un conteneur existant peut être réutilisé. Les données qui sont envoyées à la fonction peuvent être extraites par n'importe quel conteneur.

  Plusieurs appels d'une fonction à la demande peuvent être exécutés en parallèle.

  Les variables et la logique de prétraitement définies à l'extérieur du gestionnaire de fonctions ne sont pas conservées quand de nouveaux conteneurs sont créés.

- Les fonctions à longue durée de vie (ou épinglées) démarrent automatiquement lorsque le AWS IoT Greengrass noyau démarre et s'exécutent dans un seul conteneur. Toutes les données qui sont envoyées à la fonction sont extraites par le même conteneur.

  Les appels sont mis en file d'attente jusqu'à ce que les appels précédents soient exécutées.

  Les variables et la logique de prétraitement définies à l'extérieur du gestionnaire de fonctions sont conservées pour chaque appel du gestionnaire.

  Les fonctions Lambda à longue durée de vie sont utiles lorsque vous devez commencer à travailler sans aucune intervention initiale. Par exemple, une fonction à longue durée de vie peut se charger

et commencer à traiter un modèle ML pour qu'il soit prêt lorsque la fonction commencera à recevoir des données des appareils.

> **Note**
>
> N'oubliez pas que les fonctions à longue durée de vie ont des délais d'attente associés aux appels de leur gestionnaire. Si vous souhaitez exécuter indéfiniment du code d'exécution, vous devez démarrer celui-ci à l'extérieur du gestionnaire. Assurez-vous qu'il n'y a pas de code de blocage à l'extérieur du gestionnaire qui peut empêcher la fonction de terminer son initialisation.
> Ces fonctions s'exécutent à moins que le cœur ne s'arrête (par exemple, lors d'un déploiement de groupe ou du redémarrage d'un appareil) ou que la fonction n'entre dans un état d'erreur (tel qu'un délai d'expiration du gestionnaire, une exception non détectée ou lorsqu'elle dépasse ses limites de mémoire).

Pour plus d'informations sur la réutilisation des conteneurs, consultez la section Comprendre la réutilisation des conteneurs AWS Lambda dans le blog AWS Compute.

Exécutables Lambda

Cette fonctionnalité est disponible pour AWS IoT Greengrass Core v1.6 et versions ultérieures.

Un exécutable Lambda est un type de fonction Greengrass Lambda que vous pouvez utiliser pour exécuter du code binaire dans l'environnement principal. Il vous permet d'exécuter des fonctionnalités spécifiques à l'appareil de manière native et de bénéficier de l'encombrement réduit du code compilé. Les exécutables Lambda peuvent être invoqués par des événements, invoquer d'autres fonctions et accéder aux ressources locales.

Les exécutables Lambda supportent uniquement le type d'encodage binaire (pas JSON), mais sinon vous pouvez les gérer dans votre groupe Greengrass et les déployer comme les autres fonctions Lambda de Greengrass. Cependant, le processus de création d'exécutables Lambda est différent de celui de création de fonctions Lambda Python, Java et Node.js :

- Vous ne pouvez pas utiliser la AWS Lambda console pour créer (ou gérer) un exécutable Lambda. Vous pouvez créer un exécutable Lambda uniquement à l'aide de l'AWS LambdaAPI.
- Vous téléchargez le code de fonction AWS Lambda sous forme d'exécutable compilé qui inclut le SDK AWS IoT Greengrass Core pour C.

- Vous spécifiez le nom de l'exécutable en tant que gestionnaire de fonctions.

Les exécutables Lambda doivent implémenter certains appels et modèles de programmation dans leur code de fonction. Par exemple, la méthode `main` doit :

- Appeler `gg_global_init` pour initialiser des variables globales internes Greengrass. Cette fonction doit être appelée avant la création des threads et avant l'appel à toute autre fonction du kit SDK AWS IoT Greengrass Core.
- Appelez `gg_runtime_start` pour enregistrer le gestionnaire de fonctions auprès du moteur d'exécution Greengrass Lambda. Cette fonction doit être appelée lors de l'initialisation. L'appel à cette fonction entraîne l'utilisation du thread actuel par l'environnement d'exécution. Le paramètre facultatif `GG_RT_OPT_ASYNC` indique à cette fonction de ne pas se bloquer, mais de créer un nouveau thread pour l'environnement d'exécution. Cette fonction utilise un gestionnaire `SIGTERM`.

L'extrait de code suivant est la `main` méthode tirée de l'exemple de code simple_handler.c sur. 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;
}
```

Pour plus d'informations sur les exigences, les contraintes et les autres détails de mise en œuvre, consultez le SDK AWS IoT Greengrass Core pour C.

Création d'un exécutable Lambda

Après avoir compilé votre code avec le SDK, utilisez l'AWS LambdaAPI pour créer une fonction Lambda et téléchargez votre exécutable compilé.

> **Note**
>
> Votre fonction doit être compilée avec un compilateur compatible avec C89.

L'exemple suivant utilise la commande create-function CLI pour créer un exécutable Lambda. La commande spécifie :

- Le nom de l'exécutable pour le gestionnaire. Il doit s'agir du nom exact de votre exécutable compilé.
- Le chemin d'accès au fichier `.zip` qui contient l'exécutable compilé.
- `arn:aws:greengrass:::runtime/function/executable` pour l'environnement d'exécution. Il s'agit de l'environnement d'exécution de tous les exécutables Lambda.

> **Note**
>
> En effet`role`, vous pouvez spécifier l'ARN de n'importe quel rôle d'exécution Lambda. AWS IoT Greengrassn'utilise pas ce rôle, mais le paramètre est obligatoire pour créer la fonction. Pour plus d'informations sur les rôles d'exécution Lambda, consultez le modèle AWS Lambda d'autorisations dans le Guide du AWS Lambdadéveloppeur.

```
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
```

Ensuite, utilisez l'API AWS Lambda pour publier une version et créer un alias.

- Utilisez publish-version pour publier une version de fonction.

```
aws lambda publish-version \
--function-name function-name \
--region aws-region
```

- Utilisez create-alias pour créer un alias pointant vers la version que vous venez de publier. Nous vous recommandons de référencer les fonctions Lambda par alias lorsque vous les ajoutez à un groupe Greengrass.

```
aws lambda create-alias \
--function-name function-name \
--name alias-name \
--function-version version-number \
--region aws-region
```

> **Note**
>
> La AWS Lambda console n'affiche pas les exécutables Lambda. Pour mettre à jour le code de fonction, vous devez utiliser l'API AWS Lambda.

Ajoutez ensuite l'exécutable Lambda à un groupe Greengrass, configurez-le pour accepter les données d'entrée binaires dans ses paramètres spécifiques au groupe, puis déployez le groupe. Vous pouvez effectuer cette opération dans la console AWS IoT Greengrass ou à l'aide de l'API AWS IoT Greengrass.

Exécution de AWS IoT Greengrass dans un conteneur Docker

AWS IoT Greengrass peut être configuré pour s'exécuter dans un conteneur Docker.

Vous pouvez télécharger un Dockerfile via Amazon surCloudFront lequel le logicielAWS IoT Greengrass Core et ses dépendances sont installés. Pour modifier l'image Docker afin qu'elle s'exécute sur différentes architectures de plateforme ou pour réduire la taille de l'image Docker, consultez le fichier `README` dans le package de téléchargement Docker.

Pour vous aider à commencer à expérimenter avec AWS IoT Greengrass, AWS fournit également des images Docker prédéfinies dans lesquelles sont installés le logiciel AWS IoT Greengrass Core et les dépendances. Vous pouvez télécharger une image depuis Docker Hub ou Amazon Elastic Container Registry (Amazon ECR). Ces images prédéfinies utilisent des images de base Amazon Linux 2 (x86_64) et Alpine Linux (x86_64, Armv7l ou AArch64).

> **Important**
>
> Le 30 juin 2022, la maintenance des images Docker v1.x du logicielAWS IoT Greengrass Core publiées sur Amazon Elastic Container Registry (Amazon ECR) et Docker Hub aAWS IoT Greengrass pris fin. Vous pouvez continuer à télécharger ces images Docker depuis Amazon ECR et Docker Hub jusqu'au 30 juin 2023, soit 1 an après la fin de la maintenance. Toutefois, les images Docker du logicielAWS IoT Greengrass Core v1.x ne reçoivent plus de correctifs de sécurité ni de corrections de bogues après la fin de la maintenance le 30 juin 2022. Si vous exécutez une charge de travail de production qui dépend de ces images Docker, nous vous recommandons de créer vos propres images Docker à l'aide des DockerfilesAWS IoT Greengrass fournis. Pour plus d'informations, veuillez consulter Logiciel AWS IoT Greengrass Docker.

Cette rubrique explique comment télécharger l'imageAWS IoT Greengrass Docker depuis Amazon ECR et l'exécuter sur une plateforme Windows, macOS ou Linux (x86_64). Cette rubrique comporte les étapes suivantes :

1. Obtenir l'image deAWS IoT Greengrass conteneur auprès d'Amazon ECR
2. Créer et configurer le groupe et le noyau Greengrass
3. Exécuter AWS IoT Greengrass localement
4. Configurer la conteneurisation « Aucun conteneur » pour le groupe
5. Déployez des fonctions Lambda dans le conteneur Docker
6. (Facultatif) Déployez des appareils clients qui interagissent avec Greengrass dans le conteneur Docker

Les fonctions suivantes ne sont pas prises en charge lorsque vous exécutez AWS IoT Greengrass dans un conteneur Docker :

- Connecteurs qui s'exécutent en mode Greengrass container (Conteneur Greengrass). Pour exécuter un connecteur dans un conteneur Docker, le connecteur doit s'exécuter en mode No container (Aucun conteneur). Pour rechercher des connecteurs prenant en charge le mode No container (Aucun conteneur) veuillez consulter the section called “AWS- connecteurs Greengrass fournis”. Certains de ces connecteurs ont un paramètre de mode d'isolement que vous devez définir sur Aucun conteneur.

- Ressources de volumes et d'appareils locales. Vos fonctions Lambda définies par l'utilisateur qui s'exécutent dans le conteneur Docker doivent accéder directement aux appareils et aux volumes situés sur le noyau.

Ces fonctionnalités ne sont pas prises en charge lorsque l'environnement d'exécution Lambda du groupe Greengrass est défini sur Aucun conteneur, ce qui est requis pour fonctionnerAWS IoT Greengrass dans un conteneur Docker.

Prérequis

Avant de commencer ce didacticiel, vous devez effectuer les opérations suivantes.

- Vous devez installer les logiciels et les versions suivants sur votre ordinateur hôte en fonction de la versionAWS Command Line Interface (AWS CLI) que vous choisissez.

  AWS CLI version 2

  - Docker version 18.09 ou ultérieure. Les versions antérieures peuvent également fonctionner, mais nous recommandons la version 18.09 ou ultérieure.
  - AWS CLI2.0.0 0 0
    - Pour installer laAWS CLI version 2, consultez la section Installation de laAWS CLI version 2.
    - Pour configurer leAWS CLI, reportez-vous à la section Configuration duAWS CLI.

    > **Note**
    >
    > Pour effectuer la mise à niveau vers uneAWS CLI version 2 ultérieure sur un ordinateur Windows, vous devez répéter le processus d'installation MSI.

  AWS CLI version 1

  - Docker version 18.09 ou ultérieure. Les versions antérieures peuvent également fonctionner, mais nous recommandons la version 18.09 ou ultérieure.
  - Python ultérieure.
  - pip version 18.1 ou suivante.
  - AWS CLIversion 1.17.10 ou ultérieure
    - Pour installer laAWS CLI version 1, consultez la section Installation de laAWS CLI version 1.

- Pour configurer leAWS CLI, reportez-vous à la section Configuration duAWS CLI.
- Pour effectuer une mise à niveau vers laAWS CLI dernière 1, exécutez la commande suivante :

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

> **Note**
>
> Si vous utilisez l'installation MSI de laAWS CLI version 1 sous Windows, tenez compte des points suivants :
>
> - Si l'installation de laAWS CLI version 1 ne parvient pas à installer botocore, essayez d'utiliser l'installation Python et pip.
> - Pour effectuer la mise à niveau vers uneAWS CLI version 1 ultérieure, vous devez répéter le processus d'installation MSI.

- Pour accéder aux ressources Amazon Elastic Container Registry (Amazon ECR), vous devez accorder l'autorisation suivante.
  - Amazon ECR exige que les utilisateurs aient accordé l'`ecr:GetAuthorizationToken`autorisation via une politiqueAWS Identity and Access Management (IAM) avant qu'ils puissent s'authentifier auprès d'un référentiel et envoyer ou récupérer des images à partir d'un référentiel Amazon ECR. Pour plus d'informations, consultez les exemples de règles relatives aux référentiels Amazon ECR et l'accès à un référentiel Amazon ECR dans le guide de l'utilisateur d'Amazon Elastic Container Registry.

Étape 1 : Obtenir l'image deAWS IoT Greengrass conteneur auprès d'Amazon ECR

AWS fournit des images Docker dans lesquelles le logiciel AWS IoT Greengrass Core est installé.

> **Warning**
>
> À partir de la version 1.11.6 du logicielAWS IoT Greengrass Core, les images Greengrass Docker n'incluent plus Python 2.7, car Python 2.7 a atteint end-of-life en 2020 et ne reçoit plus de mises à jour de sécurité. Si vous choisissez d'effectuer la mise à jour vers ces images Docker, nous vous recommandons de vérifier que vos applications fonctionnent avec les nouvelles images Docker avant de déployer les mises à jour sur les appareils de production.

Si vous avez besoin de Python 2.7 pour votre application qui utilise une image Greengrass Docker, vous pouvez modifier le Greengrass Dockerfile pour inclure Python 2.7 pour votre application.

Pour suivre les étapes expliquant comment extraire l'`latest`image d'Amazon ECR, choisissez votre système d'exploitation :

Extraire l'image de conteneur (Linux)

Exécutez les commandes suivantes dans le terminal de votre ordinateur.

1. Connectez-vous auAWS IoT Greengrass registre dans 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
   ```

   En cas de succès, la sortie imprime `Login Succeeded`.

2. Récupérez l'image de conteneur AWS IoT Greengrass.

   ```
   docker pull 216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
   ```

   > **Note**
   >
   > L'image `latest` contient la dernière version stable du logiciel AWS IoT Greengrass Core installée sur une image de base Amazon Linux 2. Vous pouvez également extraire d'autres images du référentiel. Pour trouver toutes les images disponibles, consultez la page Tags (Balises) sur Docker Hub ou utilisez la commande aws ecr list-images. Par exemple :
   >
   > ```
   > aws ecr list-images --region us-west-2 --registry-id 216483018798 --
   > repository-name aws-iot-greengrass
   > ```

3. Activez les protections symlink and hardlink. Si vous testez l'exécution d'AWS IoT Greengrass dans un conteneur, vous pouvez activer les paramètres pour le démarrage actuel uniquement.

> **Note**
>
> Vous devrez peut-être utiliser sudo pour exécuter ces commandes.

- Pour activer les paramètres pour le démarrage en cours uniquement :

```
echo 1 > /proc/sys/fs/protected_hardlinks
echo 1 > /proc/sys/fs/protected_symlinks
```

- Pour activer les paramètres afin qu'ils persistent entre les redémarrages :

```
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. Activez le réacheminement réseau IPv4, qui est nécessaire pour que le déploiement cloud d'AWS IoT Greengrass et les communications MQTT fonctionnent sur Linux. Dans le fichier `/etc/sysctl.conf`, définissez `net.ipv4.ip_forward` sur 1, puis rechargez `sysctls`.

```
sudo nano /etc/sysctl.conf
# set this net.ipv4.ip_forward = 1
sudo sysctl -p
```

> **Note**
>
> Vous pouvez utiliser l'éditeur de votre choix à la place de nano.

Extraire l'image de conteneur (macOS)

Exécutez les commandes suivantes dans le terminal de votre ordinateur.

1. Connectez-vous auAWS IoT Greengrass registre dans 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
```

En cas de succès, la sortie imprime `Login Succeeded`.

2. Récupérez l'image de conteneur AWS IoT Greengrass.

```
docker pull 216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
```

> **Note**
>
> L'image `latest` contient la dernière version stable du logiciel AWS IoT Greengrass Core installée sur une image de base Amazon Linux 2. Vous pouvez également extraire d'autres images du référentiel. Pour trouver toutes les images disponibles, consultez la page Tags (Balises) sur Docker Hub ou utilisez la commande aws ecr list-images. Par exemple :
>
> ```
> aws ecr list-images --region us-west-2 --registry-id 216483018798 --
> repository-name aws-iot-greengrass
> ```

Extraire l'image de conteneur (Windows)

Dans une invite de commande, exécutez les commandes suivantes. Pour pouvoir utiliser des commandes Docker sur Windows, Docker Desktop doit être en cours d'exécution.

1. Connectez-vous auAWS IoT Greengrass registre dans 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
```

En cas de succès, la sortie imprime `Login Succeeded`.

2. Récupérez l'image de conteneur AWS IoT Greengrass.

```
docker pull 216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
```

> **Note**
>
> L'image `latest` contient la dernière version stable du logiciel AWS IoT Greengrass Core installée sur une image de base Amazon Linux 2. Vous pouvez également extraire

d'autres images du référentiel. Pour trouver toutes les images disponibles, consultez la page Tags (Balises) sur Docker Hub ou utilisez la commande aws ecr list-images. Par exemple :

```
aws ecr list-images --region us-west-2 --registry-id 216483018798 --
repository-name aws-iot-greengrass
```

Étape 2 : Créer et configurer le groupe et le noyau Greengrass

L'image Docker comprend le logiciel AWS IoT Greengrass Core installé, mais vous devez créer un noyau et un groupe Greengrass. Cela inclut le téléchargement des certificats et du fichier de configuration du noyau.

- Suivez les étapes de the section called “Module 2 : Installation deAWS IoT GreengrassLogiciel Core”. Ignorez les étapes de téléchargement et d'exécution du logicielAWS IoT Greengrass Core. Le logiciel et ses dépendances d'exécution sont déjà configurées dans l'image Docker.

Étape 3 : Exécuter AWS IoT Greengrass localement

Une votre groupe configuré, vous êtes prêt à configurer et démarrer le noyau. Pour les étapes qui montrent comment procéder, choisissez votre système d'exploitation :

Exécuter Greengrass localement (Linux)

Exécutez les commandes suivantes dans le terminal de votre ordinateur.

1. Créez un dossier pour les ressources de sécurité de l'appareil et déplacez le certificat et les clés dans ce dossier. Exécutez les commandes suivantes. Remplacez *path-to-security-files*par le chemin d'accès aux ressources de sécurité et remplacez *certificateId par l'ID* de certificat figurant dans les noms de fichiers.

   ```
   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. Créez un dossier pour la configuration de l'appareil et déplacez le fichier de configurationAWS IoT Greengrass Core vers ce dossier. Exécutez les commandes suivantes. Remplacez *path-to-config-file*par le chemin d'accès au fichier de configuration.

   ```
   mkdir /tmp/config
   mv path-to-config-file/config.json /tmp/config
   ```

3. Démarrez AWS IoT Greengrass et créez un montage lié des certificats et du fichier de configuration dans le conteneur Docker.

   Remplacez `/tmp` par le chemin d'accès où vous avez décompressé vos certificats et le fichier de configuration.

   ```
   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
   ```

   La sortie doit se présenter comme cet exemple :

   ```
   Setting up greengrass daemon
   Validating hardlink/softlink protection
   Waiting for up to 30s for Daemon to start

   Greengrass successfully started with PID: 10
   ```

Exécuter Greengrass localement (macOS)

Exécutez les commandes suivantes dans le terminal de votre ordinateur.

1. Créez un dossier pour les ressources de sécurité de l'appareil et déplacez le certificat et les clés dans ce dossier. Exécutez les commandes suivantes. Remplacez *path-to-security-files*par le chemin d'accès aux ressources de sécurité et remplacez *certificateId par l'ID* de certificat figurant dans les noms de fichiers.

   ```
   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. Créez un dossier pour la configuration de l'appareil et déplacez le fichier de configurationAWS IoT Greengrass Core vers ce dossier. Exécutez les commandes suivantes. Remplacez *path-to-config-file*par le chemin d'accès au fichier de configuration.

   ```
   mkdir /tmp/config
   mv path-to-config-file/config.json /tmp/config
   ```

3. Démarrez AWS IoT Greengrass et créez un montage lié des certificats et du fichier de configuration dans le conteneur Docker.

   Remplacez `/tmp` par le chemin d'accès où vous avez décompressé vos certificats et le fichier de configuration.

   ```
   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
   ```

   La sortie doit se présenter comme cet exemple :

   ```
   Setting up greengrass daemon
   Validating hardlink/softlink protection
   Waiting for up to 30s for Daemon to start

   Greengrass successfully started with PID: 10
   ```

Exécuter Greengrass localement (Windows)

1. Créez un dossier pour les ressources de sécurité de l'appareil et déplacez le certificat et les clés dans ce dossier. Dans une invite de commande, exécutez les commandes suivantes. Remplacez *path-to-security-files*par le chemin d'accès aux ressources de sécurité et remplacez *certificateId par l'ID* de certificat figurant dans les noms de fichiers.

   ```
   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. Créez un dossier pour la configuration de l'appareil et déplacez le fichier de configurationAWS IoT Greengrass Core vers ce dossier. Dans une invite de commande, exécutez les commandes suivantes. Remplacez *path-to-config-file*par le chemin d'accès au fichier de configuration.

```
mkdir C:\Users\%USERNAME%\Downloads\config
move path-to-config-file\config.json C:\Users\%USERNAME%\Downloads\config
```

3. Démarrez AWS IoT Greengrass et créez un montage lié des certificats et du fichier de configuration dans le conteneur Docker. Dans votre invite de commande, exécutez les commandes suivantes.

```
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
```

Lorsque Docker vous invite à partager votre lecteur `C:\` avec le démon Docker, laissez-le créer un montage lié du répertoire `C:\` dans le conteneur Docker. Pour plus d'informations, consultez la section Disques partagés dans la documentation Docker.

La sortie doit se présenter comme cet exemple :

```
Setting up greengrass daemon
Validating hardlink/softlink protection
Waiting for up to 30s for Daemon to start

Greengrass successfully started with PID: 10
```

> **Note**
>
> Si le conteneur n'ouvre pas le shell et se ferme immédiatement, vous pouvez résoudre le problème en procédant à un montage lié des journaux du runtime Greengrass lorsque vous démarrez l'image. Pour plus d'informations, veuillez consulter the section called “Pour conserver les journaux d'exécution Greengrass en dehors du conteneur Docker”.

Étape 4 : Configurer la conteneurisation « Aucun conteneur » pour le groupe Greengrass

Lorsque vous exécutezAWS IoT Greengrass dans un conteneur Docker, toutes les fonctions Lambda doivent s'exécuter sans conteneurisation. Dans cet étape, vous définissez la conteneurisation par défaut du groupe sur Aucun conteneur. Vous devez effectuer cette opération avant le premier déploiement du groupe.

1. Dans le volet de navigation de laAWS IoT console, sous Gérer, développez les appareils Greengrass, puis choisissez Groupes (V1).
2. Choisissez le groupe dont vous voulez modifier les paramètres.
3. Choisissez l'onglet Fonctions Lambda.
4. Dans Environnement d'exécution des fonctions Lambda par défaut, choisissez Modifier.
5. Dans l'environnement d'exécution de la fonction Lambda par défaut Modifier, sous Conteneurisation de la fonction Lambda par défaut, modifiez les paramètres de conteneurisation.
6. Choisissez Save (Enregistrer).

Les modifications prennent effet lorsque le groupe est déployé.

Pour plus d'informations, veuillez consulter the section called “Configuration de la conteneurisation par défaut pour les fonctions Lambda dans un groupe”.

> **Note**
>
> Par défaut, les fonctions Lambda utilisent le paramètre de conteneurisation des groupes. Si vous remplacez le paramètre Aucun conteneur pour une fonction LambdaAWS IoT Greengrass exécutée dans un conteneur Docker, le déploiement échoue.

Étape 5 : Déployer des fonctions Lambda dans le conteneurAWS IoT Greengrass Docker

Vous pouvez déployer des fonctions Lambda durables dans le conteneur Greengrass Docker.

- Suivez les étapes décritesthe section called “Module 3 (partie 1) : Fonctions Lambda surAWS IoT Greengrass” pour déployer une fonction Lambda Hello World de longue durée sur le conteneur.

Étape 6 : (Facultatif) Déployez des appareils clients qui interagissent avec Greengrass s'exécutant dans le conteneur Docker

Vous pouvez également déployer des appareils clients avec lesquels il interagitAWS IoT Greengrass lorsqu'il s'exécute dans un conteneur Docker.

- Suivez les étapes décrites dans lathe section called “Module 4 : Interaction avec les appareils clients dans unAWS IoT Greengrassgroupe” section pour déployer des appareils clients qui se connectent au noyau et envoient des messages MQTT.

Arrêt du conteneur Docker AWS IoT Greengrass

Pour arrêter le conteneur Docker AWS IoT Greengrass, appuyez sur Ctrl+C dans votre terminal ou dans la ligne de commande. Cette action envoie`SIGTERM` au processus démon Greengrass pour détruire le processus démon Greengrass et tous les processus Lambda qui ont été démarrés par le processus démon. Le conteneur Docker est initialisé avec le processus `/dev/init` défini comme PID 1, qui permet d'éliminer tous les processus zombies restants. Pour plus d'informations, consultez le guide de référence d'exécution Docker.

Résolution des problèmes liés à AWS IoT Greengrass dans un conteneur Docker

Aidez-vous des informations suivantes pour résoudre les problèmes que vous êtes susceptible de rencontrer lors de l'exécution d'AWS IoT Greengrass dans un conteneur Docker.

Erreur : Impossible d'effectuer une connexion interactive à partir d'un appareil autre que TTY.

Solution : cette erreur peut se produire lorsque vous exécutez la commande `aws ecr get-login-password`. Assurez-vous d'avoir installé la dernièreAWS CLI version 2 ou la version 1. Nous vous recommandons d'AWS CLIutiliser la 2 Pour plus d'informations, consultez Installation d'AWS CLI dans le Guide de l'utilisateur AWS Command Line Interface.

Erreur : options inconnues : -no-include-email.

Solution : cette erreur peut se produire lorsque vous exécutez la commande `aws ecr get-login`. Assurez-vous que vous disposez de la dernière version de l'interface de ligne de commande AWS CLI installée (par exemple, exécutez : `pip install awscli --upgrade --user`). Si vous utilisez Windows et avez installé l'interface de ligne de commande à l'aide du programme d'installation MSI, vous devez répéter le processus d'installation. Pour plus d'informations, consultez la section Installation deAWS Command Line Interface sur Microsoft Windows dans le Guide deAWS Command Line Interface l'utilisateur.

Avertissement : IPv4 est désactivé. La mise en réseau ne fonctionnera pas.

Solution : vous pouvez recevoir cet avertissement ou un message similaire lors de l'exécution d'AWS IoT Greengrass sur un ordinateur Linux. Activez le réacheminement réseau IPv4 comme décrit dans cette étape. Le déploiement cloud AWS IoT Greengrass et les communications MQTT ne fonctionnent pas lorsque le réacheminement IPv4 n'est pas activé. Pour plus d'informations, consultez Configurer les paramètres du noyau dans un espace de noms (sysctls) lors de l'exécution dans la documentation Docker.

Erreur : Un pare-feu bloque le partage de fichiers entre les fenêtres et les conteneurs.

Solution : vous pouvez recevoir cette erreur ou un message `Firewall Detected` lors de l'exécution de Docker sur un ordinateur Windows. Ce problème peut également survenir si vous êtes connecté à un réseau privé virtuel (VPN) et que vos paramètres réseau empêchent le montage du lecteur partagé. Dans ce cas, désactivez le VPN et réexécutez le conteneur Docker.

Erreur : une erreur s'est produite (AccessDeniedException) lors de l'appel de l'GetAuthorizationTokenopération : L'utilisateur : arn:aws:iam : ::user/ <account-id><user-name>n'est pas autorisé à exécuter : ecr : onGetAuthorizationToken resource : *

Cette erreur peut s'afficher lors de l'exécution de la`aws ecr get-login-password` commande si vous ne disposez pas des autorisations nécessaires pour accéder à un référentiel Amazon ECR. Pour plus d'informations, consultez les exemples de règles relatives aux référentiels Amazon ECR et l'accès à un référentiel Amazon ECR dans le guide de l'utilisateur Amazon ECR.

Pour l'aide à la résolution des problèmes AWS IoT Greengrass généraux, consultez Résolution des problèmes.

Débogage de AWS IoT Greengrass dans un conteneur Docker

Pour déboguer les problèmes avec un conteneur Docker, vous pouvez conserver les journaux d'exécution Greengrass ou attacher un shell interactif au conteneur Docker.

Pour conserver les journaux d'exécution Greengrass en dehors du conteneur Docker

Vous pouvez exécuter le conteneur Docker AWS IoT Greengrass après avoir procédé au montage lié du répertoire `/greengrass/ggc/var/log`. Les journaux sont conservés après la fermeture ou la suppression du conteneur.

Sur Linux ou macOS

1. Arrêtez les conteneurs Docker Greengrass en cours d'exécution sur l'hôte, puis exécutez la commande suivante dans un terminal. Elle procède au montage lié du répertoire `log` Greengrass et démarre l'image Docker.

   Remplacez `/tmp` par le chemin d'accès où vous avez décompressé vos certificats et le fichier de configuration.

   ```
   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
   ```

Vous pouvez alors vérifier vos journaux à l'emplacement `/tmp/log` sur votre hôte afin de voir ce qui s'est passé lors de l'exécution de Greengrass dans le conteneur Docker.

Sous Windows

Arrêtez les conteneurs Docker Greengrass en cours d'exécution sur l'hôte, puis exécutez la commande suivante dans une invite de commande. Elle procède au montage lié du répertoire `log` Greengrass et démarre l'image 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
```

Vous pouvez alors vérifier vos journaux à l'emplacement `C:/Users/%USERNAME%/Downloads/log` sur votre hôte afin de voir ce qui s'est passé lors de l'exécution de Greengrass dans le conteneur Docker.

Pour attacher un shell interactif au conteneur Docker

Vous pouvez attacher un shell interactif à un conteneur Docker AWS IoT Greengrass en cours d'exécution. Cela peut vous aider à étudier l'état du conteneur Docker Greengrass.

Sur Linux ou macOS

Pendant que le conteneur Docker Greengrass est en cours d'exécution, exécutez la commande suivante dans un autre terminal.

```
docker exec -it $(docker ps -a -q -f "name=aws-iot-greengrass") /bin/bash
```

Sous Windows

Pendant que le conteneur Docker Greengrass est en cours d'exécution, exécutez les commandes suivantes dans une invite de commande distincte.

```
docker ps -a -q -f "name=aws-iot-greengrass"
```

Remplacez *`gg-container-id`*par le`container_id` résultat de la commande précédente.

```
docker exec -it gg-container-id /bin/bash
```

Accédez à des ressources locales avec des fonctions et des connecteurs Lambda

Cette fonctionnalité est disponible pourAWS IoT Greengrass Core v1.3 et versions ultérieures.

Avec AWS IoT Greengrass, vous pouvez créer des fonctions AWS Lambda et configurer des connecteurs dans le cloud, puis les déployer vers les appareils principaux pour une exécution locale. Sur les cœurs Greengrass exécutant Linux, ces fonctions et connecteurs Lambda déployés localement peuvent accéder aux ressources locales présentes physiquement sur le périphérique principal Greengrass. Par exemple, pour communiquer avec des appareils connectés via Modbus ou CANbus, vous pouvez activer votre fonction Lambda pour accéder au port série du périphérique principal. Pour configurer un accès sécurisé aux ressources locales, vous devez garantir la sécurité de votre matériel physique et du système d'exploitation de votre appareil principal Greengrass.

Pour commencer à accéder aux ressources locales, consultez les didacticiels suivants :

- Comment configurer l'accès aux ressources locales à l'aide de l'interface de ligne de AWS commande
- Configuration de l'accès aux ressources locales à l'aide de l'AWS Management Console

Types de ressources pris en charge

Vous pouvez accéder à deux types de ressources locales : les ressources de volumes et les ressources d'appareils.

Ressources de volume

Fichiers ou répertoires sur le système de fichiers racine (sauf sous `/sys`, `/dev` ou `/var`). Il s'agit des licences suivantes :

- Dossiers ou fichiers utilisés pour lire ou écrire des informations via les fonctions Greengrass Lambda (par exemple,`/usr/lib/python2.x/site-packages/local`).
- Dossiers ou fichiers sous le système de fichiers /proc de l'hôte (par exemple, `/proc/net` ou `/proc/stat`). Pris en charge dans la version 1.6 ou ultérieure. Pour des prérequis supplémentaires, consultez the section called “Ressources de volume sous le répertoire /proc”.

> **Tip**
>
> Pour configurer les répertoires `/var`, `/var/run` et `/var/lib` en tant que ressources de volume, montez d'abord le répertoire dans un autre dossier, puis configurez le dossier en tant que ressource de volume.

Lorsque vous configurez des ressources de volume, vous devez spécifier un chemin source et un chemin de destination. Le chemin source est le chemin d'accès absolu de la ressource sur l'hôte. Le chemin absolu est le chemin absolu de la ressource dans l'environnement de nommage Lambda. Il s'agit du conteneur dans lequel s'exécute une fonction ou un connecteur Greengrass Lambda. Toute modification du chemin de destination est prise en compte dans le chemin source sur le système de fichiers hôte.

> **Note**
>
> Les fichiers du chemin de destination sont visibles uniquement dans l'espace de noms Lambda. Vous ne pouvez pas les voir dans un espace de noms Linux standard.

Ressources de l'appareil

Fichiers sous `/dev`. Seuls les périphériques de caractères ou les périphériques de stockage en mode bloc sous `/dev` sont autorisés pour les ressources de périphérique. Il s'agit des licences suivantes :

- Ports série utilisés pour communiquer avec les appareils connectés via des ports série (par exemple, `/dev/ttyS0`, `/dev/ttyS1`).
- USB utilisé pour connecter des périphériques USB (par exemple, `/dev/ttyUSB0` ou `/dev/bus/usb`).
- GPIO utilisés pour les capteurs et les actionneurs via GPIO (par exemple, `/dev/gpiomem`).
- GPU utilisés pour accélérer l'apprentissage automatique à l'aide de GPU embarqués (par exemple, `/dev/nvidia0`).
- Caméras utilisées pour capturer les images et les vidéos (par exemple, `/dev/video0`).

> **Note**
>
> `/dev/shm` est une exception. Il peut être configuré en tant que ressource de volume uniquement. Les ressources sous `/dev/shm` doivent recevoir l'autorisation `rw`.

AWS IoT Greengrass prend également en charge les types de ressources utilisés pour exécuter l'inférence de Machine Learning. Pour plus d'informations, veuillez consulter Exécuter l'inférence de Machine Learning.

Prérequis

Les exigences suivantes s'appliquent à la configuration d'un accès sécurisé aux ressources locales :

- Vous devez utiliserAWS IoT Greengrass Core Software v1.3 ou version ultérieure. Pour créer des ressources pour le répertoire /proc de l'hôte, vous devez utiliser la version 1.6 ou une version ultérieure.
- La ressource locale (y compris tous les pilotes et bibliothèques requis) doit être correctement installée sur l'appareil principal Greengrass et constamment disponible pendant l'utilisation.
- L'opération souhaitée de la ressource et l'accès à la ressource ne doivent pas nécessiter de privilèges racine.
- Seules les autorisations `read` ou `read and write` sont disponibles. Les fonctions Lambda ne peuvent pas effectuer d'opérations privilégiées sur les ressources.
- Vous devez fournir le chemin d'accès complet de la ressource locale sur le système d'exploitation de l'appareil Greengrass principal.
- Un ID ou un nom de ressource doit comporter 128 caractères au maximum et utiliser le modèle `[a-zA-Z0-9:_-]+`.

Ressources de volume sous le répertoire /proc

Les considérations suivantes s'appliquent aux ressources de volume se trouvant sous le répertoire/proc de l'hôte.

- Vous devez utiliserAWS IoT Greengrass Core Software v1.6 ou version ultérieure.
- Vous pouvez autoriser l'accès en lecture seule aux fonctions Lambda, mais pas l'accès en lecture-écriture. Ce niveau d'accès est géré par AWS IoT Greengrass.

- Vous pouvez également avoir besoin d'accorder des autorisations de groupe de système d'exploitation pour activer l'accès en lecture dans le système de fichiers. Par exemple, supposons que votre répertoire ou fichier source ait une autorisation 660, ce qui signifie que seul le propriétaire ou l'utilisateur du groupe dispose de l'accès en lecture (et en écriture). Dans ce cas, vous devez ajouter les autorisations du propriétaire du groupe de système d'exploitation à la ressource. Pour plus d'informations, veuillez consulter the section called “Autorisation d'accès fichier pour le propriétaire du groupe”.
- L'environnement hôte et l'espace de noms Lambda contiennent tous deux un répertoire /proc. Veillez donc à éviter les conflits de noms lorsque vous spécifiez le chemin de destination. Par exemple, si /proc est le chemin d'accès source, vous pouvez spécifier /host-proc comme chemin de destination (ou tout autre nom de chemin différent de « /proc »).

Autorisation d'accès fichier pour le propriétaire du groupe

Un processus de fonctionAWS IoT Greengrass Lambda s'exécute normalement sous la formeggc_user etggc_group. Vous pouvez toutefois accorder des autorisations d'accès aux fichiers supplémentaires au processus de la fonction Lambda dans la définition de la ressource locale, comme suit :

- Pour ajouter les autorisations du groupe Linux propriétaire de la ressource, utilisez leGroupOwnerSetting#AutoAddGroupOwner paramètre ou l'option Ajouter automatiquement les autorisations du système de fichiers du groupe de systèmes propriétaire de la ressource.
- Pour ajouter les autorisations d'un autre groupe Linux, utilisez leGroupOwnerSetting#GroupOwner paramètre ou l'option de console Spécifier un autre groupe de systèmes pour ajouter les autorisations du système de fichiers. La valeur GroupOwner est ignorée si GroupOwnerSetting#AutoAddGroupOwner a la valeur true.

Un processus de fonctionAWS IoT Greengrass Lambda hérite de toutes les autorisations de système de fichiers deggc_userggc_group, et du groupe Linux (s'il est ajouté). Pour que la fonction Lambda accède à une ressource, le processus de la fonction Lambda doit disposer des autorisations requises sur la ressource. Vous pouvez utiliser la commande chmod(1) pour modifier l'autorisation de la ressource, si nécessaire.

Consulter aussi

- Quotas de service pour les ressources dans le Référence générale d'Amazon Web Services

Comment configurer l'accès aux ressources locales à l'aide de l'interface de ligne de AWS commande

Cette fonctionnalité est disponible pour AWS IoT Greengrass Core v1.3 et versions ultérieures.

Pour utiliser une ressource locale, vous devez ajouter une définition de ressource à la définition de groupe déployée sur votre appareil principal Greengrass. La définition de groupe doit également contenir une définition de fonction Lambda dans laquelle vous accordez les autorisations d'accès à vos fonctions Lambda pour les ressources locales. Pour en savoir plus, notamment les exigences et contraintes, consultez Accédez à des ressources locales avec des fonctions et des connecteurs Lambda.

Ce didacticiel décrit le processus de création d'une ressource locale et de configuration de l'accès à celle-ci à l'aide de la AWS Command Line Interface (CLI). Pour suivre les étapes de ce didacticiel, vous devez avoir déjà créé un groupe Greengrass comme décrit dans Commencer avec AWS IoT Greengrass.

Pour obtenir un didacticiel qui utilise AWS Management Console, consultez Configuration de l'accès aux ressources locales à l'aide de l'AWS Management Console.

Création de ressources locales

Tout d'abord, utilisez la commande `CreateResourceDefinition` pour créer une définition de ressource qui spécifie les ressources auxquelles vous voulez accéder. Dans cet exemple, nous créons deux ressources, `TestDirectory` et `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": ""
                        }
                    }
                }
            }
        ]
    }
}'
```

Ressources : liste des objets `Resource` du groupe Greengrass. Un groupe Greengrass peut avoir jusqu'à 50 ressources.

Resource#Id : identifiant unique de la ressource. L'ID est utilisé pour se référer à une ressource dans la configuration de la fonction Lambda. Longueur maximale de 128 caractères. Modèle : [a-zA-Z0-9:_-]+.

Resource#Name : nom de la ressource. Le nom de la ressource s'affiche dans la console Greengrass. Longueur maximale de 128 caractères. Modèle : [a-zA-Z0-9:_-]+.

LocalDeviceResourceData# SourcePath : Le chemin absolu local de la ressource de l'appareil. Le chemin source d'une ressource d'appareil ne peut faire référence qu'à un appareil caractère ou un appareil bloc sous `/dev`.

LocalVolumeResourceData# SourcePath : Le chemin absolu local de la ressource de volume sur le périphérique principal de Greengrass. Cet emplacement est en dehors du conteneur dans lequel la fonction s'exécute. Le chemin source d'un type de ressource de volume ne peut pas commencer par `/sys`.

LocalVolumeResourceData# DestinationPath : Le chemin absolu de la ressource volumique dans l'environnement Lambda. Cet emplacement est dans le conteneur dans lequel la fonction s'exécute.

GroupOwnerSetting: vous permet de configurer des privilèges de groupe supplémentaires pour le processus Lambda. Ce champ est facultatif. Pour plus d'informations, consultez Autorisation d'accès fichier pour le propriétaire du groupe.

GroupOwnerSetting# AutoAddGroupOwner : Si vrai, Greengrass ajoute automatiquement le propriétaire du groupe de système d'exploitation Linux spécifié pour la ressource aux privilèges du processus Lambda. Ainsi, le processus Lambda a les autorisations d'accès aux fichiers du groupe Linux ajouté.

GroupOwnerSetting# GroupOwner : Spécifie le nom du groupe de systèmes d'exploitation Linux dont les privilèges sont ajoutés au processus Lambda. Ce champ est facultatif.

Un ARN de version de définition de ressource est renvoyé par `CreateResourceDefinition`. L'ARN doit être utilisé lors de la mise à jour d'une définition de groupe.

```
{
    "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"
}
```

Création de la fonction Greengrass

Une fois les ressources créées, utilisez la commande `CreateFunctionDefinition` pour créer la fonction Greengrass et lui accorder l'accès à la ressource :

```
aws greengrass create-function-definition --cli-input-json '{
    "Name": "MyFunctionDefinition",
    "InitialVersion": {
        "Functions": [
            {
                "Id": "greengrassLraTest",
                "FunctionArn": "arn:aws:lambda:us-
west-2:012345678901:function:lraTest:1",
                "FunctionConfiguration": {
```

```
                    "Pinned": false,
                    "MemorySize": 16384,
                    "Timeout": 30,
                    "Environment": {
                        "ResourceAccessPolicies": [
                            {
                                "ResourceId": "data-volume",
                                "Permission": "rw"
                            },
                            {
                                "ResourceId": "data-device",
                                "Permission": "ro"
                            }
                        ],
                        "AccessSysfs": true
                    }
                }
            }
        ]
    }
}'
```

ResourceAccessPolicies: contient le `resourceId` et `permission` qui permet à la fonction Lambda d'accéder à la ressource. Une fonction Lambda peut accéder à un maximum de 20 ressources.

ResourceAccessPolicy#Permission : Spécifie les autorisations dont dispose la fonction Lambda sur la ressource. Les options disponibles sont `rw` (lecture/écriture) ou `ro` (lecture seule).

AccessSysfs: Si c'est vrai, le processus Lambda peut avoir un accès en lecture au `/sys` dossier sur le périphérique principal de Greengrass. Ceci est utilisé dans les cas où la fonction Greengrass Lambda doit lire les informations de l'appareil. `/sys`

Là encore, `CreateFunctionDefinition` renvoie un ARN de version de définition de fonction. L'ARN doit être utilisé dans votre version de définition de groupe.

```
{
    "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"
}
```

Ajoutez la fonction Lambda au groupe

Enfin, utilisez `CreateGroupVersion` pour ajouter la fonction au groupe. Par exemple :

```
aws greengrass create-group-version --group-id "b36a3aeb-3243-47ff-9fa4-7e8d98cd3cf5" \
--resource-definition-version-arn "arn:aws:greengrass:us-west-2:123456789012:/
greengrass/definition/resources/db6bf40b-29d3-4c4e-9574-21ab7d74316c/versions/31d0010f-
e19a-4c4c-8098-68b79906fb87" \
--core-definition-version-arn "arn:aws:greengrass:us-west-2:123456789012:/
greengrass/definition/cores/adbf3475-f6f3-48e1-84d6-502f02729067/
versions/297c419a-9deb-46dd-8ccc-341fc670138b" \
--function-definition-version-arn "arn:aws:greengrass:us-west-2:123456789012:/
greengrass/definition/functions/d1123830-da38-4c4c-a4b7-e92eec7b6d3e/versions/a2e90400-
caae-4ffd-b23a-db1892a33c78" \
--subscription-definition-version-arn "arn:aws:greengrass:us-west-2:123456789012:/
greengrass/definition/subscriptions/7a8ef3d8-1de3-426c-9554-5b55a32fbcb6/
versions/470c858c-7eb3-4abd-9d48-230236bfbf6a"
```

> **Note**
>
> Pour savoir comment obtenir l'ID de groupe à utiliser avec ces commandes, veuillez consulter the section called “Obtention de l'ID de groupe”.

Une nouvelle version de groupe est retournée :

```
{
    "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"
}
```

Votre groupe Greengrass contient désormais la fonction LRatest Lambda qui a accès à deux ressources : et. TestDirectory TestCamera

Cet exemple de fonction Lambda, `lraTest.py`, écrite en Python, écrit sur la ressource de 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
```

Ces commandes sont fournies par l'API Greengrass pour créer et gérer des définitions de ressource et des versions de définitions de ressources :

- CreateResourceDefinition
- CreateResourceDefinitionVersion
- DeleteResourceDefinition

- GetResourceDefinition
- GetResourceDefinitionVersion
- ListResourceDefinitions
- ListResourceDefinitionVersions
- UpdateResourceDefinition

Résolution des problèmes

- Q : Pourquoi le déploiement de mon groupe Greengrass échoue-t-il avec une erreur similaire à :

  ```
  group config is invalid:
      ggc_user or [ggc_group root tty] don't have ro permission on the file: /dev/tty0
  ```

  R : Cette erreur indique que le processus Lambda n'a pas l'autorisation d'accéder à la ressource spécifiée. La solution est de modifier l'autorisation de fichier de la ressource afin que Lambda puisse y accéder. (Consultez Autorisation d'accès fichier pour le propriétaire du groupe pour plus de détails).

- Q : Quand je configure `/var/run` comme ressource de volume, pourquoi la fonction Lambda ne réussit-elle pas à démarrer avec un message d'erreur dans le 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 : le AWS IoT Greengrass noyau ne prend actuellement pas en charge la configuration de `/var` et `/var/run` en `/var/lib` tant que ressources de volume. Une solution de contournement consiste à monter d'abord `/var`, `/var/run` ou `/var/lib` dans un autre dossier, puis à configurer le dossier comme ressource de volume.

- Q : Quand je configure `/dev/shm` comme ressource de volume avec l'autorisation en lecture seule, pourquoi la fonction Lambda ne réussit-elle pas à démarrer avec un message d'erreur dans le 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` ne peut être configuré qu'en lecture/écriture. Modifiez l'autorisation de la ressource en `rw` pour résoudre le problème.

Configuration de l'accès aux ressources locales à l'aide de l'AWS Management Console

Cette fonction est disponible uniquement pourAWS IoT GreengrassCore v1.3 et versions ultérieures.

Vous pouvez configurer des fonctions Lambda pour accéder de façon sécurisée aux ressources locales sur l'appareil Greengrass principal hôte. Les ressources locales font référence aux bus et périphériques qui se trouvent physiquement sur l'hôte, ou aux volumes du système de fichiers sur le système d'exploitation hôte. Pour en savoir plus, notamment les exigences et contraintes, consultez Accédez à des ressources locales avec des fonctions et des connecteurs Lambda.

Ce didacticiel explique comment utiliserAWS Management Consolepour configurer l'accès aux ressources locales qui sont présentes sur unAWS IoT Greengrassappareil noyau. Il contient les étapes détaillées suivantes :

1. Créer un package de déploiement de fonction Lambda
2. Créer et publier une fonction Lambda
3. Ajouter la fonction Lambda au groupe
4. Ajouter une ressource locale au groupe
5. Ajouter des abonnements au groupe
6. Déployer le groupe

Pour obtenir un didacticiel qui utilise AWS Command Line Interface, consultez Comment configurer l'accès aux ressources locales à l'aide de l'interface de ligne de AWS commande.

Prérequis

Pour suivre ce didacticiel, vous devez disposer des éléments suivants :

- Un groupe Greengrass et un appareil principal (noyau) Greengrass (version 1.3 ou ultérieure). Pour savoir comment créer un groupe ou un service principal Greengrass, consultez Commencer avec AWS IoT Greengrass.
- Répertoires suivants créés sur l'appareil Greengrass principal :
  - /src/LRAtest
  - /dest/LRAtest

  Le propriétaire du groupe de ces répertoires doit disposer d'un accès en lecture et en écriture aux répertoires. Par exemple, pour accorder l'accès, vous pouvez utiliser la commande suivante :

  ```
  sudo chmod 0775 /src/LRAtest
  ```

Étape 1 : Créer un package de déploiement de fonction Lambda

Au cours de cette étape, vous créez le package de déploiement d'une fonction Lambda, qui est un fichier ZIP contenant le code et les dépendances de la fonction. Vous téléchargez également le kit SDK d'AWS IoT Greengrass Core à inclure dans le package en tant que dépendance.

1. Sur votre ordinateur, copiez le script Python suivant dans un fichier local nommé `lraTest.py`. Il s'agit de la logique d'application pour la fonction 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. À partir de laAWS IoT GreengrassKit SDK Corepage de téléchargement, téléchargez leAWS IoT GreengrassSDK Core pour Python sur votre ordinateur.
3. Décompressez le package téléchargé pour obtenir le kit SDK. Le kit SDK est représenté par le dossier `greengrasssdk`.
4. Compressez les éléments suivants dans un fichier nommé `lraTestLambda.zip` :

   - `lraTest.py`. Logique d'application.
   - `greengrasssdk`. Bibliothèque requise pour toutes les fonctions Lambda Python.

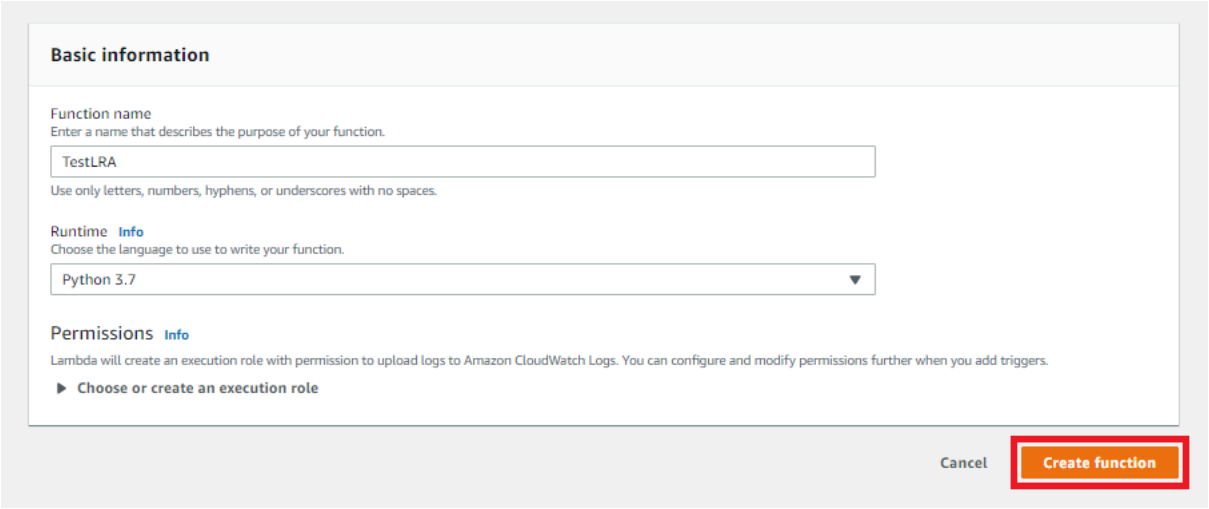
   `LelraTestLambda.zip`est le package de déploiement de votre fonction Lambda. Vous êtes maintenant prêt à créer une fonction Lambda et à télécharger le package de déploiement.

Étape 2 : Créer et publier une fonction Lambda

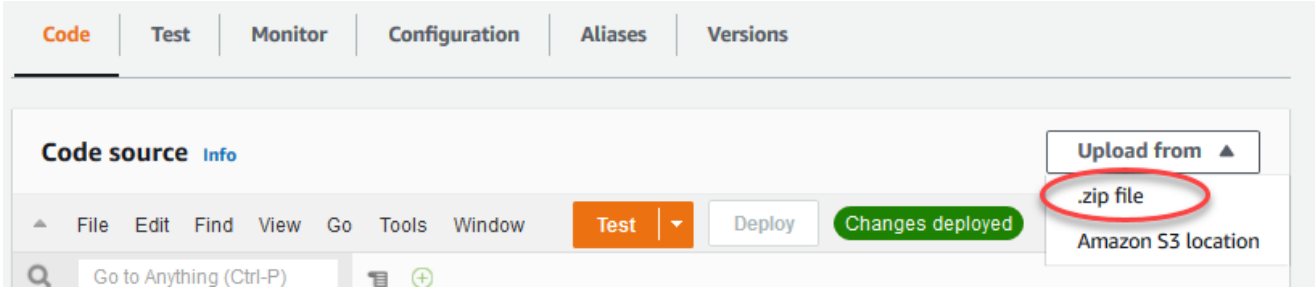
Au cours de cette étape, vous utilisez lesAWS Lambdapour créer une fonction Lambda et pour la configurer afin qu'elle utilise votre package de déploiement. Vous publiez ensuite une version de fonction et créez un alias.

Créez d'abord la fonction Lambda.

1. Dans AWS Management Console, choisissez Services et ouvrez la console AWS Lambda.
2. ChoisissezFonctions.
3. ChoisissezCréation de fonctionpuis choisissezCréer à partir de zéro.
4. Dans la section Informations de base, spécifiez les valeurs suivantes :
   a. Sous Nom de la fonction, saisissez **`TestLRA`**.
   b. Pour Runtime, sélectionnez Python 3.7.
   c. PourAutorisations, conservez le paramètre par défaut. Cela crée un rôle d'exécution qui accorde des autorisations Lambda de base. Ce rôle n'est pas utilisé parAWS IoT Greengrass.
5. Sélectionnez Create function (Créer une fonction).



6. Chargez le package de déploiement de votre fonction Lambda et enregistrez le gestionnaire.
   a. Dans la pageCodeonglet, sousCode source, choisissezChargement à partir de. Dans le menu déroulant, choisissezfichier .zip.



b. ChoisissezCharger, puis choisissez votre`lraTestLambda.zip`package de déploiement. Ensuite, choisissez Enregistrer.

c. Dans la pageCodepour la fonction, sousParamètres d'exécution, choisissezModifier, puis entrez les valeurs suivantes.

- Pour Runtime, sélectionnez Python 3.7.
- Pour Handler, entrez lraTest.function_handler.

d. Choisissez Save (Enregistrer).

> **Note**
>
> LeTestsur leAWS Lambdane fonctionne pas avec cette fonction. LeAWS IoT GreengrassLe SDK principal ne contient pas de modules nécessaires pour exécuter vos fonctions Greengrass Lambda indépendamment dans leAWS Lambdaconsole Ces modules (par exemple,`greengrass_common`) sont fournis aux fonctions après leur déploiement dans votre cœur Greengrass.

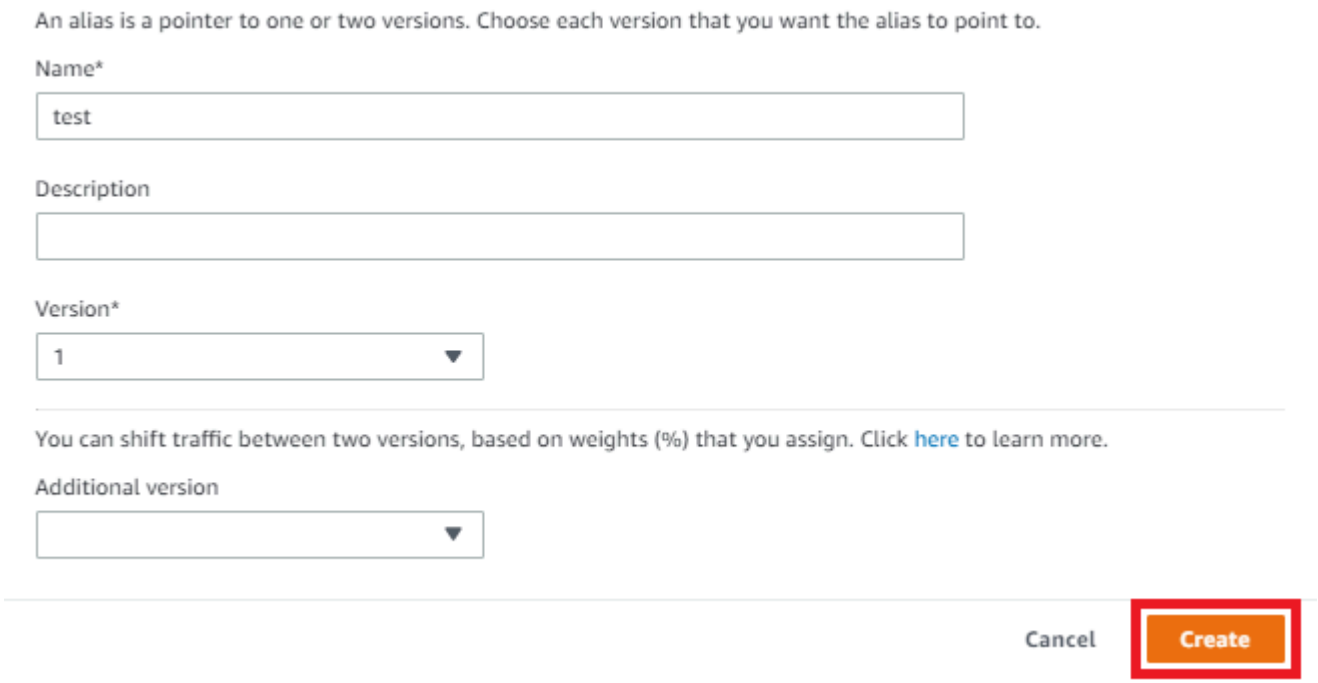
Publiez ensuite la première version de votre fonction Lambda. Puis, créez un alias pour la version.

Les groupes Greengrass peuvent référencer une fonction Lambda par alias (recommandé) ou par version. L'utilisation d'un alias facilite la gestion des mises à jour de code, car vous n'avez pas besoin de changer votre table d'abonnement ou votre définition de groupe lorsque le code de fonction est mis à jour. Au lieu de cela, il vous suffit de pointer l'alias vers la nouvelle version de fonction.

7. Dans Actions, choisissez Publier une nouvelle version.
8. Dans Description de la version, saisissez **`First version`**, puis choisissez Publish.
9. Sur la page de configuration TestLRA: 1, dans le menu Actions, choisissez Create alias (Créer un alias).
10. Dans la pageCréer un aliaspage, pourNom, saisissez**`test`**. Pour Version, entrez 1.

> **Note**
>
> AWS IoT Greengrassne prend pas en charge les alias Lambda pour$LATESTversions.

11. Sélectionnez Create (Créer).



Vous pouvez désormais ajouter la fonction Lambda à votre groupe Greengrass.

Étape 3 : Ajouter la fonction Lambda au groupe Greengrass

Au cours de cette étape, vous ajoutez la fonction à votre groupe et configurez le cycle de vie de la fonction.

Ajoutez d'abord la fonction Lambda à votre groupe Greengrass.

1. DansAWS IoTVolet de navigation de la consoleGérer, DéveloppezAppareils Greengrass, puis choisissezGroups (V1).
2. Choisissez le groupe Greengrass auquel vous souhaitez ajouter la fonction Lambda.
3. Sur la page de configuration de groupe, choisissez leFonctions LambdaOnglet.
4. UNDERFonctions Lambdasection, choisissezAddition.
5. Dans la pageFonction Lambda, choisissez l'Fonction Lambda. Sélectionnez **`TestLRA`**.
6. Cliquez sur l'ongletVersion de la fonction Lambda.
7. DansConfiguration de fonction Lambdasection, sélectionnezUtilisateur et groupe du systèmeetConteneurisation de fonction Lambda.

Configurez ensuite le cycle de vie de la fonction Lambda.

8. Pour Timeout (Délai), sélectionnez 30 secondes.

> **Important**
>
> Les fonctions Lambda qui utilisent des ressources locales (comme décrit dans cette procédure) doivent s'exécuter dans un conteneur Greengrass. Sinon, le déploiement échoue si vous essayez de déployer la fonction. Pour plus d'informations, consultez Conteneurisation.

9. Au bas de la page, choisissezFonction Lambda.

Étape 4 : Ajouter une ressource locale au groupe Greengrass

Au cours de cette étape, vous ajoutez une ressource de volume locale au groupe Greengrass et accordez à la fonction un accès en lecture et en écriture à la ressource. Une ressource locale a une portée au niveau du groupe. Vous pouvez accorder des autorisations pour n'importe quelle fonction Lambda dans le groupe pour accéder à la ressource.

1. Sur la page de configuration de groupe, choisissez leRessourcesOnglet.
2. SousDes ressources localessection, choisissezAddition.
3. Dans la pageAjouter une ressource locale, utilisez les valeurs suivantes.

   a. Sous Resource Name (Nom de la ressource), entrez **`testDirectory`**.

   b. Pour Type de ressource, choisissez Volume.

   c. PourChemin de l'appareil local, saisissez**`/src/LRAtest`**. (Ce chemin doit exister sur le système d'exploitation hôte.)

      Le chemin de périphérique local est le chemin absolu local de la ressource sur le système de fichiers de l'appareil principal. Cet emplacement est en dehors du conteneur dans lequel la fonction s'exécute. Le chemin ne peut pas commencer par `/sys`.

   d. Pour Destination path (Chemin d'accès de destination), entrez **`/dest/LRAtest`**. (Ce chemin doit exister sur le système d'exploitation hôte.)

Le chemin de destination est le chemin absolu de la ressource dans l'espace de noms Lambda. Cet emplacement est dans le conteneur dans lequel la fonction s'exécute.

e. UNDERPropriétaire du groupe système et autorisation d'accès aux fichiers, sélectionnezAjouter automatiquement les autorisations de système de fichiers du groupe de système qui possède la ressource.

LePropriétaire du groupe système et autorisation d'accès aux fichiersvous permet d'accorder au processus Lambda des autorisations supplémentaires d'accès aux fichiers au processus Lambda. Pour plus d'informations, consultez Autorisation d'accès fichier pour le propriétaire du groupe.

4. Choisissez Add resource (Ajouter ressource). La page Resources contient la nouvelle ressource testDirectory.

Étape 5 : Ajouter des abonnements au groupe Greengrass

Au cours de cette étape, vous ajoutez deux abonnements au groupe Greengrass. Ces abonnements permettent une communication bidirectionnelle entre la fonction Lambda etAWS IoT.

Créez d'abord un abonnement pour la fonction Lambda afin d'envoyer des messages àAWS IoT.

1. Sur la page de configuration de groupe, choisissez leSubscriptionsOnglet.
2. Choisissez Add (Ajouter).
3. Dans la pageCréer un abonnement, configurez la source et la cible comme suit :

   a. PourType de source, choisissezFonction Lambda, puis choisissezTestLRA.

   b. PourTarget type (Type de cible), choisissezService, puis choisissezCloud IoT.

   c. PourFiltre de rubriques, saisissez**`LRA/test`**, puis choisissezCréer un abonnement.
4. La page Abonnements affiche le nouvel abonnement.

   Configurez ensuite un abonnement qui appelle la fonction depuis AWS IoT.
5. Sur la page Abonnements, choisissez Ajouter un abonnement.
6. Sur la page Sélectionnez la source et la cible, configurez la source et la cible comme suit :

   a. PourType de source, choisissezFonction Lambda, puis choisissezCloud IoT.

b. PourTarget type (Type de cible), choisissezService, puis choisissezTestLRA.

c. Sélectionnez Suivant.

7. Dans la page Filtrer vos données avec une rubrique, dans Filtre de rubrique, saisissez **`invoke/LRAFunction`**, puis choisissez Suivant.

8. Choisissez Finish (Terminer). La page Abonnements affiche les deux abonnements.

Étape 6 : Déploiement de l'AWS IoT Greengrassgroupe

Au cours de cette étape, vous déployez la version actuelle de la définition de groupe.

1. Assurez-vous que lesAWS IoT Greengrasscore est en cours d'exécution. Dans la fenêtre de terminal de votre Raspberry Pi, exécutez les commandes suivantes, si nécessaire.

   a. Pour vérifier si le démon est en cours d'exécution :

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

   Si la sortie contient une entrée `root` pour `/greengrass/ggc/packages/1.11.6/bin/daemon`, le démon est en cours d'exécution.

   > **Note**
   >
   > La version du chemin d'accès dépend de la version du logiciel AWS IoT Greengrass Core installée sur votre appareil principal.

   b. Pour démarrer le démon :

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

2. Sur la page de configuration de groupe, choisissezDéploiement.

   > **Note**
   >
   > Le déploiement échoue si vous exécutez votre fonction Lambda sans conteneurisation et essayez d'accéder aux ressources locales attachées.

3. Si vous y êtes invité, sur leFonction Lambdaonglet, sousFonctions Lambda du système, sélectionnezDétecteur IP, puisModifier, puisDétecter automatiquement.

   Les appareils peuvent ainsi acquérir automatiquement des informations de connectivité pour le noyau, telles que l'adresse IP, le DNS et le numéro de port. La détection automatique est recommandée, mais AWS IoT Greengrass prend également en charge les points de terminaison spécifiés manuellement. Vous êtes uniquement invité à indiquer la méthode de découverte lors du déploiement initial du groupe.

   > **Note**
   >
   > Si vous y êtes invité, autorisez la création duRôle de service Greengrassetet associez-le à votreCompte AWSdans l'actuelRégion AWS. Ce rôle permetAWS IoT Greengrasspour accéder à vos ressources dansAWSServices .

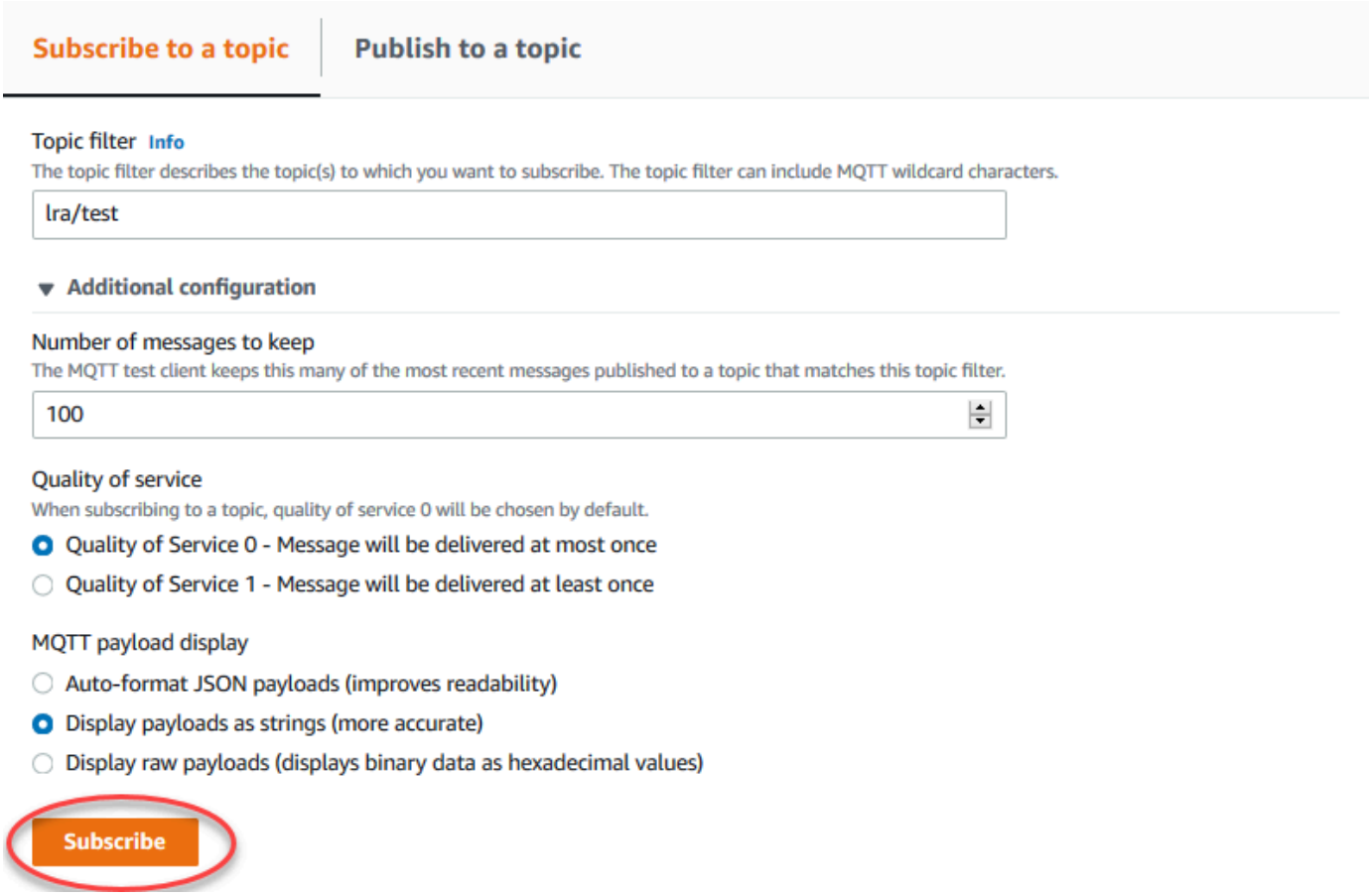
   La page Déploiements indique l'horodatage, l'ID de version et l'état du déploiement. Une fois cette procédure exécutée, le statut du déploiement estTerminé.

   Pour bénéficier d'une aide à la résolution des problèmes, consultez Résolution des problèmes.

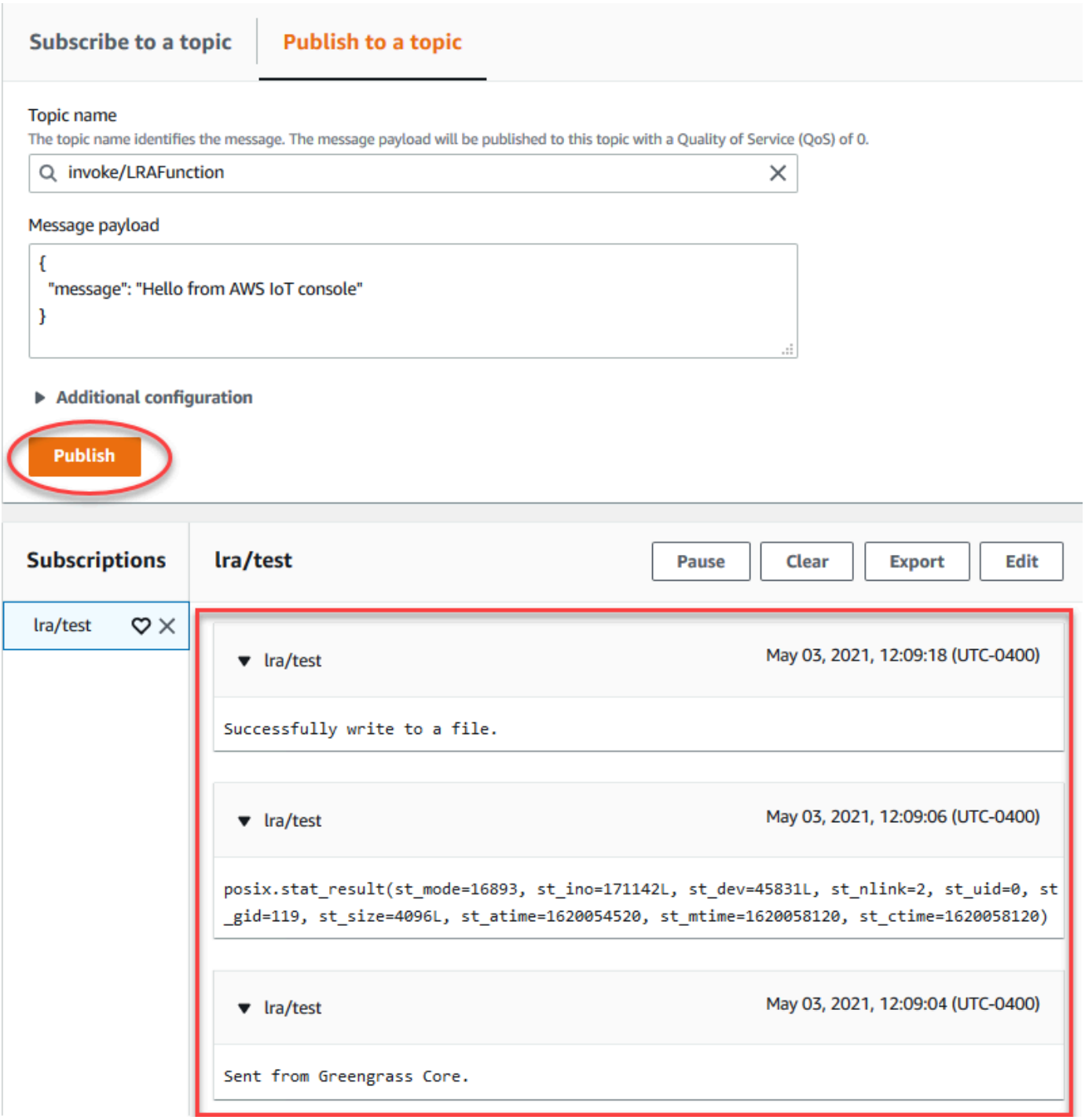
Tester l'accès aux ressources locales

À présent, vous pouvez vérifier si l'accès aux ressources locales est correctement configuré. Pour effectuer le test, abonnez-vous à la rubrique `LRA/test` et publiez n'importe quel message dans la rubrique `invoke/LRAFunction`. Le test réussit si la fonction Lambda envoie la charge utile attendue àAWS IoT.

1. À partir de laAWS IoTmenu de navigation de la console, sousTest, choisissezClient de test MQTT.
2. UNDERS'abonner à une rubrique, pourFiltre de rubriques, saisissez**`LRA/test`**.
3. UNDERInformations supplémentaires, pourAffichage de la charge utile MQTT, sélectionnezAffichage des charges utiles sous forme de chaînes.
4. Choisissez Subscribe. Votre fonction Lambda publie dans la rubrique LRA/test.



5. UNDERPublier dans une rubrique, dans leNom de la rubriqueentrer**`invoke/LRAFunction`**, puis choisissezPublierpour appeler votre fonction Lambda. Le test réussit si la page affiche les trois charges utiles de messages de la fonction.



Le fichier de test créé par la fonction Lambda se trouve dans le`/src/LRAtest`répertoire sur l'appareil principal Greengrass. Bien que la fonction Lambda écrit dans un fichier dans le`/dest/LRAtest`, ce fichier est visible dans l'espace de noms Lambda uniquement. Vous ne pouvez pas le voir dans un espace de noms Linux régulier. Toute modification du chemin de destination est néanmoins pris en compte dans le chemin source sur le système de fichiers.

Pour bénéficier d'une aide à la résolution des problèmes, consultez Résolution des problèmes.

Exécuter l'inférence de Machine Learning

Cette fonction est disponible pourAWS IoT GreengrassCore v1.6 ou une version ultérieure.

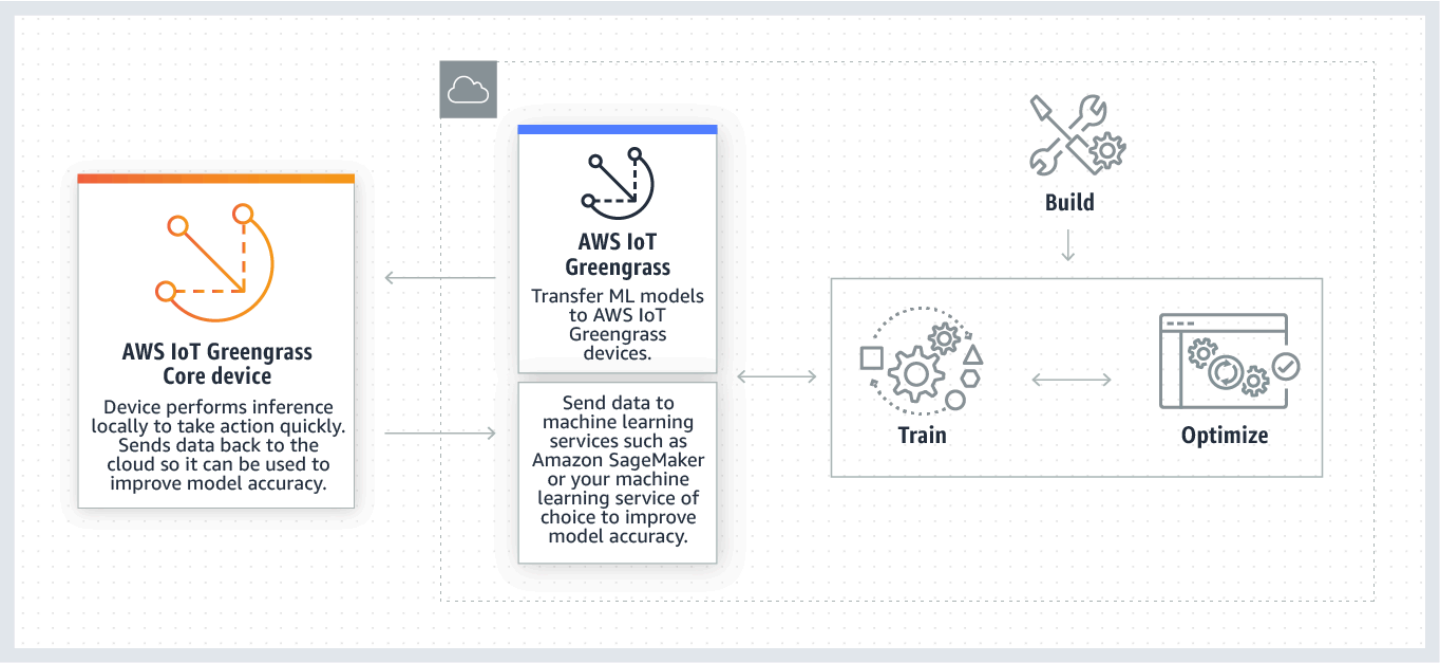
AWS IoT Greengrass vous permet d'exécuter l'inférence de Machine Learning en périphérie sur des données générées localement à l'aide de modèles formés dans le cloud. Vous bénéficiez d'une faible latence et de coûts d'inférence locale réduits, tout en profitant des avantages de la puissance du cloud computing pour les modèles de formation et les traitements complexes.

Pour commencer à exécuter l'inférence locale, consultez the section called “Configuration de l'inférence Machine Learning”.

Fonctionnement de l'inférence de Machine Learning AWS IoT Greengrass

Vous pouvez former vos modèles d'inférence n'importe où, les déployer localement commeressources de Machine Learningdans un groupe Greengrass, puis y accéder à partir des fonctions Lambda Greengrass. Vous pouvez par exemple créer et former des modèles d'apprentissage profond dansSageMakeret déployez-les sur votre noyau Greengrass. Vos fonctions Lambda peuvent ensuite utiliser les modèles locaux pour exécuter l'inférence sur des appareils connectés et renvoyer de nouvelles données d'entraînement vers le cloud.

Le schéma suivant illustre le flux de travail de l'inférence de Machine Learning AWS IoT Greengrass.



L'inférence de Machine Learning AWS IoT Greengrass simplifie chaque étape du flux de travail de Machine Learning, y compris :

- La création et le déploiement de prototypes d'infrastructure de Machine Learning.
- L'accès aux modèles formés dans le cloud et leur déploiement dans les appareils Greengrass principaux.
- La création d'applications d'inférence pouvant accéder à des accélérateurs matériels (tels que des GPU et des FPGA) en tant que ressources locales.

Ressource de Machine Learning

Les ressources de Machine Learning représentent des modèles d'inférence formés dans le cloud et déployés dans unAWS IoT Greengrassnoyau. Pour déployer des ressources de Machine Learning, commencez par ajouter les ressources à un groupe Greengrass, puis définissez la méthode d'accès des fonctions Lambda à ces ressources. Pendant le déploiement de groupe,AWS IoT Greengrassrécupère les packages de modèles source à partir du cloud et les extrait dans des répertoires à l'intérieur de l'espace de noms d'exécution Lambda. Les fonctions Lambda Greengrass utilisent ensuite les modèles déployés localement pour exécuter l'inférence.

Pour mettre à jour un modèle déployé localement, commencez par mettre à jour le modèle source (dans le cloud) qui correspond à la ressource de Machine Learning, puis déployez le groupe. Pendant le déploiement, AWS IoT Greengrass vérifie si la source a fait l'objet de modifications. Si des modifications sont détectées, AWS IoT Greengrass met à jour le modèle local.

Sources de modèles prises en charge

AWS IoT Greengrassappuis SageMaker et des sources de modèles Amazon S3 pour les ressources de Machine Learning.

Les exigences suivantes s'appliquent aux sources de modèles :

- Godets S3 qui stockent votre SageMaker et les sources du modèle Amazon S3 ne doivent pas être chiffrées à l'aide de SSE-C. Pour les compartiments qui utilisent le chiffrement côté serveur,AWS IoT GreengrassL'inférence de Machine Learning prend actuellement en charge les options de chiffrement SSE-S3 ou SSE-KMS uniquement. Pour de plus amples informations sur les options de chiffrement côté serveur, veuillez consulter.Protection des données à l'aide d'un chiffrement côté serveurdans leManuel de l'utilisateur Amazon Simple Storage Service.

- Les noms des compartiments S3 qui stockent votre SageMaker et les sources du modèle Amazon S3 ne doivent pas contenir de points (`.`). Pour de plus amples informations, veuillez consulter la règle d'utilisation des compartiments d'hébergement virtuel avec SSL dansRègles relatives à l'attribution des noms de compartimentsdans leManuel de l'utilisateur Amazon Simple Storage Service.
- Niveau de serviceRégion AWSl'assistance doit être disponible pour les deuxAWS IoT GreengrassetSageMaker. Actuellement,AWS IoT Greengrassappuis SageMaker modèles dans les régions suivantes :
  - US East (Ohio)
  - USA Est (Virginie du Nord)
  - USA Ouest (Oregon)
  - Asie-Pacifique (Mumbai)
  - Asie-Pacifique (Séoul)
  - Asie-Pacifique (Singapour)
  - Asie-Pacifique (Sydney)
  - Asie-Pacifique (Tokyo)
  - Europe (Francfort)
  - Europe (Irlande)
  - Europe (Londres)
- AWS IoT Greengrass doit disposer de l'autorisation `read` pour la source de modèle, comme décrit dans les sections suivantes.

SageMaker

AWS IoT Greengrassprend en charge les modèles enregistrés sous SageMaker tâches d'entraînement. SageMaker est un service de Machine Learning entièrement géré qui vous permet de créer et de former des modèles à l'aide d'algorithmes intégrés ou personnalisés. Pour de plus amples informations, veuillez consulterQu'est-ce que SageMaker ?dans leManuel du développeur SageMaker.

Si vous avez configuré votre SageMaker Environment bycréer un compartimentdont le nom contient`sagemaker`, puisAWS IoT Greengrassdispose d'une autorisation suffisante pour accéder à votre SageMaker tâches d'entraînement. La stratégie gérée

`AWSGreengrassResourceAccessRolePolicy` autorise l'accès aux compartiments dont le nom contient la chaîne `sagemaker`. Cette stratégie est attachée au rôle de service Greengrass.

Dans le cas contraire, vous devez accorder à AWS IoT Greengrass l'autorisation `read` pour le compartiment dans lequel votre tâche de formation est stockée. Pour ce faire, intégrez la stratégie en ligne suivante dans le rôle de service. Vous pouvez répertorier plusieurs ARN de compartiment.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::my-bucket-name"
            ]
        }
    ]
}
```

Amazon S3

AWS IoT Greengrassprend en charge les modèles stockés dans Amazon S3 sous`tar.gz`ou`.zip`fichiers suivants.

Pour activerAWS IoT GreengrassPour accéder aux modèles stockés dans des compartiments Amazon S3, vous devez accorderAWS IoT Greengrass `read`autorisation d'accès aux compartiments en effectuantundes éléments suivants :

- Stockez votre modèle dans un compartiment dont le nom contient `greengrass`.

  La stratégie gérée `AWSGreengrassResourceAccessRolePolicy` autorise l'accès aux compartiments dont le nom contient la chaîne `greengrass`. Cette stratégie est attachée au rôle de service Greengrass.

- Intégrez une stratégie en ligne dans le rôle de service Greengrass.

Si le nom de votre compartiment ne contient pas `greengrass`, ajoutez la stratégie en ligne suivante au rôle de service. Vous pouvez répertorier plusieurs ARN de compartiment.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::my-bucket-name"
            ]
        }
    ]
}
```

Pour de plus amples informations, veuillez consulterIntégrer des stratégies en lignedans leIAM User Guide.

Prérequis

Les exigences suivantes s'appliquent pour la création et l'utilisation des ressources de Machine Learning :

- Vous devez utiliserAWS IoT GreengrassCore v1.6 ou une version ultérieure.
- Les fonctions Lambda définies par l'utilisateur peuvent exécuter`readouread and writeopérations sur la ressource. Les autorisations pour d'autres opérations ne sont pas disponibles. Le mode conteneurisation des fonctions Lambda affiliées détermine la façon dont vous définissez les autorisations d'accès. Pour plus d'informations, consultez the section called “Accès aux ressources de machine learning”.
- Vous devez fournir le chemin d'accès complet de la ressource sur le système d'exploitation de l'appareil principal.
- Un ID ou un nom de ressource doit comporter 128 caractères au maximum et utiliser le modèle `[a-zA-Z0-9:_-]+`.

Environnements d'exécution et bibliothèques pour l'inférence ML

Vous pouvez utiliser les environnement d'exécution et bibliothèques ML suivants avec AWS IoT Greengrass.

- Amazon SageMaker Exécution Neo deep learning
- Apache MXNet
- TensorFlow

Ces environnements d’exécution et ces bibliothèques peuvent être installés sur les plateformes NVIDIA Jetson TX2, Intel Atom et Raspberry Pi. Pour obtenir des informations sur le téléchargement, consultez the section called “Bibliothèques et environnements d'exécution de machine learning pris en charge”. Vous pouvez les installer directement sur votre appareil principal (noyau).

Assurez-vous de lire les informations de compatibilités et les limitations suivantes.

Exécution SageMaker Neo Deep Learning

Vous pouvez utiliser le plugin SageMaker Exécution Neo deep learning pour exécuter des inférences avec des modèles de machine learning optimisés sur votreAWS IoT Greengrassappareils. Ces modèles sont optimisés à l'aide du SageMaker Compilateur Neo Deep Learning pour améliorer les vitesses de prédiction des inférences de Machine Learning. Pour de plus amples informations sur l'optimisation des modèles dans SageMaker, consultez leDocumentation SageMaker Neo.

> **Note**
>
> Actuellement, vous pouvez optimiser des modèles de Machine Learning à l'aide du compilateur Neo Deep Learning dans des régions Amazon Web Services spécifiques uniquement. Cependant, vous pouvez utiliser le moteur d'exécution Neo Deep Learning avec des modèles optimisés dans chaqueRégion AWSoùAWS IoT Greengrasscore est pris en charge. Pour plus d'informations, consultez Configuration de l'inférence Machine Learning optimisée.

Gestion des versions MXNet

Apache MXNet ne garantit actuellement pas une compatibilité ascendante. Par conséquent, les modèles que vous formez à l'aide de versions ultérieures de l'infrastructure peuvent ne pas

fonctionner correctement dans des versions antérieures de l'infrastructure. Pour éviter les conflits entre les étapes de formation des modèles et de service des modèles, et fournir une cohérence end-to-end , utilisez la même version du framework MXNet dans les deux étapes.

MXNet sur Raspberry Pi

Les fonctions Lambda Greengrass qui accèdent aux modèles MXNet locaux doivent définir la variable d'environnement suivante :

```
MXNET_ENGINE_TYPE=NativeEngine
```

Vous pouvez définir la variable d'environnement dans le code de la fonction ou l'ajouter à la configuration spécifique au groupe de la fonction. Cette étape présente un exemple qui ajoute la variable d'environnement en tant que paramètre de configuration.

> **Note**
>
> Pour une utilisation générale de l'infrastructure MXNet, telle que l'exécution d'un exemple de code tiers, la variable d'environnement doit être configurée sur le Raspberry Pi.

Limitations d'utilisation des modèles TensorFlow sur Raspberry Pi

Les recommandations suivantes visent à améliorer les résultats d'inférence et sont basées sur les tests réalisés avec le TensorFlow bibliothèques Arm 32 bits sur la plateforme Raspberry Pi. Ces recommandations sont destinées aux utilisateurs avancés pour référence uniquement, sans garantie d'aucune sorte.

- Les modèles formés à l'aide du format Checkpoint doivent être « bloqués » dans le format tampon du protocole avant d'être utilisés. Pour obtenir un exemple, consultez la bibliothèque du modèle de classification d'images TensorFlow-Slim.
- N'utilisez pas les bibliothèques TF-Estimator et TF-Slim dans le code de formation ou d'inférence. À la place, utilisez le modèle de chargement de fichier `.pb` indiqué dans l'exemple suivant.

```
graph = tf.Graph()
graph_def = tf.GraphDef()
graph_def.ParseFromString(pb_file.read())
with graph.as_default():
  tf.import_graph_def(graph_def)
```

> **Note**
>
> Pour de plus amples informations sur les plateformes prises en charge pour TensorFlow, veuillez consulter.Installation de TensorFlowdans le TensorFlow .

Accédez aux ressources d'apprentissage automatique à partir des fonctions Lambda

Les fonctions Lambda définies par l'utilisateur peuvent accéder aux ressources d'apprentissage automatique pour exécuter une inférence locale sur le cœur. AWS IoT Greengrass Une ressource de Machine Learning se compose du modèle formé et d'autres artefacts qui sont téléchargés sur l'appareil principal (noyau).

Pour permettre à une fonction Lambda d'accéder à une ressource d'apprentissage automatique sur le cœur, vous devez associer la ressource à la fonction Lambda et définir des autorisations d'accès. Le mode de conteneurisation de la fonction Lambda affiliée (ou attachée) détermine la manière dont vous procédez.

Autorisations d'accès pour les ressources de machine learning

À compter de la version 1.10.0 d'AWS IoT Greengrass Core, vous pouvez définir un propriétaire de ressource pour une ressource de Machine Learning. Le propriétaire de la ressource représente le groupe de système d'exploitation et les autorisations utilisés par AWS IoT Greengrass pour télécharger les artefacts de ressource. Si aucun propriétaire de ressource n'est défini, les artefacts de ressource téléchargés ne sont accessibles que pour l'utilisateur racine (root).

- Si des fonctions Lambda non conteneurisées accèdent à une ressource d'apprentissage automatique, vous devez définir un propriétaire de ressource car le conteneur ne permet aucun contrôle d'autorisation. Les fonctions Lambda non conteneurisées peuvent hériter des autorisations du propriétaire de la ressource et les utiliser pour accéder à la ressource.

- Si seules les fonctions Lambda conteneurisées accèdent à la ressource, nous vous recommandons d'utiliser des autorisations au niveau des fonctions au lieu de définir un propriétaire de ressource.

Propriétés du propriétaire de la ressource

Un propriétaire de ressource spécifie un propriétaire de groupe et des autorisations de propriétaire de groupe.

Propriétaire du groupe. ID (GID) d'un groupe d'OS Linux existant sur l'appareil principal (noyau). Les autorisations du groupe sont ajoutées au processus Lambda. Plus précisément, le GID est ajouté aux identifiants de groupe supplémentaires de la fonction Lambda.

Si une fonction Lambda du groupe Greengrass est configurée pour s'exécuter dans le même groupe de système d'exploitation que le propriétaire de la ressource d'une ressource d'apprentissage automatique, la ressource doit être attachée à la fonction Lambda. Dans le cas contraire, le déploiement échoue car cette configuration donne des autorisations implicites que la fonction Lambda peut utiliser pour accéder à la ressource sans AWS IoT Greengrass autorisation. La vérification de validation du déploiement est ignorée si la fonction Lambda s'exécute en tant que root (UID=0).

Nous vous recommandons d'utiliser un groupe de systèmes d'exploitation qui n'est pas utilisé par d'autres ressources, fonctions Lambda ou fichiers du noyau de Greengrass. L'utilisation d'un groupe de systèmes d'exploitation partagé donne aux fonctions Lambda associées des autorisations d'accès supérieures à celles dont elles ont besoin. Si vous utilisez un groupe de systèmes d'exploitation partagé, une fonction Lambda associée doit également être attachée à toutes les ressources d'apprentissage automatique qui utilisent le groupe de systèmes d'exploitation partagé. Sinon, le déploiement échoue.

Autorisations du propriétaire du groupe. L'autorisation de lecture seule ou de lecture et d'écriture à ajouter au processus Lambda.

Les fonctions Lambda non conteneurisées doivent hériter de ces autorisations d'accès à la ressource. Les fonctions Lambda conteneurisées peuvent hériter de ces autorisations au niveau des ressources ou définir des autorisations au niveau des fonctions. Si elles définissent des autorisations de niveau fonction, ces autorisations doivent être identiques ou plus restrictives que les autorisations de niveau ressource.

Le tableau suivant présente les configurations d'autorisation d'accès prises en charge.

GGC v1.10 or later

Propriété	Si seules les fonctions Lambda conteneurisées accèdent à la ressource	Si des fonctions Lambda non conteneurisées accèdent à la ressource
Propriétés de niveau fonction		
Autorisations (lecture/écriture)	Obligatoires, sauf si la ressource définit un propriétaire de ressource. Si un propriétaire de ressource est défini, les autorisations de niveau fonction doivent être identiques ou plus restrictives que les autorisations du propriétaire de la ressource. Si seules les fonctions Lambda conteneurisées accèdent à la ressource, nous vous recommandons de ne pas définir de propriétaire de ressource.	Fonctions Lambda non conteneurisées : Non pris en charge. Les fonctions Lambda non conteneurisées doivent hériter des autorisations au niveau des ressources. Fonctions Lambda conteneurisées : Facultatives, mais doivent être identiques ou plus restrictives que les autorisations de niveau ressource.
Propriétés de niveau ressource		
Propriétaire de ressource	Facultatif (non recommandé).	Obligatoire.
Autorisations (lecture/écriture)	Facultatif (non recommandé).	Obligatoire.

GGC v1.9 or earlier

Propriété	Si seules les fonctions Lambda conteneurisées accèdent à la ressource	Si des fonctions Lambda non conteneurisées accèdent à la ressource
Propriétés de niveau fonction		
Autorisations (lecture/écriture)	Obligatoire.	Non pris en charge.
Propriétés de niveau ressource		
Propriétaire de ressource	Non pris en charge.	Non pris en charge.
Autorisations (lecture/écriture)	Non pris en charge.	Non pris en charge.

> **Note**
>
> Lorsque vous utilisez l'AWS IoT GreengrassAPI pour configurer des fonctions et des ressources Lambda, la `ResourceId` propriété au niveau de la fonction est également requise. La `ResourceId` propriété associe la ressource d'apprentissage automatique à la fonction Lambda.

Définition des autorisations d'accès pour les fonctions Lambda (console)

Dans la AWS IoT console, vous définissez les autorisations d'accès lorsque vous configurez une ressource de machine learning ou que vous en associez une à une fonction Lambda.

Fonctions Lambda conteneurisées

Si seules des fonctions Lambda conteneurisées sont associées à la ressource d'apprentissage automatique :

- Choisissez Aucun groupe système comme propriétaire de la ressource d'apprentissage automatique. Il s'agit du paramètre recommandé lorsque seules les fonctions Lambda conteneurisées accèdent à la ressource d'apprentissage automatique. Dans le cas contraire,

vous pourriez accorder aux fonctions Lambda associées des autorisations d'accès supérieures à celles dont elles ont besoin.

Fonctions Lambda non conteneurisées (nécessite GGC v1.10 ou version ultérieure)

Si des fonctions Lambda non conteneurisées sont associées à la ressource d'apprentissage automatique :

- Spécifiez l'ID de groupe système (GID) à utiliser en tant que propriétaire de la ressource d'apprentissage automatique. Choisissez Spécifier le groupe système et les autorisations, puis entrez le GID. Vous pouvez utiliser la `getent group` commande sur votre appareil principal pour rechercher l'ID d'un groupe de systèmes.

- Choisissez Accès en lecture seule ou Accès en lecture et écriture pour les autorisations du groupe système.

Définition des autorisations d'accès pour les fonctions Lambda (API)

Dans l'AWS IoT GreengrassAPI, vous définissez les autorisations d'accès aux ressources de machine learning dans la `ResourceAccessPolicy` propriété de la fonction Lambda ou dans la `OwnerSetting` propriété de la ressource.

Fonctions Lambda conteneurisées

Si seules des fonctions Lambda conteneurisées sont associées à la ressource d'apprentissage automatique :

- Pour les fonctions Lambda conteneurisées, définissez les autorisations d'accès dans la `Permission` propriété de la propriété. `ResourceAccessPolicies` Par exemple :

```
"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
        }
    }
]
```

- Pour les ressources de Machine Learning, omettez la propriété `OwnerSetting`. Par exemple :

```
"Resources": [
    {
        "Id": "my-resource-id",
        "Name": "my-resource-name",
        "ResourceDataContainer": {
            "S3MachineLearningModelResourceData": {
                "DestinationPath": "/local-destination-path",
                "S3Uri": "s3://uri-to-resource-package"
            }
        }
    }
]
```

  Il s'agit de la configuration recommandée lorsque seules les fonctions Lambda conteneurisées accèdent à la ressource d'apprentissage automatique. Dans le cas contraire, vous pourriez accorder aux fonctions Lambda associées des autorisations d'accès supérieures à celles dont elles ont besoin.

Fonctions Lambda non conteneurisées (nécessite GGC v1.10 ou version ultérieure)

Si des fonctions Lambda non conteneurisées sont associées à la ressource d'apprentissage automatique :

- Pour les fonctions Lambda non conteneurisées, omettez la propriété dans. `Permission` `ResourceAccessPolicies` Cette configuration est requise et permet à la fonction d'hériter de l'autorisation de niveau ressource. Par exemple :

```
"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
        }
    }
]
```

- Pour les fonctions Lambda conteneurisées qui accèdent également à la ressource d'apprentissage automatique, omettez la `Permission` propriété `ResourceAccessPolicies` ou définissez une autorisation identique ou plus restrictive que l'autorisation au niveau de la ressource. Par exemple :

```
"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
        }
    }
]
```

- Pour les ressources de Machine Learning, définissez la propriété `OwnerSetting`, ainsi que les propriétés enfants `GroupOwner` et `GroupPermission`. Par exemple :

```
"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"
                }
            }
        }
    }
]
```

Accès aux ressources d'apprentissage automatique à partir du code de fonction Lambda

Les fonctions Lambda définies par l'utilisateur utilisent des interfaces de système d'exploitation spécifiques à la plate-forme pour accéder aux ressources d'apprentissage automatique sur un périphérique principal.

GGC v1.10 or later

Pour les fonctions Lambda conteneurisées, la ressource est montée dans le conteneur Greengrass et disponible sur le chemin de destination local défini pour la ressource. Pour les fonctions Lambda non conteneurisées, la ressource est liée symboliquement à un répertoire de travail spécifique à Lambda et transmise à la variable d'environnement dans le processus Lambda. `AWS_GG_RESOURCE_PREFIX`

Pour obtenir le chemin d'accès aux artefacts téléchargés d'une ressource d'apprentissage automatique, les fonctions Lambda ajoutent la variable d'`AWS_GG_RESOURCE_PREFIX`environnement au chemin de destination local défini pour la ressource. Pour les fonctions Lambda conteneurisées, la valeur renvoyée est une barre oblique unique (). `/`

```
resourcePath = os.getenv("AWS_GG_RESOURCE_PREFIX") + "/destination-path"
with open(resourcePath, 'r') as f:
    # load_model(f)
```

GGC v1.9 or earlier

Les artefacts téléchargés d'une ressource de Machine Learning se trouvent dans le chemin de destination local défini pour la ressource. Seules les fonctions Lambda conteneurisées peuvent accéder aux ressources d'apprentissage automatique dans AWS IoT Greengrass Core v1.9 et versions antérieures.

```
resourcePath = "/local-destination-path"
with open(resourcePath, 'r') as f:
    # load_model(f)
```

Votre implémentation du chargement du modèle dépend de votre bibliothèque de Machine Learning.

Résolution des problèmes

Utilisez les informations suivantes pour résoudre les problèmes liés à l'accès aux ressources de Machine Learning.

Rubriques

- InvalidML ModelOwner - GroupOwnerSetting est fourni dans la ressource du modèle ML, mais n' GroupOwner GroupPermission est pas présent

- NoContainer La fonction ne peut pas configurer les autorisations lors de l'attachement de ressources Machine Learning. <function-arn>fait référence à une ressource d'apprentissage automatique <resource-id>avec autorisation <ro/rw> dans la politique d'accès aux ressources.
- <function-arn>La fonction fait référence à une ressource Machine Learning dont l'<resource-id>autorisation est manquante à la fois dans la ressource ResourceAccessPolicy et dans la ressource OwnerSetting.
- <function-arn>La fonction fait référence à la ressource Machine Learning <resource-id>avec l'autorisation \ "rw \ », tandis que le paramètre du propriétaire de la ressource autorise GroupPermission uniquement \ "ro \ ».
- NoContainer La fonction <function-arn>fait référence aux ressources du chemin de destination imbriqué.
- La fonction Lambda <function-arn> accède à la ressource <resource-id> en partageant le même identifiant de propriétaire de groupe

InvalidML ModelOwner - GroupOwnerSetting est fourni dans la ressource du modèle ML, mais n' GroupOwner GroupPermission est pas présent

Solution : vous recevez cette erreur si une ressource d'apprentissage automatique contient l'ResourceDownloadOwnerSettingobjet mais que la `GroupPermission` propriété `GroupOwner` ou la propriété requise n'est pas définie. Pour résoudre ce problème, définissez la propriété manquante.

NoContainer La fonction ne peut pas configurer les autorisations lors de l'attachement de ressources Machine Learning. <function-arn>fait référence à une ressource d'apprentissage automatique <resource-id>avec autorisation <ro/rw> dans la politique d'accès aux ressources.

Solution : vous recevez cette erreur si une fonction Lambda non conteneurisée spécifie des autorisations au niveau de la fonction pour une ressource de machine learning. Les fonctions non conteneurisées doivent hériter des autorisations du propriétaire de la ressource définies sur la ressource de Machine Learning. Pour résoudre ce problème, choisissez d'hériter des autorisations du propriétaire des ressources (console) ou de supprimer les autorisations de la politique d'accès aux ressources (API) de la fonction Lambda.

<function-arn>La fonction fait référence à une ressource Machine Learning dont l'<resource-id>autorisation est manquante à la fois dans la ressource ResourceAccessPolicy et dans la ressource OwnerSetting.

Solution : vous recevez cette erreur si les autorisations d'accès à la ressource d'apprentissage automatique ne sont pas configurées pour la fonction Lambda attachée ou pour la ressource. Pour résoudre ce problème, configurez les autorisations dans la ResourceAccessPolicypropriété de la fonction Lambda ou dans la OwnerSettingpropriété de la ressource.

<function-arn>La fonction fait référence à la ressource Machine Learning <resource-id>avec l'autorisation \ "rw \ », tandis que le paramètre du propriétaire de la ressource autorise GroupPermission uniquement \ "ro \ ».

Solution : vous recevez cette erreur si les autorisations d'accès définies pour la fonction Lambda associée dépassent les autorisations du propriétaire de la ressource définies pour la ressource d'apprentissage automatique. Pour résoudre ce problème, définissez des autorisations plus restrictives pour la fonction Lambda ou des autorisations moins restrictives pour le propriétaire de la ressource.

NoContainer La fonction <function-arn>fait référence aux ressources du chemin de destination imbriqué.

Solution : vous recevez cette erreur si plusieurs ressources de machine learning associées à une fonction Lambda non conteneurisée utilisent le même chemin de destination ou un chemin de destination imbriqué. Pour résoudre ce problème, spécifiez des chemins de destination distincts pour les ressources.

La fonction Lambda <function-arn> accède à la ressource <resource-id> en partageant le même identifiant de propriétaire de groupe

Solution : vous recevez cette erreur `runtime.log` si le même groupe de système d'exploitation est spécifié comme identité Run as de la fonction Lambda et propriétaire de la ressource pour une

ressource d'apprentissage automatique, mais que la ressource n'est pas attachée à la fonction Lambda. Cette configuration donne à la fonction Lambda des autorisations implicites qu'elle peut utiliser pour accéder à la ressource sans AWS IoT Greengrass autorisation.

Pour résoudre ce problème, utilisez un autre groupe de systèmes d'exploitation pour l'une des propriétés ou associez la ressource d'apprentissage automatique à la fonction Lambda.

Consultez aussi

- Exécuter l'inférence de Machine Learning
- the section called “Configuration de l'inférence Machine Learning”
- the section called “Configuration de l'inférence Machine Learning optimisée”
- Référence d’API AWS IoT Greengrass Version 1

Configuration de l'inférence Machine Learning à l'aide d'AWS Management Console

Pour suivre les étapes dans ce didacticiel, vous devezAWS IoT GreengrassCore v1.10 ou version ultérieure.

Vous pouvez exécuter l'inférence Machine Learning (ML) localement sur un appareil principal Greengrass (core) à l'aide des données provenant d'appareils connectés. Pour en savoir plus, notamment sur les exigences et contraintes, consultez la page Exécuter l'inférence de Machine Learning.

Ce didacticiel explique comment utiliser leAWS Management Consolepour configurer un groupe Greengrass afin d'exécuter une application d'inférence Lambda qui reconnaît localement les images d'une caméra, sans avoir à envoyer des données dans le cloud. L'application d'inférence accède au module caméra sur un Raspberry Pi et exécute l'inférence à l'aide de l'open sourceSqueezeNetModèle d'

Le didacticiel contient les étapes détaillées suivantes :

1. Configurer le Raspberry Pi
2. Installer l'infrastructure (framework) MXNet
3. Créer un package de modèle.
4. Créer et publier une fonction Lambda

5. Ajout de la fonction Lambda au groupe
6. Ajouter des ressources au groupe
7. Ajouter un abonnement au groupe
8. Déployer le groupe
9. Tester l'application

Prérequis

Pour suivre ce didacticiel, vous devez disposer des éléments suivants :

- Raspberry Pi 4 Modèle B, ou Raspberry Pi 3 Modèle B/B+, installé et configuré pour une utilisation avecAWS IoT Greengrass. Pour configurer votre Raspberry Pi avec AWS IoT Greengrass, exécutez le script de configuration de l'appareil Greengrass ou assurez-vous d'avoir terminé le Module 1 et le Module 2 de Commencer avec AWS IoT Greengrass.

  > **Note**
  >
  > Le Raspberry Pi peut nécessiter un processeur 2,5 Aalimentationpour exécuter les infrastructures de deep learning généralement utilisées pour la classification des images. Un bloc d'alimentation dont la puissance nominale est inférieure peut provoquer le redémarrage de l'appareil.

- Module caméra V2 du Raspberry Pi - 8 mégapixels, 1080p. Pour plus d'informations surpour configurer l'appareil photo, veuillez consulterConnexion de l'appareil photodans la documentation Raspberry Pi.
- Un groupe Greengrass et un service principal Greengrass. Pour plus d'informations surcomment créer un groupe ou un service principal Greengrass, consultezCommencer avec AWS IoT Greengrass.

> **Note**
>
> Ce didacticiel utilise un Raspberry Pi, mais AWS IoT Greengrass prend en charge d'autres plateformes, telles qu'Intel Atom et NVIDIA Jetson TX2. Dans l'exemple pour Jetson TX2, vous pouvez utiliser des images statiques au lieu d'images diffusées depuis une caméra. Si vous utilisez l'exemple Jetson TX2, vous devrez peut-être installer Python 3.6 au lieu de Python 3.7. Pour de plus amples informations sur la configuration de votre appareilcettevous

pouvez installerAWS IoT GreengrassLogiciel Core, veuillez consulterthe section called “Configuration d'autres appareils”.
Pour les plateformes tierces quiAWS IoT Greengrassne prend pas en charge, vous devez exécuter votre fonction Lambda en mode non conteneurisé. Pour exécuter en mode non conteneurisé, vous devez exécuter votre fonction Lambda en tant qu'utilisateur racine. Pour plus d’informations, consultez the section called “Considérations à prendre en compte lors du choix de la conteneurisation des fonctions Lambda” et the section called “Définition de l'identité d'accès par défaut pour les fonctions Lambda dans un groupe”.

Étape 1 : Configurer le Raspberry Pi

Au cours de cette étape, installez les mises à jour du système d'exploitation Raspbian, installez le logiciel du module caméra et les dépendances Python, et activez l'interface de la caméra.

Dans la fenêtre de terminal de votre Raspberry Pi, exécutez les commandes suivantes :

1. Installez les mises à jour sur Raspbian.

   ```
   sudo apt-get update
   sudo apt-get dist-upgrade
   ```

2. Installez l'interface `picamera` pour le module caméra, ainsi que les autres bibliothèques Python requises pour ce didacticiel.

   ```
   sudo apt-get install -y python3-dev python3-setuptools python3-pip python3-picamera
   ```

   Valider l'installation :

   - Assurez-vous que votre installation Python 3.7 inclut pip.

     ```
     python3 -m pip
     ```

     Si pip n'est pas installé, téléchargez-le à partir du site web pip, puis exécutez la commande suivante.

     ```
     python3 get-pip.py
     ```

   - Assurez-vous que votre Python est en version 3.7 ou supérieure.

```
python3 --version
```

Si la sortie mentionne une version antérieure, exécutez la commande suivante.

```
sudo apt-get install -y python3.7-dev
```

- Assurez-vous que Setuptools et Picamera sont installés correctement.

```
sudo -u ggc_user bash -c 'python3 -c "import setuptools"'
sudo -u ggc_user bash -c 'python3 -c "import picamera"'
```

Si la sortie ne contient pas d'erreurs, la validation a abouti.

> **Note**
>
> Si l'exécutable Python installé sur votre appareil est `python3.7`, utilisez `python3.7` plutôt que `python3` pour les commandes de ce didacticiel. Assurez-vous que votre installation pip correspond à la version correcte (`python3.7` ou `python3`) pour éviter les erreurs de dépendance.

3. Redémarrez l'appareil Raspberry Pi.

```
sudo reboot
```

4. Ouvrez l'outil de configuration du Raspberry Pi.

```
sudo raspi-config
```

5. Utilisez les touches flèches pour ouvrir les Options d'interface et activer l'interface de la caméra. Si vous y êtes invité, autorisez le redémarrage de l'appareil.
6. Utilisez la commande suivante pour tester la configuration de la caméra.

```
raspistill -v -o test.jpg
```

Elle ouvre une fenêtre d'aperçu sur le Raspberry Pi, enregistre une image nommée `test.jpg` dans votre répertoire actuel et affiche des informations sur la caméra dans la fenêtre de terminal du Raspberry Pi.

Étape 2 : Installer l'infrastructure (framework) MXNet

Au cours de cette étape, installez les bibliothèques MXNet sur votre Raspberry Pi.

1. Connectez-vous à votre Raspberry Pi à distance.

   ```
   ssh pi@your-device-ip-address
   ```

2. Ouvrez la documentation MXNet, ouvrez Installing MXNet et suivez les instructions d'installation de MXNet sur l'appareil.

   > **Note**
   >
   > Nous vous recommandons d'installer la version 1.5.0 et de compiler MXNet à partir des sources pour ce didacticiel afin d'éviter les conflits d'appareil.

3. Après avoir installé MXNet, validez la configuration suivante :

   - Assurez-vous que le compte système `ggc_user` peut utiliser le framework MXNet.

     ```
     sudo -u ggc_user bash -c 'python3 -c "import mxnet"'
     ```

   - Make sure NumPy est installé.

     ```
     sudo -u ggc_user bash -c 'python3 -c "import numpy"'
     ```

Étape 3 : Créer un package de modèle MXNet

Au cours de cette étape, créez un package de modèle contenant un exemple de modèle MXNet déjà entraîné à télécharger sur Amazon Simple Storage Service (Amazon S3).AWS IoT Greengrasspeut utiliser un package modèle d'Amazon S3, à condition que vous utilisiez le format tar.gz ou zip.

1. Sur votre ordinateur, téléchargez l'exemple MXNet pour Raspberry Pi à partir de the section called “Exemples de machine learning”.
2. Décompressez le fichier `mxnet-py3-armv7l.tar.gz` téléchargé.
3. Accédez au répertoire `squeezenet`.

   ```
   cd path-to-downloaded-sample/mxnet-py3-armv7l/models/squeezenet
   ```

Le fichier `squeezenet.zip` de ce répertoire est votre package de modèle. Il contient SqueezeNet artefacts de modèle open source pour un modèle de classification d'image. Vous chargerez ultérieurement ce package de modèle sur Amazon S3.

Étape 4 : Créer et publier une fonction Lambda

Au cours de cette étape, créez un package de déploiement de fonction Lambda et une fonction Lambda. Ensuite, publiez une version de fonction et créez un alias.

Tout d'abord, créez le package de déploiement de fonction Lambda.

1. Sur votre ordinateur, accédez au répertoire `examples` de l'exemple de package que vous avez décompressé dans the section called “Créer un package de modèle.”.

   ```
   cd path-to-downloaded-sample/mxnet-py3-armv7l/examples
   ```

   Le répertoire `examples` contient le code de fonction et les dépendances.

   - `greengrassObjectClassification.py` est le code d'inférence utilisé dans ce didacticiel. Vous pouvez utiliser ce code comme modèle pour créer votre propre fonction d'inférence.
   - `greengrasssdk`est la version 1.5.0 duAWS IoT GreengrassKit SDK Core pour Python.

   > **Note**
   >
   > Si une nouvelle version est disponible, vous pouvez la télécharger et mettre à niveau la version du kit SDK dans votre package de déploiement. Pour de plus amples informations, veuillez consulterAWS IoT GreengrassKit SDK Core pour Pythonsur GitHub.

2. Compressez le contenu du répertoire `examples` dans un fichier nommé `greengrassObjectClassification.zip`. Vous obtiendrez alors votre package de déploiement.

   ```
   zip -r greengrassObjectClassification.zip .
   ```

> **Note**
>
> Assurez-vous que les fichiers `.py` et les dépendances se trouvent dans la racine du répertoire.

Créez ensuite la fonction Lambda.

3. De laAWS IoTconsole, choisissezFonctionsetCréation de fonction.
4. ChoisissezCréer à partir de zéroet utilisez les valeurs suivantes pour créer votre fonction :

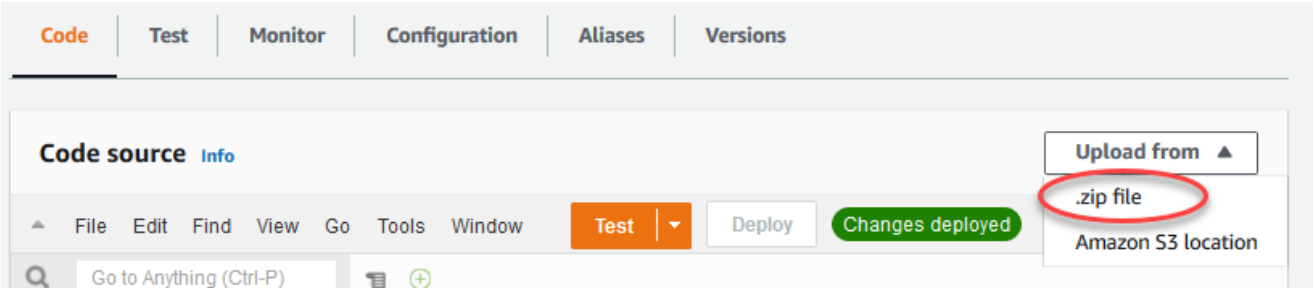
   - Sous Nom de la fonction, saisissez **`greengrassObjectClassification`**.
   - Pour Runtime, sélectionnez Python 3.7.

   PourAutorisations, conservez le paramètre par défaut. Cela crée un rôle d'exécution qui accorde des autorisations Lambda de base. Ce rôle n'est pas utilisé parAWS IoT Greengrass.
5. Sélectionnez Create function (Créer une fonction).

Téléchargez maintenant le package de déploiement de fonction Lambda et enregistrez le gestionnaire.

6. Choisissez votre fonction Lambda et téléchargez le package de déploiement de fonction Lambda.

   a. Dans la pageCodeonglet, sousSource du code, choisissezChargement à partir de. Dans le menu déroulant, choisissezfichier .zip.



b. ChoisissezCharger, puis choisissez votre`greengrassObjectClassification.zip`package de déploiement. Ensuite, choisissez Enregistrer.

c. Dans la pageCodepour la fonction, sousParamètres d'exécution, choisissezModifier, puis entrez les valeurs suivantes.

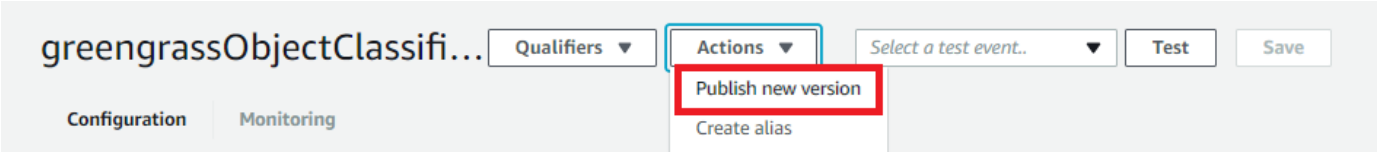
- Pour Runtime, sélectionnez Python 3.7.
- Pour Gestionnaire, entrez **`greengrassObjectClassification.function_handler`**.

Choisissez Save (Enregistrer).

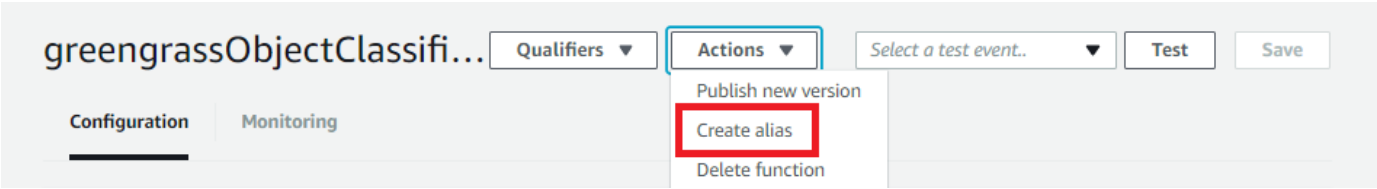
Publiez ensuite la première version de votre fonction Lambda. Puis, créez un alias pour la version.

> **Note**
>
> Les groupes Greengrass peuvent référencer une fonction Lambda par alias (recommandé) ou par version. L'utilisation d'un alias facilite la gestion des mises à jour de code, car vous n'avez pas besoin de changer la table d'abonnement ou la définition de groupe lorsque le code fonction est mis à jour. Au lieu de cela, il vous suffit de pointer l'alias vers la nouvelle version de fonction.

7. Dans le menu Actions, sélectionnez Publier une nouvelle version.



8. Dans Description de la version, saisissez **`First version`**, puis choisissez Publish.
9. Dans la pagegreengrassObjectClassification : 1page de configuration, depuis la pageActions, choisissezCréer un alias.



10. Sur la page Create a new alias, utilisez les valeurs suivantes :

    - Pour Name (Nom), saisissez **`mlTest`**.
    - Pour Version, entrez **`1`**.

    > **Note**
    >
    > AWS IoT Greengrassne prend pas en charge les alias Lambda pour$LATESTversions.

11. Choisissez Save (Enregistrer).

Maintenant, ajoutez la fonction Lambda à votre groupe Greengrass.

Étape 5 : Ajout de la fonction Lambda au groupe Greengrass

Au cours de cette étape, ajoutez la fonction Lambda au groupe, puis configurez son cycle de vie et ses variables d'environnement.

Ajoutez d'abord la fonction Lambda à votre groupe Greengrass.

1. DansAWS IoTVolet de navigation de la console sousGérer, DéveloppezAppareils Greengrass, puis choisissezGroupes (V1).
2. Sur la page de configuration de groupe, choisissez leFonctions LambdaOnglet.
3. SousFonctions My Lambdasection, choisissezAddition.
4. PourFonction Lambda, choisissezgreengrassObjectClassification.
5. PourVersion de la fonction Lambda, choisissezAlias : mltest.

Ensuite, configurez le cycle de vie et les variables d'environnement de la fonction Lambda.

6. Dans la pageConfiguration de fonction Lambda, effectuez les mises à jour suivantes.

    > **Note**
    >
    > Nous vous recommandons d'exécuter votre fonction Lambda sans conteneurisation, à moins que votre analyse de rentabilisation ne l'exige. Cela permet d'accéder au

GPU et à la caméra de votre appareil sans configurer les ressources de l'appareil. Si vous exécutez sans conteneurisation, vous devez également accorder un accès root à votreAWS IoT GreengrassFonctions Lambda.

a. Pour exécuter sans conteneurisation :

- PourUtilisateur et groupe du système, choisissez**`Another user ID/group ID`**. PourID utilisateur du système, saisissez**`0`**. PourID du groupe système, saisissez**`0`**.

  Cela permet à votre fonction Lambda de s'exécuter en tant que root. Pour plus d'informations sur l'exécution en tant que root, consultezthe section called “Définition de l'identité d'accès par défaut pour les fonctions Lambda dans un groupe”.

  > **Tip**
  >
  > Vous devez également mettre à jour votre`config.json`pour accorder à l'accès root à votre fonction Lambda. Pour la procédure, veuillez consulterthe section called “Exécution d'une fonction Lambda en tant que root”.

- PourConteneurisation de fonction Lambda, choisissezAucun conteneur.

  Pour de plus amples informations sur l'exécution sans conteneurisation, veuillez consulterthe section called “Considérations à prendre en compte lors du choix de la conteneurisation des fonctions Lambda”.

- Pour Expiration, entrez **`10 seconds`**.
- PourPinned, choisissezVrai.

  Pour plus d'informations, consultez the section called “Configuration du cycle de vie”.

b. Pour exécuter plutôt en mode conteneurisé :

> **Note**
>
> Nous vous déconseillons une exécution en mode conteneurisé, à moins que votre cas d'utilisation ne l'exige.

- PourUtilisateur et groupe du système, choisissezUtiliser le groupe par défaut.

- PourConteneurisation de fonction Lambda, choisissezUtiliser le groupe par défaut.
- Pour Limite de mémoire, entrez **`96 MB`**.
- Pour Expiration, entrez **`10 seconds`**.
- PourPinned, choisissezVrai.

  Pour plus d'informations, consultez the section called “Configuration du cycle de vie”.

7. Sous Variables d'environnement, créez une paire clé-valeur. Une paire clé-valeur est requise par les fonctions qui interagissent avec les modèles MXNet sur un Raspberry Pi.

   Pour la clé, utilisez MXNET_ENGINE_TYPE. Pour la valeur, utilisezNaiveEngine.

   > **Note**
   >
   > Dans vos propres fonctions Lambda définies par l'utilisateur, vous pouvez, si vous le souhaitez, définir la variable d'environnement dans le code.

8. Conservez les valeurs par défaut pour toutes les autres propriétés et choisissezAjout d'une fonction Lambda.

Étape 6 : Ajouter des ressources au groupe Greengrass

Au cours de cette étape, créez des ressources pour le module de caméra et le modèle d'inférence ML, puis affiliez les ressources à la fonction Lambda. Cela permet à la fonction Lambda d'accéder aux ressources sur l'appareil principal (noyau).

> **Note**
>
> Si vous exécutez en mode non conteneurisé,AWS IoT Greengrasspeut accéder au GPU et à la caméra de votre appareil sans configurer ces ressources.

Tout d'abord, créez deux ressources d'appareil local pour la caméra : un pour la mémoire partagée et l'autre pour l'interface de l'appareil. Pour plus d'informations sur l'accès aux ressources locales, consultez la page Accédez à des ressources locales avec des fonctions et des connecteurs Lambda.

1. Sur la page de configuration du groupe, choisissez leRessourcesOnglet.
2. DansDes ressources localessection, choisissezAjout d'une ressource locale.

3. Dans la pageAjout d'une ressource locale, utilisez les valeurs suivantes :

   - Sous Resource Name (Nom de la ressource), entrez **`videoCoreSharedMemory`**.
   - Pour Type de ressource, choisissez Appareil.
   - PourChemin de l'appareil local, saisissez**`/dev/vcsm`**.

     Le chemin de l'appareil est le chemin absolu local de la ressource d'appareil. Ce chemin ne peut faire référence qu'à un périphérique de caractères ou un périphérique de stockage en mode bloc sous `/dev`.
   - PourPropriétaire du groupe système et autorisations d'accès aux fichiers, choisissezAjouter automatiquement les autorisations de système de fichiers du groupe système qui possède la ressource.

     LePropriétaire du groupe système et autorisations d'accès aux fichiersvous permet d'accorder des autorisations supplémentaires d'accès aux fichiers au processus Lambda. Pour plus d'informations, consultez Autorisation d'accès fichier pour le propriétaire du groupe.
4. Ensuite, vous ajoutez une ressource d'appareil local pour l'interface de la caméra.
5. ChoisissezAjout d'une ressource locale.
6. Dans la pageAjout d'une ressource locale, utilisez les valeurs suivantes :

   - Sous Resource Name (Nom de la ressource), entrez **`videoCoreInterface`**.
   - Pour Type de ressource, choisissez Appareil.
   - PourChemin de l'appareil local, saisissez**`/dev/vchiq`**.
   - PourPropriétaire du groupe système et autorisations d'accès aux fichiers, choisissezAjouter automatiquement les autorisations de système de fichiers du groupe système qui possède la ressource.
7. Au bas de la page, choisissezAjout d'une ressource.

Ajoutez à présent le modèle d'inférence en tant que ressource d'apprentissage automatique. Cette étape inclut le chargement d`usqueezenet.zip`package modèle vers Amazon S3.

1. Dans la pageRessourcespour votre groupe, sous l'ongletMachine Learning (apprentissage automatique), choisissezAjouter une ressource de machine learning.

2. Dans la pageAjouter une ressource d'apprentissage automatiquePage, pourNom de la ressource, saisissez**`squeezenet_model`**.
3. PourSource du modèle, choisissezUtilisez un modèle stocké dans S3, tel qu'un modèle optimisé via Deep Learning Compiler.
4. PourURI S3, entrez le chemin d'accès où le compartiment S3 est enregistré.
5. Choisissez Browse S3 (Parcourir S3). Cette action a pour effet d'ouvrir un nouvel onglet dans la console Amazon S3.
6. Dans l'onglet de la console Amazon S3, chargez le`squeezenet.zip`fichier vers un compartiment S3. Pour plus d'informations, consultezComment charger des fichiers ou des dossiers dans un compartiment S3 ?dans leManuel de l'utilisateur Amazon Simple Storage Service.

   > **Note**
   >
   > Pour que le compartiment S3 soit accessible, le nom de votre compartiment doit contenir la chaîne**`greengrass`**et le compartiment doit se situer dans la même région que celle que vous utilisez pourAWS IoT Greengrass. Choisissez un nom unique (comme **`greengrass-bucket-`*`user-id`*`-`*`epoch-time`**). N'utilisez pas de point (`.`) dans le nom du compartiment.

7. Dans la pageAWS IoT Greengrass, localisez puis choisissez le compartiment S3. Localisez votre fichier chargé `squeezenet.zip`, puis choisissez Sélectionner. Vous devrez peut-être choisir Actualiser pour mettre à jour la liste des compartiments et des fichiers disponibles.
8. Pour Destination path (Chemin d'accès de destination), entrez **`/greengrass-machine-learning/mxnet/squeezenet`**.

   Il s'agit de la destination pour le modèle local dans l'espace de noms d'exécution Lambda. Lorsque vous déployez le groupe, AWS IoT Greengrass permet de récupérer le package de modèle source, puis d'extraire le contenu dans le répertoire spécifié. La fonction Lambda utilisé comme exemple dans ce didacticiel est déjà configurée pour utiliser ce chemin (dans le`model_path`Variable).
9. UNDERPropriétaire du groupe système et autorisations d'accès aux fichiers, choisissezAucun groupe système.
10. Choisissez Add resource (Ajouter ressource).

A l'aide de SageMaker de modèles entraînés

Ce didacticiel utilise un modèle stocké dans Amazon S3, mais vous pouvez utiliser SageMaker de modèles. LeAWS IoT Greengrassla console est intégrée SageMaker . Vous n'avez donc pas besoin de charger manuellement ces modèles dans Amazon S3. Pour connaître les exigences et limitations relatives à l'utilisation SageMaker modèles, voirthe section called “Sources de modèles prises en charge”.

Pour utiliser un SageMaker Modèle d'

- PourSource du modèle, choisissezUtilisez un modèle forméAWS SageMaker, puis choisissez le nom de la tâche de formation du modèle.
- PourChemin de destination, saisissez le chemin d'accès au répertoire dans lequel votre fonction Lambda recherche le modèle.

Étape 7 : Ajouter un abonnement au groupe Greengrass

Au cours des cette étape, ajoutez un abonnement au groupe. Cet abonnement permet à la fonction Lambda d'envoyer les résultats des prédictions àAWS IoTen publiant dans une rubrique MQTT.

1. Sur la page de configuration du groupe, choisissez leSubscriptions, puis choisissezAjouter un abonnement.
2. Dans la pageDétails de l'abonnement, configurez la source et la cible comme suit :
   a. DansType de source, choisissezFonction Lambda, puis choisissezgreengrassObjectClassification.
   b. DansTarget type (Type de cible), choisissezService, puis choisissezCloud IoT.
3. DansFiltre de rubriques, saisissez**`hello/world`**, puis choisissezCréer un abonnement.

Étape 8 : Déploiement du groupe Greengrass

Au cours de cette étape, déployez la version actuelle de la définition de groupe sur l'appareil Greengrass principal (noyau). La définition contient la fonction Lambda, les ressources et les configurations d'abonnement que vous avez ajoutées.

1. Assurez-vous que leAWS IoT Greengrasscore est en cours d'exécution. Dans la fenêtre de terminal de votre Raspberry Pi, exécutez les commandes suivantes, si nécessaire.

a. Pour vérifier si le démon est en cours d'exécution :

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

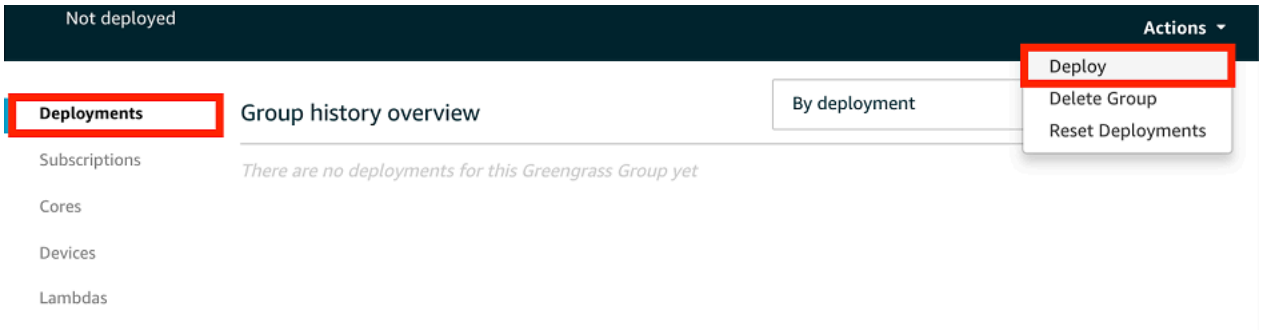
Si la sortie contient une entrée `root` pour `/greengrass/ggc/packages/1.11.6/bin/daemon`, le démon est en cours d'exécution.

> **Note**
>
> La version du chemin d'accès dépend de la version du logiciel AWS IoT Greengrass Core installée sur votre appareil principal.

b. Pour démarrer le démon :

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

2. Sur la page de configuration du groupe, choisissezDéploiement.



3. DansFonctions Lambda, sous l'ongletFonctions Lambda systèmesection, sélectionnezDétecteur IPet choisissezModifier.
4. DansModifier les paramètres du détecteur IPboîte de dialogue, sélectionnezDétecter et remplacer automatiquement les points de terminaison des courtiers MQTT.
5. Choisissez Save (Enregistrer).

Les appareils peuvent ainsi acquérir automatiquement des informations de connectivité pour le noyau, telles que l'adresse IP, le DNS et le numéro de port. La détection automatique est recommandée, mais AWS IoT Greengrass prend également en charge les points de terminaison spécifiés manuellement. Vous êtes uniquement invité à indiquer la méthode de découverte lors du déploiement initial du groupe.

> **Note**
>
> Si vous y êtes invité, autorisez la création duRôle de service Greengrasset associez-le à votreCompte AWSdans les actuelsRégion AWS. Ce rôle permet àAWS IoT Greengrasspour accéder à vos ressources dansAWSServices .

La page Déploiements indique l'horodatage, l'ID de version et l'état du déploiement. Une fois terminé, le statut affiché pour le déploiement doit afficherTerminé.

Pour de plus amples informations sur les déploiements, veuillez consulterDéploiement de groupes AWS IoT Greengrass. Pour bénéficier d'une aide à la résolution des problèmes, consultez Résolution des problèmes.

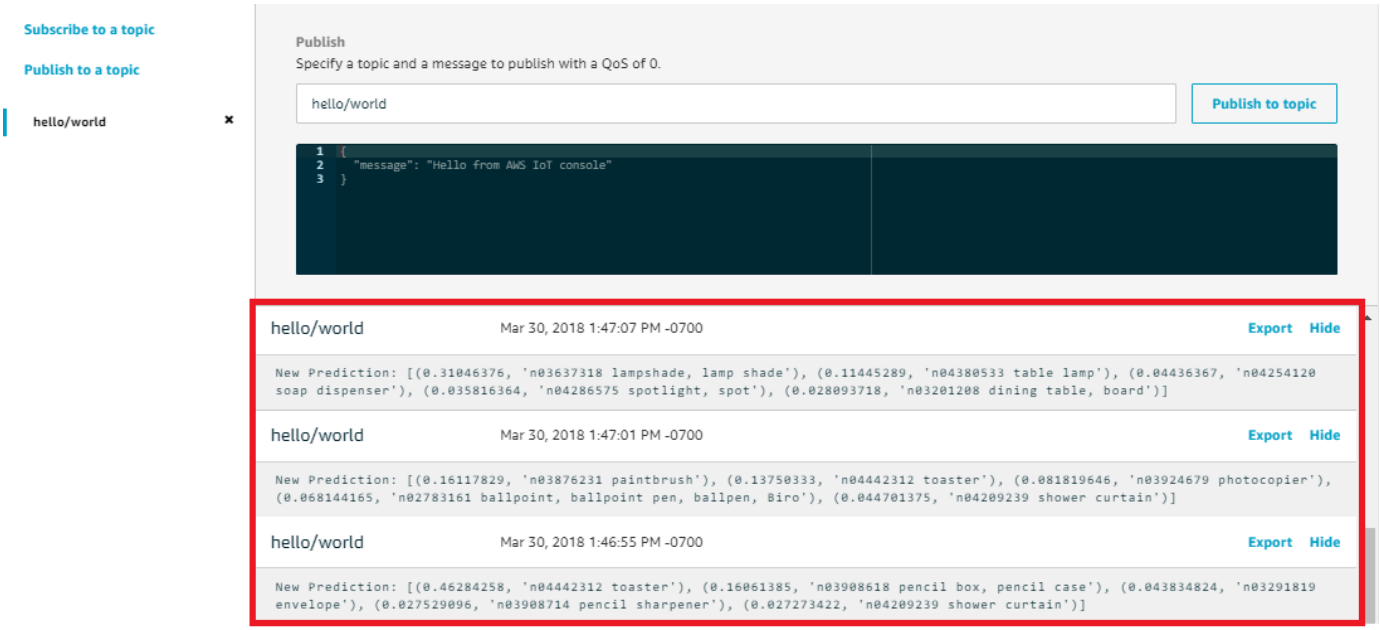
Étape 9 : Tester l'application d'inférence

À présent, vous pouvez vérifier si le déploiement est correctement configuré. Pour effectuer le test, abonnez-vous à la`hello/world`det visualisez les résultats des prédictions publiés par la fonction Lambda.

> **Note**
>
> Si un écran est attaché au Raspberry Pi, le flux de la caméra est diffusé en direct dans une fenêtre d'aperçu.

1. DansAWS IoTconsole, sousTest, choisissezClient de test MQTT.
2. Pour Abonnements, utilisez les valeurs suivantes :
   - Pour la rubrique d'abonnement, utilisez hello/world.
   - UNDERConfiguration supplémentaire, pourAffichage de la charge utile MQTT, choisissezAffichage des charges utiles sous forme de chaînes.
3. Choisissez Subscribe.

   Si le test est réussi, les messages issus de la fonction Lambda apparaissent au bas de la page. Chaque message contient les cinq premiers résultats des prédictions de l'image, en utilisant le format suivant : probabilité, ID de classe prédite et nom de classe correspondante.



Dépannage de l'inférence ML AWS IoT Greengrass

Si le test n'est pas réussi, vous pouvez essayer les étapes de dépannage suivantes. Exécutez les commandes dans la fenêtre de terminal de votre Raspberry Pi.

Consultez les journaux des erreurs

1. Connectez-vous en tant qu'utilisateur racine et accédez au répertoire `log`. L'accès aux journaux AWS IoT Greengrass nécessite les autorisations racines.

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

2. Dans le répertoire `system`, consultez `runtime.log` ou `python_runtime.log`.

   Dans le répertoire `user/`*`region`*`/`*`account-id`*, consultez `greengrassObjectClassification.log`.

   Pour plus d'informations, consultez the section called “Résolution des problèmes liés aux journaux”.

DéballageErreur dansruntime.log

Si `runtime.log` contient une erreur similaire à ce qui suit, assurez-vous que votre package de modèle source `tar.gz` possède un répertoire parent.

```
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
```

Si votre package n'a pas de répertoire parent qui contient les fichiers de modèle, utilisez la commande suivante pourremballageeLe modèle :

```
tar -zcvf model.tar.gz ./model
```

Par exemple :

```
#$ tar -zcvf test.tar.gz ./test
./test
./test/some.file
./test/some.file2
./test/some.file3
```

> **Note**
>
> N'incluez pas les caractères de fin `/*` dans cette commande.

Vérifiez que la fonction Lambda a été déployée avec succès

1. Affichez le contenu de Lambda déployée dans la section`/lambda`répertoire. Remplacez les valeurs d'espace réservé avant d'exécuter la commande.

   ```
   cd /greengrass/ggc/deployment/lambda/
   arn:aws:lambda:region:account:function:function-name:function-version
   ls -la
   ```

2. Vérifiez que le répertoire contient le même contenu que le package de déploiement `greengrassObjectClassification.zip` que vous avez chargé dans Étape 4 : Créer et publier une fonction Lambda.

   Assurez-vous que les fichiers `.py` et les dépendances se trouvent dans la racine du répertoire.

Vérifiez que le modèle d'inférence a été déployé avec succès

1. Trouvez le numéro d'identification de processus (PID) du processus d'exécution Lambda :

   ```
   ps aux | grep 'lambda-function-name*'
   ```

   Dans la sortie, le PID s'affiche dans la deuxième colonne de la ligne pour le processus d'exécution Lambda.

2. Entrez l'espace de noms d'exécution Lambda. Veillez à remplacer la valeur d'espace réservé *pid* avant d'exécuter la commande.

   > **Note**
   >
   > Ce répertoire et son contenu se trouvent dans l'espace de noms d'exécution Lambda ; par conséquent, ils ne sont pas visibles dans un espace de noms Linux classique.

   ```
   sudo nsenter -t pid -m /bin/bash
   ```

3. Affichez le contenu du répertoire local que vous avez spécifié comme ressource de ML.

   ```
   cd /greengrass-machine-learning/mxnet/squeezenet/
   ls -ls
   ```

   Vous devriez voir les fichiers suivants :

   ```
   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
   ```

Étapes suivantes

Ensuite, explorez les autres applications d'inférence.AWS IoT Greengrasspropose d'autres fonctions Lambda que vous pouvez utiliser pour tester l'inférence locale. Vous pouvez trouver les packages d'exemple dans le dossier des bibliothèques précompilées que vous avez téléchargé à l'the section called “Installer l'infrastructure (framework) MXNet”.

Configuration d'un processeur Intel Atom

Pour exécuter ce didacticiel sur un appareil Intel Atom, vous devez fournir des images source, configurer la fonction Lambda et ajouter une autre ressource d'appareil local. Pour utiliser le processeur graphique à des fins d'inférence, assurez-vous que les logiciels suivants sont installés sur votre appareil :

- OpenCL version 1.0 ou ultérieure
- Python 3.7 et pip

> **Note**
>
> Si votre appareil est préconfiguré avec Python 3.6, vous pouvez aussi créer un lien symbolique vers Python 3.7. Pour plus d'informations, consultez Step 2.

- NumPy
- OpenCV sur Wheels

1. Téléchargez les images statiques PNG ou JPG que la fonction Lambda utilisera pour la classification d'images. L'exemple fonctionne mieux avec des fichiers image de petite taille.

   Enregistrez vos fichiers image dans le répertoire qui contient le fichier `greengrassObjectClassification.py` (ou dans un sous-répertoire de ce répertoire). Cela fait partie du package de déploiement de fonction Lambda que vous chargez dansthe section called “Créer et publier une fonction Lambda”.

   > **Note**
   >
   > Si vous utilisez AWS DeepLens, vous pouvez utiliser la caméra embarquée ou monter votre propre caméra pour effectuer des inférences sur des images capturées plutôt

que sur des images statiques. Cependant, nous vous recommandons vivement de commencer par les images statiques.
Si vous utilisez une caméra, assurez-vous que le package APT `awscam` est installé et à jour. Pour de plus amples informations, veuillez consulterMettre à jour votreAWS DeepLensappareildans leAWS DeepLensManuel du développeur.

2. Si vous n'utilisez pas Python 3.7, assurez-vous de créer un lien symbolique de Python 3.x vers Python 3.7. Cela permet de configurer votre appareil pour utiliser Python 3 avec AWS IoT Greengrass. Exécutez la commande suivante pour localiser votre installation Python :

```
which python3
```

Exécutez la commande suivante pour créer le lien symbolique :

```
sudo ln -s path-to-python-3.x/python3.x path-to-python-3.7/python3.7
```

Redémarrez l'appareil.

3. Modifiez la configuration de la fonction Lambda. Suivez la procédure décrite dans the section called “Ajout de la fonction Lambda au groupe”.

> **Note**
>
> Nous vous recommandons d'exécuter votre fonction Lambda sans conteneurisation, à moins que votre analyse de rentabilisation ne l'exige. Cela permet d'accéder au GPU et à la caméra de votre appareil sans configurer les ressources de l'appareil. Si vous exécutez sans conteneurisation, vous devez également accorder un accès root à votreAWS IoT GreengrassFonctions Lambda.

a. Pour exécuter sans conteneurisation :

- PourUtilisateur et groupe du système, choisissez**`Another user ID/group ID`**. PourID utilisateur du système, saisissez**`0`**. PourID de groupe de systèmes, saisissez**`0`**.

  Cela permet à votre fonction Lambda de s'exécuter en tant que root. Pour plus d'informations sur l'exécution en tant que root, consultezthe section called “Définition de l'identité d'accès par défaut pour les fonctions Lambda dans un groupe”.

> **Tip**
>
> Vous devez également mettre à jour votre`config.json`pour accorder à l'accès root à votre fonction Lambda. Pour la procédure, veuillez consulterthe section called “Exécution d'une fonction Lambda en tant que root”.

- PourConteneurisation de fonction Lambda, choisissezAucun conteneur.

  Pour de plus amples informations sur l'exécution sans conteneurisation, veuillez consulterthe section called “Considérations à prendre en compte lors du choix de la conteneurisation des fonctions Lambda”.
- Mettez à jour la valeur Délai d'expiration à 5 secondes. Cela permet de s'assurer que la demande n'expire pas trop tôt. L'exécution de l'inférence nécessite quelques minutes après la configuration.
- UNDERPinned, choisissezVrai.
- UNDERParamètres supplémentaires, pourAccès en lecture au répertoire /sys, choisissezActivé.
- Sous Cycle de vie Lambda, sélectionnez Attribuer une longue durée de vie à cette fonction et l'exécuter indéfiniment.

b. Pour exécuter plutôt en mode conteneurisé :

> **Note**
>
> Nous vous déconseillons une exécution en mode conteneurisé, à moins que votre cas d'utilisation ne l'exige.

- Mettez à jour la valeur Délai d'expiration à 5 secondes. Cela permet de s'assurer que la demande n'expire pas trop tôt. L'exécution de l'inférence nécessite quelques minutes après la configuration.
- PourPinned, choisissezVrai.
- UNDERParamètres supplémentaires, pourAccès en lecture au répertoire /sys, choisissezActivé.

4. En cas d'exécution en mode conteneurisé, ajoutez la ressource locale requise pour autoriser l'accès au GPU de votre appareil.

> **Note**
>
> Si vous exécutez en mode non conteneurisé,AWS IoT Greengrasspeut accéder au GPU de votre appareil sans configurer les ressources de l'appareil.

a. Sur la page de configuration du groupe, choisissez leRessourcesOnglet.

b. ChoisissezAjout d'une ressource locale.

c. Définissez la ressource :

- Sous Resource Name (Nom de la ressource), entrez **`renderD128`**.
- PourType de ressource, choisissezAppareil local.
- Pour Device path (Chemin de l'appareil), entrez **`/dev/dri/renderD128`**.
- PourPropriétaire du groupe système et autorisations d'accès aux fichiers, choisissezAjouter automatiquement les autorisations de système de fichiers du groupe système qui possède la ressource.
- PourAffiliations aux fonctions Lambda, octroiAccès en lecture et en écritureà votre fonction Lambda.

Configuration d'un NVIDIA Jetson TX2

Pour exécuter ce didacticiel sur un NVIDIA Jetson TX2, fournissez des images source et configurez la fonction Lambda. Si vous utilisez le processeur graphique, vous devez également ajouter des ressources de périphérique local.

1. Assurez-vous que votre appareil Jetson est configuré pour que vous puissiez installer le logiciel AWS IoT Greengrass Core. Pour de plus amples informations sur la configuration de votre appareil, veuillez consulter the section called “Configuration d'autres appareils”.
2. Ouvrez la documentation MXNet, accédez à Installing MXNet on a Jetson et suivez les instructions d'installation de MXNet sur l'appareil Jetson.

> **Note**
>
> Si vous souhaitez créer MXNet à partir de la source, suivez les instructions de création de la bibliothèque partagée. Modifiez les paramètres suivants dans votre fichier `config.mk` pour utiliser un appareil Jetson TX2 :
>
> - Ajoutez `-gencode arch=compute-62, code=sm_62` au paramètre `CUDA_ARCH`.
> - Activez CUDA.
>
> ```
> USE_CUDA = 1
> ```

3. Téléchargez les images statiques PNG ou JPG que la fonction Lambda utilisera pour la classification d'images. L'application fonctionne mieux avec des fichiers image de petite taille. Vous pouvez également instrumenter une caméra sur le plateau Jetson pour capturer les images source.

   Enregistrez vos fichiers image dans le répertoire contenant le fichier `greengrassObjectClassification.py`. Vous pouvez également les enregistrer dans un sous-répertoire de ce répertoire. Ce répertoire fait partie du package de déploiement de fonction Lambda que vous chargez dansthe section called “Créer et publier une fonction Lambda”.

4. Créez un lien symbolique de Python 3.7 vers Python 3.6 pour utiliser Python 3 avec AWS IoT Greengrass. Exécutez la commande suivante pour localiser votre installation Python :

   ```
   which python3
   ```

   Exécutez la commande suivante pour créer le lien symbolique :

   ```
   sudo ln -s path-to-python-3.6/python3.6 path-to-python-3.7/python3.7
   ```

   Redémarrez l'appareil.

5. Assurez-vous que le compte système `ggc_user` peut utiliser le framework MXNet :

   ```
   “sudo -u ggc_user bash -c 'python3 -c "import mxnet"'
   ```

6. Modifiez la configuration de la fonction Lambda. Suivez la procédure décrite dans the section called “Ajout de la fonction Lambda au groupe”.

> **Note**
>
> Nous vous recommandons d'exécuter votre fonction Lambda sans conteneurisation, à moins que votre analyse de rentabilisation ne l'exige. Cela permet d'accéder au GPU et à la caméra de votre appareil sans configurer les ressources de l'appareil. Si vous exécutez sans conteneurisation, vous devez également accorder un accès root à votreAWS IoT GreengrassFonctions Lambda.

a. Pour exécuter sans conteneurisation :

- PourUtilisateur et groupe du système, choisissez**`Another user ID/group ID`**. PourID utilisateur du système, saisissez**`0`**. PourID de groupe de systèmes, saisissez**`0`**.

  Cela permet à votre fonction Lambda de s'exécuter en tant que root. Pour plus d'informations sur l'exécution en tant que root, consultezthe section called “Définition de l'identité d'accès par défaut pour les fonctions Lambda dans un groupe”.

  > **Tip**
  >
  > Vous devez également mettre à jour votre`config.json`pour accorder à l'accès root à votre fonction Lambda. Pour la procédure, veuillez consulterthe section called “Exécution d'une fonction Lambda en tant que root”.

- PourConteneurisation de fonction Lambda, choisissezAucun conteneur.

  Pour de plus amples informations sur l'exécution sans conteneurisation, veuillez consulterthe section called “Considérations à prendre en compte lors du choix de la conteneurisation des fonctions Lambda”.

- UNDERParamètres supplémentaires, pourAccès en lecture au répertoire /sys, choisissezActivé.
- UNDEREnvironment variables (Variables d'environnement), ajoutez les paires clé-valeur suivantes à votre fonction Lambda. Cela permet de configurer AWS IoT Greengrass pour utiliser le framework MXNet.

Clé	Valeur
PATH	/usr/local/cuda/bin:$PATH
MXNET_HOME	$HOME/mxnet/
PYTHONPATH	$MXNET_HOME/python:$PYTHONPATH
CUDA_HOME	/usr/local/cuda
LD_LIBRARY_PATH	$LD_LIBRARY_PATH:${CUDA_HOME}/lib64

b. Pour exécuter plutôt en mode conteneurisé :

> **Note**
>
> Nous vous déconseillons une exécution en mode conteneurisé, à moins que votre cas d'utilisation ne l'exige.

- Augmentez la valeur Limite de mémoire. Utilisez 500 Mo pour le processeur ou au moins 2000 Mo pour le processeur graphique.
- UNDERParamètres supplémentaires, pourAccès en lecture au répertoire /sys, choisissezActivé.
- UNDEREnvironment variables (Variables d'environnement), ajoutez les paires clé-valeur suivantes à votre fonction Lambda. Cela permet de configurer AWS IoT Greengrass pour utiliser le framework MXNet.

Clé	Valeur
PATH	/usr/local/cuda/bin:$PATH
MXNET_HOME	$HOME/mxnet/
PYTHONPATH	$MXNET_HOME/python:$PYTHONPATH

Clé	Valeur
CUDA_HOME	/usr/local/cuda
LD_LIBRARY_PATH	$LD_LIBRARY_PATH:${CUDA_HOME}/lib64

7. En cas d'exécution en mode conteneurisé, ajoutez les ressources locales suivantes pour autoriser l'accès au GPU de votre appareil. Suivez la procédure décrite dans the section called “Ajouter des ressources au groupe”.

   > **Note**
   >
   > Si vous exécutez en mode non conteneurisé,AWS IoT Greengrasspeut accéder au GPU de votre appareil sans configurer les ressources de l'appareil.

   Pour chaque ressource :

   - Pour Type de ressource, choisissez Appareil.
   - PourPropriétaire du groupe système et autorisations d'accès aux fichiers, choisissezAjouter automatiquement les autorisations de système de fichiers du groupe système qui possède la ressource.

Nom	Chemin de l'appareil
nvhost-ctrl	/dev/nvhost-ctrl
nvhost-gpu	/dev/nvhost-gpu
nvhost-ctrl-gpu	/devdevdev/nvhost-ctrl-gpu
nvhost-dbg-gpu	/devdevdev/nvhost-dbg-gpu
nvhost-prof-gpu	/devdevdev/nvhost-prof-gpu
nvmap	/dev/nvmap

Nom	Chemin de l'appareil
nvhost-vic	/dev/nvhost-vic
tegra_dc_ctrl	/dev/tegra_dc_ctrl

8. En cas d'exécution en mode conteneurisé, ajoutez la ressource de volume local suivante pour autoriser l'accès à l'appareil photo de votre appareil. Suivez la procédure décrite dans the section called “Ajouter des ressources au groupe”.

   > **Note**
   >
   > Si vous exécutez en mode non conteneurisé,AWS IoT Greengrasspeut accéder à l'appareil photo de votre appareil sans configurer les ressources de volume.

   - Pour Type de ressource, choisissez Volume.
   - PourPropriétaire du groupe système et autorisations d'accès aux fichiers, choisissezAjouter automatiquement les autorisations de système de fichiers du groupe système qui possède la ressource.

Nom	Chemin source	Chemin de destination
shm	/dev/shm	/dev/shm
tmp	/tmp	/tmp

Configuration de l'inférence Machine Learning optimisée à l'aide d'AWS Management Console

Pour suivre les étapes dans ce didacticiel, vous devez utiliserAWS IoT GreengrassCore v1.10 ou version ultérieure.

Vous pouvez utiliser le plugin SageMaker Compilateur Neo Deep Learning pour optimiser l'efficacité de prédiction des modèles natifs d'inférence Machine Learning dans Tensorflow, PyTorch, ONNX

et XGBoost pour un encombrement réduit et des performances plus rapides. Vous pouvez ensuite télécharger le modèle optimisé et installer le SageMaker Neo Deep Learning Service (core) et déployez-les dans votreAWS IoT Greengrassappareils pour une inférence plus rapide.

Ce didacticiel explique comment utiliser leAWS Management Consolepour configurer un groupe Greengrass afin d'exécuter un exemple d'inférence Lambda qui reconnaît localement les images d'une caméra, sans avoir à envoyer des données dans le cloud. L'exemple d'inférence accède au module de caméra sur un Raspberry Pi. Dans ce didacticiel, vous téléchargez un modèle préemballé formé par Resnet C-50 et optimisé dans le compilateur Neo Deep Learning. Ensuite, vous utilisez le modèle pour effectuer une classification locale d'images sur votre appareil AWS IoT Greengrass.

Le didacticiel contient les étapes détaillées suivantes :

1. Configurer le Raspberry Pi
2. Installer l'environnement d'exécution Neo Deep Learning
3. Créer une fonction Lambda d'inférence
4. Ajouter la fonction Lambda au groupe
5. Ajouter une ressource de modèle optimisé pour Neo au groupe
6. Ajouter la ressource d'appareil de votre caméra au groupe
7. Ajouter des abonnements au groupe
8. Déployer le groupe
9. Tester l'exemple

Prérequis

Pour suivre ce didacticiel, vous devez disposer des éléments suivants :

- Raspberry Pi 4 Modèle B, ou Raspberry Pi 3 Modèle B/N +, installé et configuré pour une utilisation avecAWS IoT Greengrass. Pour configurer votre Raspberry Pi avec AWS IoT Greengrass, exécutez le script de configuration de l'appareil Greengrass ou assurez-vous d'avoir terminé le Module 1 et le Module 2 de Commencer avec AWS IoT Greengrass.

  > **Note**
  >
  > Le Raspberry Pi peut nécessiter un processeur 2,5 Aalimentationpour exécuter les infrastructures de deep learning généralement utilisées pour la classification des images.

Un bloc d'alimentation dont la puissance nominale est inférieure peut provoquer le redémarrage de l'appareil.

- Module caméra V2 du Raspberry Pi - 8 mégapixels, 1080p. Pour savoir comment configurer la caméra, consultez la section Connexion de la caméra dans la documentation du Raspberry Pi.
- Un groupe Greengrass et un service principal Greengrass. Pour savoir comment créer un groupe ou un service principal Greengrass, consultez Commencer avec AWS IoT Greengrass.

> **Note**
>
> Ce didacticiel utilise un Raspberry Pi, mais AWS IoT Greengrass prend en charge d'autres plateformes, telles qu'Intel Atom et NVIDIA Jetson TX2. Si vous utilisez l'exemple Intel Atom, vous devrez peut-être installer Python 3.6 au lieu de Python 3.7. Pour de plus amples informations sur la configuration de votre appareil afin de pouvoir installer le logiciel AWS IoT Greengrass Core, veuillez consulter the section called “Configuration d'autres appareils”. Pour les plateformes tierces quiAWS IoT Greengrassne prend pas en charge, vous devez exécuter votre fonction Lambda en mode non conteneurisé. Pour exécuter en mode non conteneurisé, vous devez exécuter votre fonction Lambda en tant qu'utilisateur racine. Pour plus d’informations, consultez the section called “Considérations à prendre en compte lors du choix de la conteneurisation des fonctions Lambda” et the section called “Définition de l'identité d'accès par défaut pour les fonctions Lambda dans un groupe”.

Étape 1 : Configurer le Raspberry Pi

Au cours de cette étape, installez les mises à jour du système d'exploitation Raspbian, installez le logiciel du module caméra et les dépendances Python, et activez l'interface de la caméra.

Dans la fenêtre de terminal de votre Raspberry Pi, exécutez les commandes suivantes :

1. Installez les mises à jour sur Raspbian.

   ```
   sudo apt-get update
   sudo apt-get dist-upgrade
   ```

2. Installez l'interface `picamera` pour le module caméra, ainsi que les autres bibliothèques Python requises pour ce didacticiel.

```
sudo apt-get install -y python3-dev python3-setuptools python3-pip python3-picamera
```

Valider l'installation :

- Assurez-vous que votre installation Python 3.7 inclut pip.

  ```
  python3 -m pip
  ```

  Si pip n'est pas installé, téléchargez-le à partir du site web pip, puis exécutez la commande suivante.

  ```
  python3 get-pip.py
  ```

- Assurez-vous que votre Python est en version 3.7 ou supérieure.

  ```
  python3 --version
  ```

  Si la sortie mentionne une version antérieure, exécutez la commande suivante.

  ```
  sudo apt-get install -y python3.7-dev
  ```

- Assurez-vous que Setuptools et Picamera sont installés correctement.

  ```
  sudo -u ggc_user bash -c 'python3 -c "import setuptools"'
  sudo -u ggc_user bash -c 'python3 -c "import picamera"'
  ```

  Si la sortie ne contient pas d'erreurs, la validation a abouti.

> **Note**
>
> Si l'exécutable Python installé sur votre appareil est `python3.7`, utilisez `python3.7` plutôt que `python3` pour les commandes de ce didacticiel. Assurez-vous que votre installation pip correspond à la version correcte (`python3.7` ou `python3`) pour éviter les erreurs de dépendance.

3. Redémarrez l'appareil Raspberry Pi.

```
sudo reboot
```

4. Ouvrez l'outil de configuration du Raspberry Pi.

   ```
   sudo raspi-config
   ```

5. Utilisez les touches flèches pour ouvrir les Options d'interface et activer l'interface de la caméra. Si vous y êtes invité, autorisez le redémarrage de l'appareil.
6. Utilisez la commande suivante pour tester la configuration de la caméra.

   ```
   raspistill -v -o test.jpg
   ```

   Elle ouvre une fenêtre d'aperçu sur le Raspberry Pi, enregistre une image nommée `test.jpg` dans votre répertoire actuel et affiche des informations sur la caméra dans la fenêtre de terminal du Raspberry Pi.

Étape 2 : Installer Amazon SageMaker Exécution Neo deep learning

Au cours de cette étape, installez le moteur d'exécution Deep Learning (DLR) sur votre Raspberry Pi.

> **Note**
>
> Nous vous recommandons d'installer la version 1.1.0 pour ce tutoriel.

1. Connectez-vous à votre Raspberry Pi à distance.

   ```
   ssh pi@your-device-ip-address
   ```

2. Ouvrez la documentation DLR, ouvrez Installation du DLR et recherchez l'URL du Wheel des appareils Raspberry Pi. Suivez ensuite les instructions d'installation du DLR sur votre appareil. Par exemple, vous pouvez utiliser pip :

   ```
   pip3 install rasp3b-wheel-url
   ```

3. Après avoir installé le DLR, validez la configuration suivante :

   - Assurez-vous que le compte système `ggc_user` peut utiliser la bibliothèque DLR.

```
sudo -u ggc_user bash -c 'python3 -c "import dlr"'
```

- Vérifiez les NumPy est installé.

```
sudo -u ggc_user bash -c 'python3 -c "import numpy"'
```

Étape 3 : Créer une fonction Lambda d'inférence

Au cours de cette étape, créez un package de déploiement de fonction Lambda et une fonction Lambda. Ensuite, publiez une version de fonction et créez un alias.

1. Sur votre ordinateur, téléchargez l'exemple DLR pour Raspberry Pi à partir de the section called “Exemples de machine learning”.
2. Décompressez le fichier `dlr-py3-armv7l.tar.gz` téléchargé.

   ```
   cd path-to-downloaded-sample
   tar -xvzf dlr-py3-armv7l.tar.gz
   ```

   Le répertoire `examples` de l'exemple de package extrait contient le code de fonction et les dépendances.

   - `inference.py` est le code d'inférence utilisé dans ce didacticiel. Vous pouvez utiliser ce code comme modèle pour créer votre propre fonction d'inférence.
   - `greengrasssdk`est la version 1.5.0 duAWS IoT GreengrassKit SDK principal pour Python.

   > **Note**
   >
   > Si une nouvelle version est disponible, vous pouvez la télécharger et mettre à niveau la version du kit SDK dans votre package de déploiement. Pour de plus amples informations, veuillez consulterAWS IoT GreengrassKit SDK principal pour Pythonsur GitHub.

3. Compressez le contenu du répertoire `examples` dans un fichier nommé `optimizedImageClassification.zip`. Vous obtiendrez alors votre package de déploiement.

```
cd path-to-downloaded-sample/dlr-py3-armv7l/examples
zip -r optimizedImageClassification.zip .
```

Le package de déploiement contient votre code de fonction et vos dépendances. Cela inclut le code qui appelle les API Python du moteur d'exécution Deep Learning afin d'effectuer des inférences avec les modèles de compilation Neo Deep Learning.

> **Note**
>
> Assurez-vous que les fichiers `.py` et les dépendances se trouvent dans la racine du répertoire.

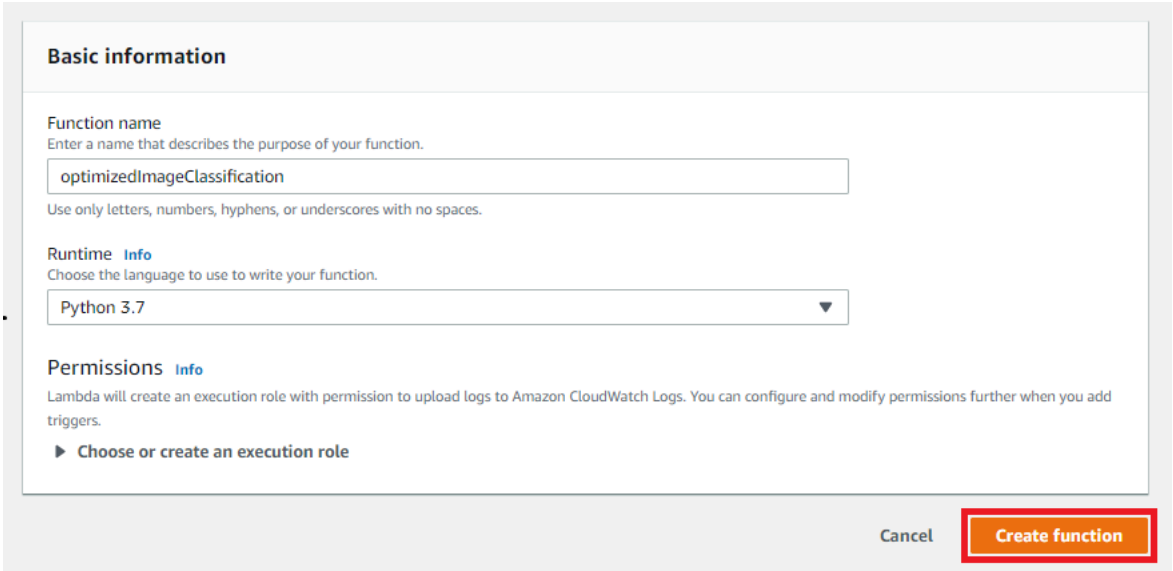
4. Ajoutez maintenant la fonction Lambda à votre groupe Greengrass.

   Sur la page de la console Lambda, choisissezFonctionset choisissezCréation de fonction.

5. ChoisissezCréer à partir de zéroet utilisez les valeurs suivantes pour créer votre fonction :

   - Sous Nom de la fonction, saisissez **`optimizedImageClassification`**.
   - Pour Runtime, sélectionnez Python 3.7.

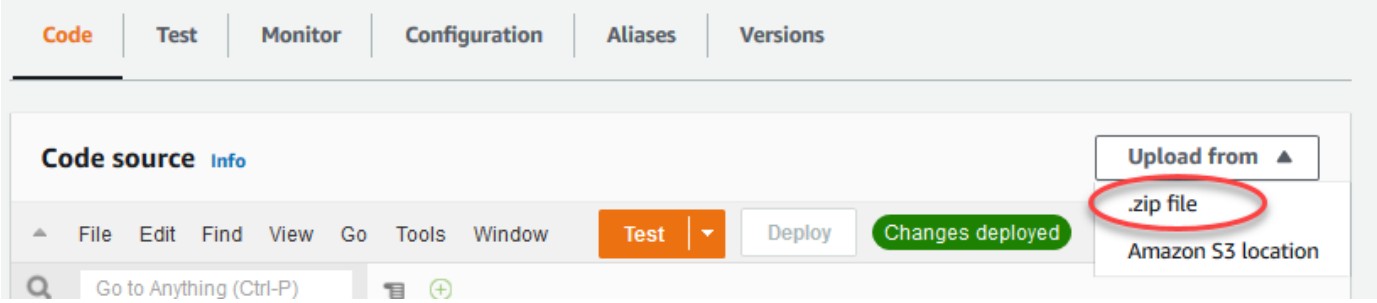
   PourAutorisations, conservez le paramètre par défaut. Cela crée un rôle d'exécution qui accorde des autorisations Lambda de base. Ce rôle n'est pas utilisé parAWS IoT Greengrass.



6. Sélectionnez Create function (Créer une fonction).

Téléchargez maintenant le package de déploiement de fonction Lambda et enregistrez le gestionnaire.

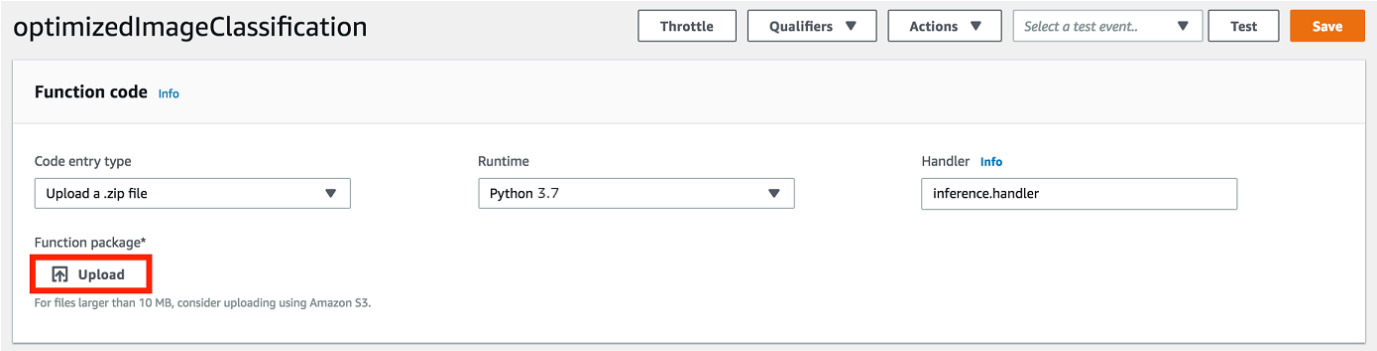
1. Dans la pageCodeonglet, sousCode source, choisissezChargement à partir de. Dans le menu déroulant, choisissezfichier .zip.



2. Choisissez votre`optimizedImageClassification.zip`package de déploiement, puis choisissezEnregistrer.
3. Dans la pageCodepour la fonction, sousParamètres d'exécution, choisissezModifier, puis entrez les valeurs suivantes.

   - Pour Runtime, sélectionnez Python 3.7.
   - Pour Gestionnaire, entrez **`inference.handler`**.

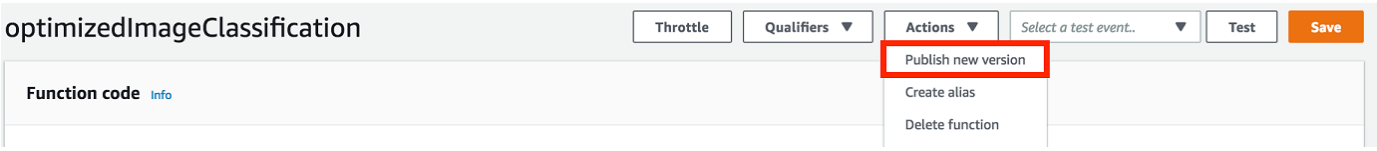
   Choisissez Save (Enregistrer).



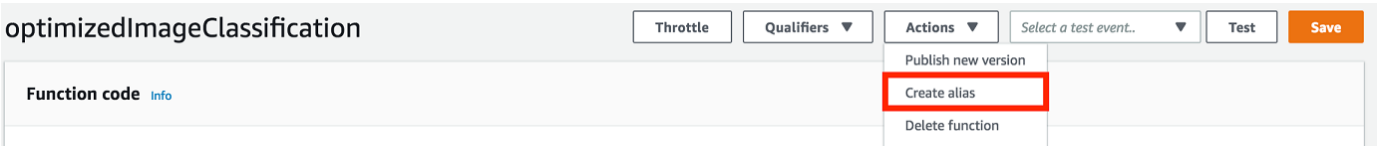
Publiez ensuite la première version de votre fonction Lambda. Puis, créez un alias pour la version.

> **Note**
>
> Les groupes Greengrass peuvent référencer une fonction Lambda par alias (recommandé) ou par version. L'utilisation d'un alias facilite la gestion des mises à jour de code, car vous n'avez pas besoin de changer votre table d'abonnement ou définition de groupe lorsque le code fonction est mis à jour. Il vous suffit de pointer l'alias vers la nouvelle version de fonction.

1. Dans le menu Actions, sélectionnez Publier une nouvelle version.



2. Dans Description de la version, saisissez **`First version`**, puis choisissez Publish.
3. Dans la pageoptimizedImageClassification : 1page de configuration, depuis la pageActions, choisissezCréer un alias.



4. Sur la page Create a new alias, utilisez les valeurs suivantes :

   - Pour Name (Nom), saisissez **`mlTestOpt`**.
   - Pour Version, entrez **`1`**.

   > **Note**
   >
   > AWS IoT Greengrassne prend pas en charge les alias Lambda pour$LATESTversions.

5. Sélectionnez Create (Créer).

   Ajoutez maintenant la fonction Lambda à votre groupe Greengrass.

Étape 4 : Ajouter la fonction Lambda au groupe Greengrass

Au cours de cette étape, ajoutez la fonction Lambda au groupe, puis configurez son cycle de vie.

Ajoutez d'abord la fonction Lambda à votre groupe Greengrass.

1. DansAWS IoTVolet de navigation de la console sousGérer, développezAppareils Greengrasset choisissezGroupes (V1).
2. Sur la page de configuration de groupe, choisissez leFonctions Lambdaonglet, puis choisissezAddition.
3. Cliquez sur l'ongletFonction Lambdaet sélectionnezoptimizedImageClassification.
4. Dans la pageVersion de la fonction Lambda, choisissez l'alias de la version que vous avez publiée.

Configurez ensuite le cycle de vie de la fonction Lambda.

1. DansConfiguration des fonctions Lambda, effectuez les mises à jour suivantes.

   > **Note**
   >
   > Nous vous recommandons d'exécuter votre fonction Lambda sans conteneurisation, à moins que votre analyse de rentabilisation ne l'exige. Cela permet d'accéder au GPU et à la caméra de votre appareil sans configurer les ressources de l'appareil. Si vous exécutez sans conteneurisation, vous devez également accorder un accès root à votreAWS IoT GreengrassFonctions Lambda.

   a. Pour exécuter sans conteneurisation :

      - PourUtilisateur et groupe du système, choisissez**`Another user ID/group ID`**. PourID utilisateur du système, saisissez**`0`**. PourID du groupe du système, saisissez**`0`**.

        Cela permet à votre fonction Lambda de s'exécuter en tant que root. Pour plus d'informations sur l'exécution en tant que root, veuillez consulterthe section called “Définition de l'identité d'accès par défaut pour les fonctions Lambda dans un groupe”.

> **Tip**
>
> Vous devez également mettre à jour votre`config.json`pour accorder à l'accès root à votre fonction Lambda. Pour la procédure, veuillez consulterthe section called “Exécution d'une fonction Lambda en tant que root”.

- PourConteneurisation de fonction Lambda, choisissezAucun conteneur.

  Pour de plus amples informations sur l'exécution sans conteneurisation, veuillez consulterthe section called “Considérations à prendre en compte lors du choix de la conteneurisation des fonctions Lambda”.
- Pour Expiration, entrez **`10 seconds`**.
- PourPinned, choisissezVrai.

  Pour plus d'informations, consultez the section called “Configuration du cycle de vie”.
- UNParamètre supplémentaire, pourAccès en lecture au répertoire /sys, choisissezActivé.

b. Pour exécuter plutôt en mode conteneurisé :

> **Note**
>
> Nous ne recommandons pas une exécution en mode conteneurisé, à moins que votre cas d'utilisation ne l'exige.

- PourUtilisateur et groupe du système, choisissezUtiliser le groupe par défaut.
- PourConteneurisation de fonction Lambda, choisissezUtiliser le groupe par défaut.
- Pour Limite de mémoire, entrez **`1024 MB`**.
- Pour Expiration, entrez **`10 seconds`**.
- PourPinned, choisissezVrai.

  Pour plus d'informations, consultez the section called “Configuration du cycle de vie”.
- UNParamètres supplémentaires, pourAccès en lecture au répertoire /sys, choisissezActivé.

2. ChoisissezAjout d'une fonction Lambda.

Étape 5 : Ajout d'un SageMaker Ressources de modèle optimisé pour le groupe Greengrass

Au cours de cette étape, créez une ressource pour le modèle d'inférence ML optimisé et chargez-le dans un compartiment Amazon S3. Ensuite, recherchez le modèle chargé Amazon S3 dans la sectionAWS IoT Greengrasset associez la nouvelle ressource créée à la fonction Lambda. Cela permet à la fonction d'accéder à ses ressources sur l'appareil principal.

1. Sur votre ordinateur, accédez au répertoire `resnet50` de l'exemple de package que vous avez décompressé dans the section called “Créer une fonction Lambda d'inférence”.

   > **Note**
   >
   > Si vous utilisez l'exemple NVIDIA Jetson, vous devez plutôt utiliser le répertoire `resnet18` dans l'exemple de package. Pour plus d'informations, consultez the section called “Configuration d'un NVIDIA Jetson TX2”.

   ```
   cd path-to-downloaded-sample/dlr-py3-armv7l/models/resnet50
   ```

   Ce répertoire contient des artefacts de modèles précompilés pour un modèle de classification d'images avec Resnet-50.

2. Compressez les fichiers du répertoire `resnet50` dans un fichier nommé `resnet50.zip`.

   ```
   zip -r resnet50.zip .
   ```

3. Sur la page de configuration de groupe de votreAWS IoT Greengrass, choisissez leRessourcesonglet. Accédez à la section Machine Learning, choisissez Ajouter une ressource de Machine Learning. Sur la page Créer une ressource de Machine Learning, pour Nom de la ressource, entrez **`resnet50_model`**.

4. PourSource du modèle, choisissezUtilisez un modèle stocké dans S3, tel qu'un modèle optimisé via Deep Learning Compiler.

5. UNURI S3, choisissezParcourir S3.

> **Note**
>
> Actuellement, optimisé SageMaker les modèles sont stockés automatiquement dans Amazon S3. Vous pouvez trouver votre modèle optimisé dans votre compartiment Amazon S3 à l'aide de cette option. Pour de plus amples informations sur l'optimisation de modèle dans SageMaker, voir lesSageMaker Neo.

6. Choisissez Charger un modèle.
7. Dans l'onglet de la console Amazon S3, chargez le fichier zip dans un compartiment Amazon S3. Pour plus d'informations, consultezComment charger des fichiers ou des dossiers dans un compartiment S3 ?dans leManuel de l'utilisateur Amazon Simple Service.

> **Note**
>
> Le nom de votre compartiment doit contenir la chaîne **`greengrass`**. Choisissez un nom unique (comme **`greengrass-dlr-bucket-`*`user-id`*`-`*`epoch-time`**). N'utilisez pas de point (`.`) dans le nom du compartiment.

8. DansAWS IoT Greengrassonglet de la console, localisez puis choisissez le compartiment Amazon S3. Localisez votre fichier chargé `resnet50.zip`, puis choisissez Sélectionner. Vous devrez peut-être actualiser la page pour mettre à jour la liste des compartiments et des fichiers disponibles.
9. DansChemin de destination, saisissez**`/ml_model`**.

Il s'agit de la destination pour le modèle local dans l'espace de noms d'exécution Lambda. Lorsque vous déployez le groupe, AWS IoT Greengrass permet de récupérer le package de modèle source, puis d'extraire le contenu dans le répertoire spécifié.

> **Note**
>
> Nous vous recommandons vivement d'utiliser le chemin d'accès local exact. L'utilisation d'un autre chemin de destination de modèle local dans cette étape entraîne l'inexactitude

de certaines commandes de dépannage fournies dans ce didacticiel. Si vous utilisez un autre chemin, vous devez configurer une variable d'environnement `MODEL_PATH` qui utilise le chemin d'accès exact que vous fournissez ici. Pour de plus amples informations sur les variables d'environnement, veuillez consulter Variables d'environnement AWS Lambda.

10. En cas d'exécution en mode conteneurisé :

    a. UNPropriétaire du groupe système et autorisations d'accès aux fichiers, choisissezSpécifier le groupe système et les autorisations.

    b. ChoisissezAccès en lecture seule, puis.Ajout d'une ressource.

Étape 6 : Ajouter la ressource d'appareil de votre caméra au groupe Greengrass

Au cours de cette étape, créez une ressource pour le module de caméra et affiliez-la à la fonction Lambda. Cela permet à la fonction Lambda d'accéder à ses ressources sur l'appareil principal.

> **Note**
>
> Si vous exécutez en mode non conteneurisé,AWS IoT Greengrasspeut accéder au GPU et à la caméra de votre appareil sans configurer cette ressource de périphérique.

1. Sur la page de configuration de groupe, choisissez leRessourcesonglet.
2. Sur leDes ressources locales, choisissezAjout d'une ressource locale.
3. Dans la pageAjout d'une ressource locale, utilisez les valeurs suivantes :

    - Sous Resource Name (Nom de la ressource), entrez **`videoCoreSharedMemory`**.
    - Pour Type de ressource, choisissez Appareil.
    - PourChemin de l'appareil local, saisissez**`/dev/vcsm`**.

      Le chemin de l'appareil est le chemin absolu local de la ressource d'appareil. Ce chemin ne peut faire référence qu'à un périphérique de caractères ou un périphérique de stockage en mode bloc sous `/dev`.

- PourPropriétaire du groupe système et autorisations d'accès aux fichiers, choisissezAjouter automatiquement les autorisations de système de fichiers du groupe de fichiers du groupe de système de fichiers du groupe.

  L'option Autorisation d'accès fichier pour le propriétaire du groupe vous permet d'accorder des autorisations supplémentaires d'accès aux fichiers au processus Lambda. Pour plus d'informations, consultez Autorisation d'accès fichier pour le propriétaire du groupe.

4. Au bas de la page, choisissezAjout d'une ressource.
5. À partir de laRessources, créez une autre ressource locale en choisissantAdditionet utilisez les valeurs suivantes :

   - Sous Resource Name (Nom de la ressource), entrez **`videoCoreInterface`**.
   - Pour Type de ressource, choisissez Appareil.
   - PourChemin de l'appareil local, saisissez**`/dev/vchiq`**.
   - PourPropriétaire du groupe système et autorisations d'accès aux fichiers, choisissezAjouter automatiquement les autorisations de système de fichiers du groupe de fichiers du groupe de système de fichiers du groupe.
6. Choisissez Add resource (Ajouter ressource).

Étape 7 : Ajouter des abonnements au groupe Greengrass

Au cours de cette étape, ajoutez des abonnements au groupe. Ces abonnements permettent à la fonction Lambda d'envoyer les résultats des prédictions àAWS IoTen publiant dans une rubrique MQTT.

1. Sur la page de configuration de groupe, choisissez leSubscriptions, puisAjouter un abonnement.
2. Dans la pageCréer un abonnement, configurez la source et la cible comme suit :

   a. DansType de source, choisissezFonction Lambdaet choisissezoptimizedImageClassification.

   b. DansTarget type (Type de cible), choisissezServiceet choisissezCloud IoT.

   c. DansFiltre de rubriques, saisissez**`/resnet-50/predictions`**et choisissezCréer un abonnement.
3. Ajoutez un second abonnement. Cliquez sur l'ongletSubscriptions, choisissezAjouter un abonnementet configurez la source et la cible comme suit :

a. DansType de source, choisissezServiceset choisissezCloud IoT.

b. DansTarget type (Type de cible), choisissezFonction Lambdaet choisissezoptimizedImageClassification.

c. DansFiltre de rubriques, saisissez**`/resnet-50/test`**et choisissezCréer un abonnement.

Étape 8 : Déployer le groupe Greengrass

Au cours de cette étape, déployez la version actuelle de la définition de groupe sur l'appareil Greengrass principal (noyau). La définition contient la fonction Lambda, les ressources et les configurations d'abonnement que vous avez ajoutées.

1. Vérifiez les éléments suivants :AWS IoT Greengrasscore est en cours d'exécution. Dans la fenêtre de terminal de votre Raspberry Pi, exécutez les commandes suivantes, si nécessaire.

   a. Pour vérifier si le démon est en cours d'exécution :

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

   Si la sortie contient une entrée `root` pour `/greengrass/ggc/packages/`*`latest-core-version`*`/bin/daemon`, le démon est en cours d'exécution.

   b. Pour démarrer le démon :

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

2. Sur la page de configuration de groupe, choisissezDéploiement.

3. Dans la pageFonctions Lambdaonglet, sélectionnezDétecteur IPet choisissezModifier.

4. À partir de laModifier les paramètres du détecteur IPboîte de dialogue, sélectionnezDétecter et remplacer automatiquement les points de terminaison des courtiers MQTTet choisissezEnregistrer.

   Les appareils peuvent ainsi acquérir automatiquement des informations de connectivité pour le noyau, telles que l'adresse IP, le DNS et le numéro de port. La détection automatique est recommandée, mais AWS IoT Greengrass prend également en charge les points de terminaison spécifiés manuellement. Vous êtes uniquement invité à indiquer la méthode de découverte lors du déploiement initial du groupe.

> **Note**
>
> Si vous y êtes invité, autorisez la création duRôle de service Greengrasset associez-le à votreCompte AWSdans les actuelsRégion AWS. Ce rôle permetAWS IoT Greengrasspour accéder à vos ressources dansAWSServices .

La page Déploiements indique l'horodatage, l'ID de version et l'état du déploiement. Une fois terminé, le statut affiché pour le déploiement doit êtreTerminé.

Pour de plus amples informations sur les déploiements, veuillez consulterDéploiement de groupes AWS IoT Greengrass. Pour bénéficier d'une aide à la résolution des problèmes, consultez Résolution des problèmes.

Tester l'exemple d'inférence

À présent, vous pouvez vérifier si le déploiement est correctement configuré. Pour effectuer le test, abonnez-vous à la rubrique `/resnet-50/predictions` et publiez n'importe quel message dans la rubrique `/resnet-50/test`. Cela pousse la fonction Lambda à prendre une photo avec votre Raspberry Pi et à effectuer des inférences sur l'image qu'elle capture.

> **Note**
>
> Si vous utilisez l'exemple NVIDIA Jetson, veillez à utiliser plutôt les rubriques `resnet-18/predictions` et `resnet-18/test`.

> **Note**
>
> Si un écran est attaché au Raspberry Pi, le flux de la caméra est diffusé en direct dans une fenêtre d'aperçu.

1. Dans la pageAWS IoTpage d'accueil de la console, sousTest, choisissezClient de test MQTT.
2. PourSubscriptions, choisissezAbonnement à une rubrique. Utilisez les valeurs suivantes. Conservez les valeurs par défaut des autres options.

- Pour Rubrique d'abonnement, entrez **`/resnet-50/predictions`**.
- UNConfiguration supplémentaire, pourAffichage de la charge utile MQTT, choisissezAffichage des charges utiles sous forme de chaînes.

3. Choisissez Subscribe.
4. ChoisissezPublier dans une rubrique, saisissez**`/resnet-50/test`**en tant queNom de la rubrique, et choisissezPublier.
5. Si le test réussit, le message publié provoque la capture d'une image par la caméra Raspberry Pi. Un message provenant de la fonction Lambda s'affiche en bas de la page. Ce message contient le résultat des prédictions de l'image en utilisant le format suivant : nom de classe prédite, probabilité et utilisation de la mémoire.

Configuration d'un processeur Intel Atom

Pour exécuter ce didacticiel sur un appareil Intel Atom, vous devez fournir des images source, configurer la fonction Lambda et ajouter une autre ressource d'appareil local. Pour utiliser le processeur graphique à des fins d'inférence, assurez-vous que les logiciels suivants sont installés sur votre appareil :

- OpenCL version 1.0 ou ultérieure
- Python 3.7 et pip
- NumPy
- OpenCV sur Wheels

1. Téléchargez les images statiques PNG ou JPG pour la fonction Lambda à utiliser pour la classification d'images. L'exemple fonctionne mieux avec des fichiers image de petite taille.

   Enregistrez vos fichiers image dans le répertoire qui contient le fichier `inference.py` (ou dans un sous-répertoire de ce répertoire). Cela fait partie du package de déploiement de fonction Lambda que vous chargez dansthe section called “Créer une fonction Lambda d'inférence”.

   > **Note**
   >
   > Si vous utilisez AWS DeepLens, vous pouvez utiliser la caméra embarquée ou monter votre propre caméra pour effectuer des inférences sur des images capturées plutôt

que sur des images statiques. Cependant, nous vous recommandons vivement de commencer par les images statiques.
Si vous utilisez une caméra, assurez-vous que le package APT `awscam` est installé et à jour. Pour de plus amples informations, veuillez consulterMettre à jour votreAWS DeepLensappareildans leAWS DeepLensManuel du développeur.

2. Modifiez la configuration de la fonction Lambda. Suivez la procédure décrite dans the section called “Ajouter la fonction Lambda au groupe”.

> **Note**
>
> Nous vous recommandons d'exécuter votre fonction Lambda sans conteneurisation, à moins que votre analyse de rentabilisation ne l'exige. Cela permet d'accéder au GPU et à la caméra de votre appareil sans configurer les ressources de l'appareil. Si vous exécutez sans conteneurisation, vous devez également accorder un accès root à votreAWS IoT GreengrassFonctions Lambda.

   a. Pour exécuter sans conteneurisation :

      - PourUtilisateur et groupe du système, choisissez**`Another user ID/group ID`**. PourID utilisateur du système, saisissez**`0`**. PourID de groupe système, saisissez**`0`**.

        Cela permet à votre fonction Lambda de s'exécuter en tant que root. Pour plus d'informations sur l'exécution en tant que root, veuillez consulterthe section called “Définition de l'identité d'accès par défaut pour les fonctions Lambda dans un groupe”.

        > **Tip**
        >
        > Vous devez également mettre à jour votre`config.json`pour accorder à l'accès root à votre fonction Lambda. Pour la procédure, veuillez consulterthe section called “Exécution d'une fonction Lambda en tant que root”.

      - PourConteneurisation de fonction Lambda, choisissezAucun conteneur.

        Pour de plus amples informations sur l'exécution sans conteneurisation, veuillez consulterthe section called “Considérations à prendre en compte lors du choix de la conteneurisation des fonctions Lambda”.

- Augmentez la valeur Expiration en la faisant passer à 2 minutes. Cela permet de s'assurer que la demande n'expire pas trop tôt. L'exécution de l'inférence nécessite quelques minutes après la configuration.
- PourPinned, choisissezVrai.
- UNParamètres supplémentaires, pourAccès en lecture au répertoire /sys, choisissezActivé.

b. Pour exécuter plutôt en mode conteneurisé :

> **Note**
>
> Nous ne recommandons pas une exécution en mode conteneurisé, à moins que votre cas d'utilisation ne l'exige.

- Augmentez la valeur Limite de mémoire en la faisant passer à 3 000 Mo.
- Augmentez la valeur Expiration en la faisant passer à 2 minutes. Cela permet de s'assurer que la demande n'expire pas trop tôt. L'exécution de l'inférence nécessite quelques minutes après la configuration.
- PourPinned, choisissezVrai.
- UNParamètres supplémentaires, pourAccès en lecture au répertoire /sys, choisissezActivé.

3. Ajoutez une ressource de modèle optimisé pour Neo au groupe. Téléchargez les ressources du modèle dans le répertoire `resnet50` de l'exemple de package que vous avez décompressé dans the section called “Créer une fonction Lambda d'inférence”. Ce répertoire contient des artefacts de modèles précompilés pour un modèle de classification d'images avec Resnet-50. Exécutez la procédure décrite dans the section called “Ajouter une ressource de modèle optimisé pour Neo au groupe” en appliquant les mises à jour qui suivent :

   - Compressez les fichiers du répertoire `resnet50` dans un fichier nommé `resnet50.zip`.
   - Sur la page Créer une ressource de Machine Learning, pour Nom de la ressource, entrez **`resnet50_model`**.
   - Chargez le fichier `resnet50.zip`.

4. En cas d'exécution en mode conteneurisé, ajoutez la ressource locale requise pour autoriser l'accès au GPU de votre appareil.

> **Note**
>
> Si vous exécutez en mode non conteneurisé,AWS IoT Greengrasspeut accéder au GPU de votre appareil sans configurer les ressources de l'appareil.

a. Sur la page de configuration de groupe, choisissez leRessourcesonglet.

b. DansDes ressources localessection, choisissezAjout d'une ressource locale.

c. Définissez la ressource :

- Sous Resource Name (Nom de la ressource), entrez **`renderD128`**.
- Pour Type de ressource, choisissez Appareil.
- PourChemin de l'appareil local, saisissez**`/dev/dri/renderD128`**.
- PourPropriétaire du groupe système et autorisations d'accès aux fichiers, choisissezAjouter automatiquement les autorisations de système de fichiers du groupe de fichiers du groupe de système de fichiers du groupe.

Configuration d'un NVIDIA Jetson TX2

Pour exécuter ce didacticiel sur un NVIDIA Jetson TX2, fournissez des images source, configurez la fonction Lambda et ajoutez des ressources d'appareil local supplémentaires.

1. Assurez-vous que votre appareil Jetson est configuré pour que vous puissiez installer le logiciel AWS IoT Greengrass Core et utiliser le processeur graphique pour inférence. Pour de plus amples informations sur la configuration de votre appareil, veuillez consulter the section called “Configuration d'autres appareils”. Pour utiliser le processeur graphique pour inférence sur un NVIDIA Jetson TX2, vous devez installer CUDA 10.0 et cuDNN 7.0 sur votre appareil lorsque vous créez l'image de votre carte avec Jetpack 4.3.
2. Téléchargez les images statiques PNG ou JPG pour la fonction Lambda à utiliser pour la classification d'images. L'exemple fonctionne mieux avec des fichiers image de petite taille.

   Enregistrez vos fichiers image dans le répertoire contenant le fichier `inference.py`. Vous pouvez également les enregistrer dans un sous-répertoire de ce répertoire. Ce répertoire fait partie du package de déploiement de fonction Lambda que vous chargez dansthe section called “Créer une fonction Lambda d'inférence”.

> **Note**
>
> À la place, vous pouvez choisir d'instrumenter une caméra sur la carte Jetson pour capturer les images source. Cependant, nous vous recommandons vivement de commencer par les images statiques.

3. Modifiez la configuration de la fonction Lambda. Suivez la procédure décrite dans the section called “Ajouter la fonction Lambda au groupe”.

> **Note**
>
> Nous vous recommandons d'exécuter votre fonction Lambda sans conteneurisation, à moins que votre analyse de rentabilisation ne l'exige. Cela permet d'accéder au GPU et à la caméra de votre appareil sans configurer les ressources de l'appareil. Si vous exécutez sans conteneurisation, vous devez également accorder un accès root à votreAWS IoT GreengrassFonctions Lambda.

   a. Pour exécuter sans conteneurisation :

      - PourExécuter en tant que, choisissez**`Another user ID/group ID`**. PourUID, saisissez**`0`**. PourGUID, saisissez**`0`**.

        Cela permet à votre fonction Lambda de s'exécuter en tant que root. Pour plus d'informations sur l'exécution en tant que root, veuillez consulterthe section called “Définition de l'identité d'accès par défaut pour les fonctions Lambda dans un groupe”.

        > **Tip**
        >
        > Vous devez également mettre à jour votre`config.json`pour accorder à l'accès root à votre fonction Lambda. Pour la procédure, veuillez consulterthe section called “Exécution d'une fonction Lambda en tant que root”.

      - PourConteneurisation de fonction Lambda, choisissezAucun conteneur.

        Pour de plus amples informations sur l'exécution sans conteneurisation, veuillez consulterthe section called “Considérations à prendre en compte lors du choix de la conteneurisation des fonctions Lambda”.

- Augmentez la valeur Expiration en la faisant passer à 5 minutes. Cela permet de s'assurer que la demande n'expire pas trop tôt. L'exécution de l'inférence nécessite quelques minutes après la configuration.
- PourPinned, choisissezVrai.
- UNParamètres supplémentaires, pourAccès en lecture au répertoire /sys, choisissezActivé.

b. Pour exécuter plutôt en mode conteneurisé :

> **Note**
>
> Nous ne recommandons pas une exécution en mode conteneurisé, à moins que votre cas d'utilisation ne l'exige.

- Augmentez la valeur Limite de mémoire. Pour utiliser le modèle fourni en mode GPU (processeur graphique), utilisez au moins 2000 Mo.
- Augmentez la valeur Expiration en la faisant passer à 5 minutes. Cela permet de s'assurer que la demande n'expire pas trop tôt. L'exécution de l'inférence nécessite quelques minutes après la configuration.
- PourPinned, choisissezVrai.
- UNParamètres supplémentaires, pourAccès en lecture au répertoire /sys, choisissezActivé.

4. Ajoutez une ressource de modèle optimisé pour Neo au groupe. Téléchargez les ressources du modèle dans le répertoire `resnet18` de l'exemple de package que vous avez décompressé dans the section called “Créer une fonction Lambda d'inférence”. Ce répertoire contient des artefacts de modèles précompilés pour un modèle de classification d'images entraîné avec Resnet-18. Exécutez la procédure décrite dans the section called “Ajouter une ressource de modèle optimisé pour Neo au groupe” en appliquant les mises à jour qui suivent :

   - Compressez les fichiers du répertoire `resnet18` dans un fichier nommé `resnet18.zip`.
   - Sur la page Créer une ressource de Machine Learning, pour Nom de la ressource, entrez **`resnet18_model`**.
   - Chargez le fichier `resnet18.zip`.

5. En cas d'exécution en mode conteneurisé, ajoutez les ressources locales requises pour autoriser l'accès au GPU de votre appareil.

> **Note**
>
> Si vous exécutez en mode non conteneurisé,AWS IoT Greengrasspeut accéder au GPU de votre appareil sans configurer les ressources de l'appareil.

a. Sur la page de configuration de groupe, choisissez leRessourcesonglet.

b. DansDes ressources localessection, choisissezAjout d'une ressource locale.

c. Définissez chaque ressource :

- Pour Resource name (Nom de la ressource) et Device path, (Chemin de l'appareil), utilisez les valeurs du tableau suivant. Créez une ressource de périphérique pour chaque ligne de la table.
- Pour Type de ressource, choisissez Appareil.
- PourPropriétaire du groupe système et autorisations d'accès aux fichiers, choisissezAjouter automatiquement les autorisations de système de fichiers du groupe de fichiers du groupe de système de fichiers du groupe.

Nom	Chemin de l'appareil
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

6. En cas d'exécution en mode conteneurisé, ajoutez la ressource de volume local suivante pour autoriser l'accès à l'appareil photo de votre appareil. Suivez la procédure décrite dans the section called “Ajouter une ressource de modèle optimisé pour Neo au groupe”.

   > **Note**
   >
   > Si vous exécutez en mode non conteneurisé,AWS IoT Greengrasspeut accéder à la caméra de votre appareil sans configurer les ressources de l'appareil.

   - Pour Type de ressource, choisissez Volume.
   - PourPropriétaire du groupe système et autorisations d'accès aux fichiers, choisissezAjouter automatiquement les autorisations de système de fichiers du groupe de fichiers du groupe de système de fichiers du groupe.

Nom	Chemin source	Chemin de destination
shm	/dev/shm	/dev/shm
tmp	/tmp	/tmp

7. Mettez à jour vos abonnements de groupe pour utiliser le répertoire correct. Exécutez la procédure décrite dans the section called “Ajouter des abonnements au groupe” en appliquant les mises à jour qui suivent :

   - Pour votre premier filtre de rubrique, entrez **`/resnet-18/predictions`**.
   - Pour votre deuxième filtre de rubrique, entrez **`/resnet-18/test`**.

8. Mettez à jour vos abonnements de test pour utiliser le répertoire correct. Exécutez la procédure décrite dans the section called “Tester l'exemple” en appliquant les mises à jour qui suivent :

   - PourSubscriptions, choisissezS'abonner à une rubrique. Pour Rubrique d'abonnement, entrez **`/resnet-18/predictions`**.
   - Dans la page `/resnet-18/predictions`, spécifiez la rubrique `/resnet-18/test` sur laquelle publier.

Dépannage de l'inférence ML AWS IoT Greengrass

Si le test n'est pas réussi, vous pouvez essayer les étapes de dépannage suivantes. Exécutez les commandes dans la fenêtre de terminal de votre Raspberry Pi.

Consultez les journaux des erreurs

1. Connectez-vous en tant qu'utilisateur racine et accédez au répertoire `log`. L'accès aux journaux AWS IoT Greengrass nécessite les autorisations racines.

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

2. Check`runtime.log`pour toutes erreurs.

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

   Vous pouvez également rechercher les éventuelles erreurs dans le journal des fonctions Lambda définies par l'utilisateur :

   ```
   cat user/your-region/your-account-id/lambda-function-name.log | grep 'ERROR'
   ```

   Pour plus d'informations, consultez the section called “Résolution des problèmes liés aux journaux”.

Vérifiez que la fonction Lambda a été déployée avec succès

1. Affichez le contenu de Lambda déployé dans le`/lambda`annuaire. Remplacez les valeurs d'espace réservé avant d'exécuter la commande.

   ```
   cd /greengrass/ggc/deployment/lambda/
   arn:aws:lambda:region:account:function:function-name:function-version
   ls -la
   ```

2. Vérifiez que le répertoire contient le même contenu que le package de déploiement `optimizedImageClassification.zip` que vous avez chargé dans Étape 3 : Créer une fonction Lambda d'inférence.

Assurez-vous que les fichiers `.py` et les dépendances se trouvent dans la racine du répertoire.

Vérifiez que le modèle d'inférence a été déployé avec succès

1. Trouvez le numéro d'identification de processus (PID) du processus d'exécution Lambda :

   ```
   ps aux | grep lambda-function-name
   ```

   Dans la sortie, le PID s'affiche dans la deuxième colonne de la ligne pour le processus d'exécution Lambda.

2. Saisissez l'espace de noms d'exécution Lambda. Veillez à remplacer la valeur d'espace réservé *`pid`* avant d'exécuter la commande.

   > **Note**
   >
   > Ce répertoire et son contenu se trouvent dans l'espace de noms d'exécution Lambda ; par conséquent, ils ne sont pas visibles dans un espace de noms Linux classique.

   ```
   sudo nsenter -t pid -m /bin/bash
   ```

3. Affichez le contenu du répertoire local que vous avez spécifié comme ressource de ML.

   > **Note**
   >
   > Si votre chemin de ressource ML est différent de `ml_model`, vous devez le remplacer ici.

   ```
   cd /ml_model
   ls -ls
   ```

   Vous devriez voir les fichiers suivants :

   ```
       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 fonction Lambda ne parvient pas à trouver **/dev/dri/renderD128**

Cela peut se produire si OpenCL ne parvient pas à se connecter aux appareils GPU dont il a besoin. Vous devez créer des ressources pour les appareils nécessaires pour votre fonction Lambda.

Étapes suivantes

Ensuite, explorez d'autres modèles optimisés. Pour plus d'informations, consultez la documentationSageMaker Neo.

Gestion des flux de données sur AWS IoT Greengrass Core

AWS IoT Greengrassle gestionnaire de flux facilite et fiabilise le transfert de données IoT à volume élevé vers leAWS Cloud. Le gestionnaire de flux traite les flux de données localement et les exporte vers leAWS Cloudautomatiquement. Cette fonctionnalité s'intègre à des scénarios courants, tels que l'inférence de Machine Learning (ML), dans lequel les données sont traitées et analysées localement avant d'être exportées vers leAWS Cloudou des destinations de stockage locales.

Le gestionnaire de flux simplifie le développement d'applications. Vos applications IoT peuvent utiliser un mécanisme standardisé pour traiter les flux à volume élevé et gérer les stratégies de conservation des données locales au lieu de créer des fonctionnalités de gestion des flux personnalisées. Les applications IoT peuvent lire et écrire dans des flux. Elles peuvent définir des stratégies pour le type et la taille du stockage, ainsi que pour la conservation des données par flux afin de contrôler la façon dont le gestionnaire de flux traite et exporte les flux.

Le gestionnaire de flux est conçu pour fonctionner dans des environnements à connectivité intermittente ou limitée. Vous pouvez définir l'utilisation de la bande passante, le comportement du délai d'expiration et la façon dont les données du flux sont traitées lorsque le noyau est connecté ou déconnecté. Pour les données critiques, vous pouvez définir des priorités afin de contrôler l'ordre d'exportation des flux vers leAWS Cloud.

Vous pouvez configurer les exportations automatiques vers leAWS Cloudpour le stockage ou le traitement et l'analyse ultérieurs. Stream Manager prend en charge l'exportation vers les éléments suivantsAWS CloudDestinations.

- Canaux dansAWS IoT Analytics.AWS IoT Analyticsvous permet d'effectuer des analyses avancées de vos données pour aider à prendre des décisions commerciales et à améliorer les modèles d'apprentissage automatique. Pour de plus amples informations, veuillez consulterPrésentation d'AWS IoT Analytics?dans leAWS IoT AnalyticsGuide de l'utilisateur.
- Flux dans Kinesis Data Streams. Kinesis Data Streams est couramment utilisé pour agréger des données à volume élevé et les charger dans un entrepôt de données ou un cluster de réduction de mappe. Pour de plus amples informations, veuillez consulterPrésentation de Amazon Kinesis Data Streamsdans leManuel du développeur Amazon Kinesis.
- Propriétés des ressources dansAWS IoT SiteWise.AWS IoT SiteWisevous permet de collecter, d'organiser et d'analyser à grande échelle les données provenant d'équipements industriels. Pour de plus amples informations, veuillez consulterPrésentation d'AWS IoT SiteWise?dans leAWS IoT SiteWiseGuide de l'utilisateur.

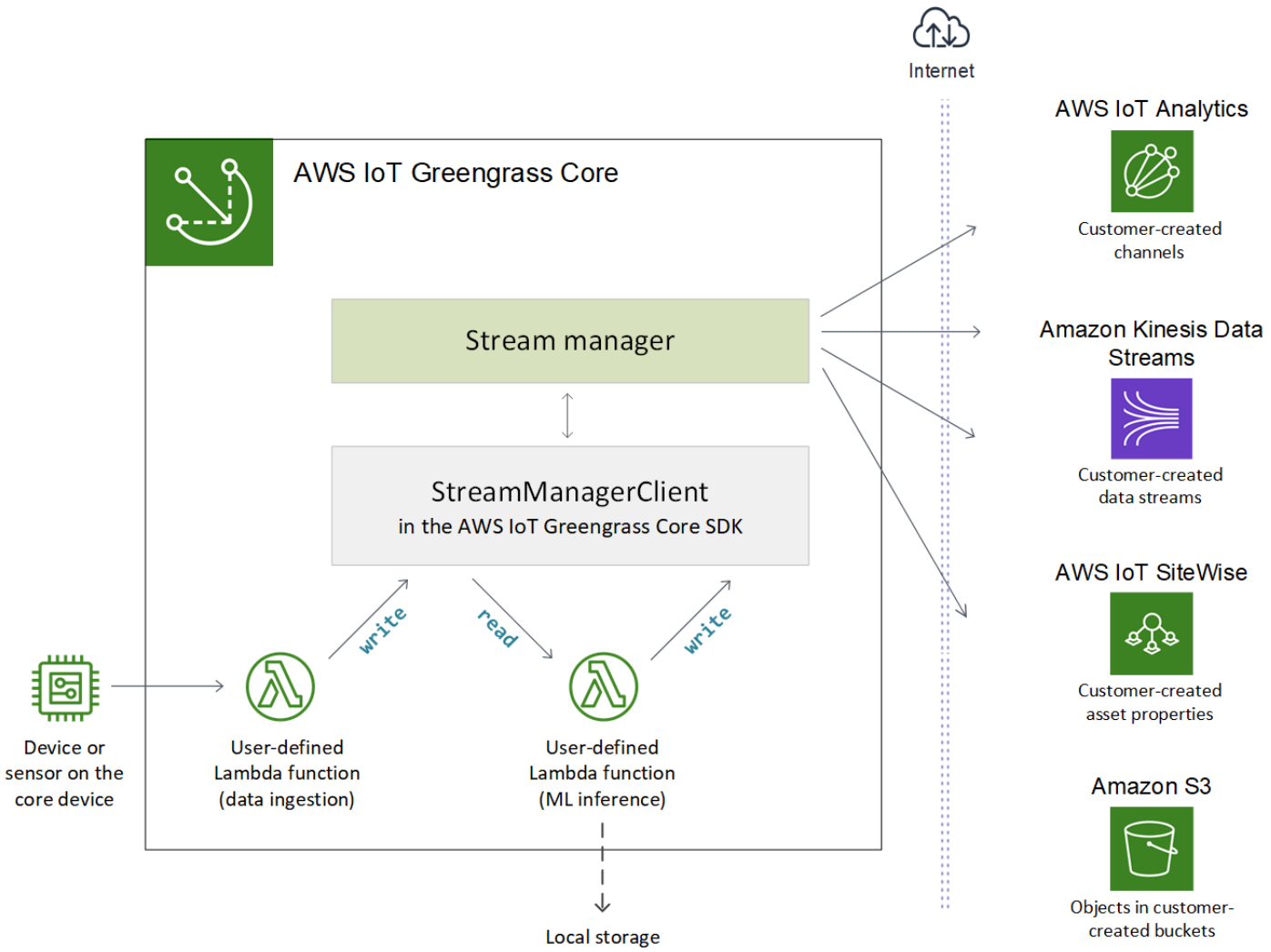
- Objets dans Amazon S3. Vous pouvez utiliser Amazon S3 pour stocker et récupérer de grandes quantités de données. Pour de plus amples informations, veuillez consulterQu'est-ce qu'Amazon S3 ?dans leManuel du développeur Amazon Simple Storage Service.

Flux de travail de gestion des flux

Vos applications IoT interagissent avec le gestionnaire de flux via leAWS IoT GreengrassKit SDK Core. Dans un flux de travail simple, une fonction Lambda définie par l'utilisateur et s'exécutant sur le noyau de Greengrass consomme des données IoT, telles que les mesures de température et de pression sous forme de séries chronologiques. La fonction Lambda peut filtrer ou compresser les données, puis appeler leAWS IoT GreengrassKit SDK Core pour écrire les données dans un flux dans le gestionnaire de flux. Le gestionnaire de flux peut exporter le flux vers leAWS Cloudautomatiquement, en fonction des stratégies définies pour le flux. Les fonctions Lambda définies par l'utilisateur peuvent également envoyer des données directement aux bases de données locales ou aux référentiels de stockage locaux.

Vos applications IoT peuvent inclure plusieurs fonctions Lambda définies par l'utilisateur qui lisent ou écrivent dans des flux. Ces fonctions Lambda locales peuvent lire et écrire dans des flux afin de filtrer, d'agréger et d'analyser les données localement. Cela permet de répondre rapidement aux événements locaux et d'extraire des informations précieuses avant que les données ne soient transférées du noyau vers le cloud ou vers des destinations locales.

Le diagramme suivant présente un exemple de flux de travail.



Pour utiliser le gestionnaire de flux, commencez par configurer les paramètres du gestionnaire de flux afin de définir les paramètres d'exécution au niveau du groupe qui s'appliquent à tous les flux du cœur de Greengrass. Ces paramètres personnalisables vous permettent de contrôler la manière dont le gestionnaire de flux stocke, traite et exporte les flux en fonction des besoins de votre entreprise et des contraintes de votre environnement. Pour plus d'informations, consultez the section called “Configuration du gestionnaire de flux”.

Une fois que vous avez configuré Stream Manager, vous pouvez créer et déployer vos applications IoT. Il s'agit généralement de fonctions Lambda définies par l'utilisateur qui utilisent`StreamManagerClient`dans leAWS IoT GreengrassKit SDK Core pour créer des flux et interagir avec ces derniers. Lors de la création de flux, la fonction Lambda définit des stratégies par flux, telles que les destinations d'exportation, la priorité et la persistance. Pour en savoir plus, notamment sur les extraits de code pour`StreamManagerClient`, consultezthe section called “Utiliser StreamManagerClient pour travailler avec des flux”.

Pour obtenir des didacticiels qui configurent un flux de travail simple, consultezthe section called “Exportation de flux de données (console)”outhe section called “Exportation de flux de données (interface de ligne de commande)”.

Prérequis

Les conditions requises suivantes s'appliquent pour utiliser le gestionnaire de flux :

- Vous devez utiliserAWS IoT GreengrassLogiciel de base v1.10 ou version ultérieure, avec le gestionnaire de flux activé. Pour plus d'informations, consultez the section called “Configuration du gestionnaire de flux”.

  Le gestionnaire de flux n'est pas pris en charge sur OpenWrt distributions.
- L'environnement d'exécution Java 8 (JDK 8) doit être installé sur le noyau.
  - Pour les distributions basées sur Debian (y compris Raspbian) ou Ubuntu, exécutez la commande suivante :

    ```
    sudo apt install openjdk-8-jdk
    ```
  - Pour les distributions basées sur Red Hat (y compris Amazon Linux), exécutez la commande suivante :

    ```
    sudo yum install java-1.8.0-openjdk
    ```

    Pour de plus amples informations, veuillez consulter How to download and install prebuilt OpenJDK packages sur le site web OpenJDK.

- Le gestionnaire de flux nécessite un minimum de 70 Mo de RAM en plus de votre logiciel AWS IoT Greengrass Core de base. Vos besoins totaux en mémoire dépendent de votre charge de travail.

- Les fonctions Lambda définies par l'utilisateur doivent utiliser leAWS IoT GreengrassKit SDK Corepour interagir avec le gestionnaire de flux. LeAWS IoT GreengrassLe kit SDK Core est disponible en plusieurs langues, mais seules les versions suivantes prennent en charge les opérations du gestionnaire de flux :
  - Kit SDK Java (v1.4.0 ou version ultérieure)

- Kit SDK Python (v1.5.0 ou version ultérieure)
- Kit SDK Node.js (v1.6.0 ou version ultérieure)

Téléchargez la version du kit SDK qui correspond à l'environnement d'exécution de votre fonction Lambda et incluez-la dans votre package de déploiement de votre fonction Lambda.

> **Note**
>
> LeAWS IoT GreengrassLe kit SDK Core pour Python nécessite Python 3.7 ou version ultérieure et a d'autres dépendances en matière de package. Pour de plus amples informations, veuillez consulterCréer un package de déploiement de votre fonction Lambda (console)ouCréation d'un package de déploiement de fonction Lambda (CLI).

- Si vous définissezAWS CloudDestinations d'exportation pour un flux, vous devez créer vos cibles d'exportation et accorder des autorisations d'accès dans le rôle de groupe Greengrass. Selon la destination, d'autres exigences peuvent également s'appliquer. Pour plus d'informations, consultez :
  - the section called “Canaux AWS IoT Analytics”
  - the section called “Amazon Kinesis Data Streams”
  - the section called “AWS IoT SiteWisepropriétés des ressources”
  - the section called “Objets Amazon S3”

  Vous êtes responsable de la maintenance de cesAWS CloudAWS.

Sécurité des données

Lorsque vous utilisez le gestionnaire de flux, tenez compte des considérations de sécurité suivantes.

Sécurité des données locales

AWS IoT Greengrass ne chiffre pas les données de flux au repos ou en transit localement entre les composants de l'appareil principal.

- Données au repos. Les données de flux sont stockées localement dans un répertoire de stockage sur le noyau de Greengrass. Pour la sécurité des données, AWS IoT Greengrass s'appuie sur les autorisations de fichier Unix et sur le chiffrement intégral du disque, si cette option est activée. Vous pouvez utiliser le paramètre facultatif STREAM_MANAGER_STORE_ROOT_DIR pour

spécifier le répertoire de stockage. Si vous modifiez ce paramètre ultérieurement pour utiliser un répertoire de stockage différent, AWS IoT Greengrass ne supprime pas le répertoire de stockage précédent ni son contenu.

- Données en transit local.AWS IoT Greengrassne chiffre pas les données de flux en transit local sur le noyau entre les sources de données, les fonctions Lambda, leAWS IoT GreengrassSDK principal et gestionnaire de flux.

- Données en transit vers leAWS Cloud. Flux de données exportés par le gestionnaire de flux vers leAWS CloudUtilisation standardAWSle chiffrement du client de service avec Transport Layer Security (TLS).

Pour plus d'informations, consultez the section called “Chiffrement des données”.

Authentification client

Les clients du gestionnaire de flux utilisent leAWS IoT GreengrassKit SDK Core pour communiquer avec le gestionnaire de flux. Lorsque l'authentification client est activée, seules les fonctions Lambda du groupe Greengrass peuvent interagir avec les flux du gestionnaire de flux. Lorsque l'authentification client est désactivée, tout processus s'exécutant sur le noyau Greengrass (tels que les conteneurs Docker) peut interagir avec les flux du gestionnaire de flux. Vous ne devez désactiver l'authentification que si votre analyse de rentabilisation l'exige.

Vous utilisez le paramètre STREAM_MANAGER_AUTHENTICATE_CLIENT pour définir le mode d'authentification du client. Vous pouvez configurer ce paramètre à partir de la console ou de l'API AWS IoT Greengrass. Les modifications prennent effet après le déploiement du groupe.

	Enabled	Désactivé
Valeur de paramètre	`true` (valeur par défaut et recommandée)	`false`
Clients autorisés	Fonctions Lambda définies par l'utilisateur dans le groupe Greengrass	Fonctions Lambda définies par l'utilisateur dans le groupe Greengrass

	Enabled	Désactivé
		Autres processus en cours d'exécution sur l'appareil principal Greengrass

Consulter aussi

- the section called “Configuration du gestionnaire de flux”
- the section called “Utiliser StreamManagerClient pour travailler avec des flux”
- the section called “Configurations d'exportation pour prises en chargeAWS Clouddestinations”
- the section called “Exportation de flux de données (console)”
- the section called “Exportation de flux de données (interface de ligne de commande)”

Configuration du gestionnaire de flux AWS IoT Greengrass

Dans la pageAWS IoT GreengrassLe gestionnaire de flux peut stocker, traiter et exporter les données des appareils IoT sur les appareils. Le gestionnaire de flux fournit les paramètres que vous utilisez pour configurer les paramètres d'exécution au niveau du groupe. Ces paramètres s'appliquent à tous les flux sur le noyau Greengrass. Vous pouvez utiliser le pluginAWS IoTConsole ouAWS IoT GreengrassAPI pour configurer les paramètres du gestionnaire de flux. Les modifications prennent effet après le déploiement du groupe.

> **Note**
>
> Après avoir configuré Stream Manager, vous pouvez créer et déployer des applications IoT qui s'exécutent sur le cœur de Greengrass et interagissent avec Stream Manager. Ces applications IoT sont généralement des fonctions Lambda définies par l'utilisateur. Pour plus d'informations, consultez the section called “Utiliser StreamManagerClient pour travailler avec des flux”.

Paramètres du gestionnaire de flux

Le gestionnaire de flux fournit les paramètres suivants qui vous permettent de définir des paramètres au niveau du groupe. Tous les paramètres sont facultatifs.

Répertoire de stockage

Nom du paramètre: `STREAM_MANAGER_STORE_ROOT_DIR`

Chemin absolu du répertoire local utilisé pour stocker les flux. Cette valeur doit commencer par une barre oblique (par exemple, `/data`).

Pour de plus amples informations sur la sécurité des données de flux, veuillez consulter the section called “Sécurité des données locales”.

MinimumAWS IoT GreengrassVersion Core : 1.10.0

Server port

Nom du paramètre: `STREAM_MANAGER_SERVER_PORT`

Numéro de port local utilisé pour communiquer avec le gestionnaire de flux. La valeur par défaut est `8088`.

MinimumAWS IoT GreengrassVersion Core : 1.10.0

Authentification du client

Nom du paramètre: `STREAM_MANAGER_AUTHENTICATE_CLIENT`

Indique si les clients doivent être authentifiés pour interagir avec le gestionnaire de flux. Toute interaction entre les clients et le gestionnaire de flux est contrôlée par leAWS IoT GreengrassKit SDK Core. Ce paramètre détermine quels sont les clients qui peuvent appeler leAWS IoT GreengrassKit de développement logiciel de base pour utiliser des flux. Pour plus d'informations, consultez the section called “Authentification client”.

Les valeurs valides sont `true` ou `false`. La valeur par défaut est `true` (recommandé).

- `true`. Autorise uniquement les fonctions Lambda de Greengrass en tant que clients. Les clients de fonction Lambda utilisentAWS IoT Greengrassprotocoles de base pour s'authentifier auprès deAWS IoT GreengrassKit SDK Core.
- `false`. Permet à tout processus s'exécutant surAWS IoT Greengrassessentiel pour être un client. Ne définissez pas ce paramètre sur `false`, à moins que votre analyse de rentabilisation

ne l'exige. Par exemple, définissez cette valeur sur`false`uniquement si les processus autres que sur l'appareil principal doivent communiquer directement avec le gestionnaire de flux, comme c'est la cas pour les cas pour lesConteneur Dockersur le noyau.

MinimumAWS IoT GreengrassVersion Core : 1.10.0

Bande passante maximum

Nom du paramètre: `STREAM_MANAGER_EXPORTER_MAX_BANDWIDTH`

Bande passante maximale moyenne (en kilobits par seconde) pouvant être utilisée pour exporter des données. La valeur par défaut permet une utilisation illimitée de la bande passante disponible.

MinimumAWS IoT GreengrassVersion Core : 1.10.0

Taille du groupe de threads

Nom du paramètre: `STREAM_MANAGER_EXPORTER_THREAD_POOL_SIZE`

Nombre maximal de threads actifs pouvant être utilisés pour exporter des données. La valeur par défaut est `5`.

La taille optimale dépend de votre matériel, du volume de flux et du nombre planifié de flux d'exportation. Si votre vitesse d'exportation est faible, vous pouvez ajuster ce paramètre afin de trouver la taille optimale en fonction de votre matériel et de votre analyse de rentabilisation. Le processeur et la mémoire de votre appareil principal sont des facteurs limitatifs. Pour commencer, vous pouvez essayer de définir cette valeur par le nombre de cœurs de processeur sur l'appareil.

Veillez à ne pas définir une taille supérieure à ce que votre matériel peut prendre en charge. Chaque flux consomme des ressources matérielles. Vous devez donc essayer de limiter le nombre de flux d'exportation sur les appareils soumis à des contraintes.

MinimumAWS IoT GreengrassVersion Core : 1.10.0

Arguments JVM

Nom du paramètre: `JVM_ARGS`

Arguments JVM (machine virtuelle Java) personnalisés à transmettre au gestionnaire de flux au démarrage. Les arguments doivent être séparés par des espaces.

Utilisez ce paramètre uniquement lorsque vous devez remplacer les paramètres par défaut utilisés par la JVM. Par exemple, il peut s'avérer nécessaire d'augmenter la taille de pile par défaut si vous prévoyez d'exporter un grand nombre de flux.

MinimumAWS IoT GreengrassVersion Core : 1.10.0

Répertoires de fichiers en lecture seule

Nom du paramètre: `STREAM_MANAGER_READ_ONLY_DIRS`

Liste séparée par des virgules de chemins absolus vers les répertoires autres que sur le système de fichiers racine, dans laquelle stocker les fichiers d'entrée. Stream Manager lit et télécharge les fichiers sur Amazon S3 et monte les répertoires en lecture seule. Pour plus d'informations sur l'exportation vers Amazon S3, consultezthe section called “Objets Amazon S3”.

Utilisez ce paramètre uniquement si les conditions suivantes sont vraies :

- Le répertoire du fichier d'entrée d'un flux qui exporte vers Amazon S3 se trouve dans l'un des emplacements suivants :
  - Une partition autre que le système de fichiers racine.
  - UNDER`/tmp`sur le système de fichiers racine.
- Leconteneurisation par défautdu groupe GreengrassConteneur Greengrass.

Exemple de valeur : `/mnt/directory-1,/mnt/directory-2,/tmp`

MinimumAWS IoT GreengrassVersion Core : 1.11.0

Taille minimale pour le chargement partitionné

Nom du paramètre:
`STREAM_MANAGER_EXPORTER_S3_DESTINATION_MULTIPART_UPLOAD_MIN_PART_SIZE_BYTES`

Taille minimale (en octets) d'une partie d'un chargement partitionné vers Amazon S3. Le gestionnaire de flux utilise ce paramètre et la taille du fichier d'entrée pour déterminer comment regrouper les données dans une requête PUT partitionnée. La valeur par défaut et minimale est`5242880`octets (5 Mo).

> **Note**
>
> Le gestionnaire de flux utilise le`sizeThresholdForMultipartUploadBytes`pour déterminer s'il faut exporter vers Amazon S3 en tant que chargement partitionné ou en plusieurs parties. Les fonctions Lambda définies par l'utilisateur définissent ce seuil lorsqu'elles créent un flux qui exporte vers Amazon S3. Le seuil par défaut est de 5 Mo.

MinimumAWS IoT GreengrassVersion Core : 1.11.0

Configurer les paramètres du gestionnaire de flux (console)

Vous pouvez utiliser le pluginAWS IoTpour les tâches de gestion suivantes :

- Vérifier si le gestionnaire de flux est activé
- Activer ou désactiver le gestionnaire de flux lors de la création de groupe
- Activer ou désactiver le gestionnaire de flux pour un groupe existant
- Modifier les paramètres du gestionnaire de flux

Les modifications prennent effet après le déploiement du groupe Greengrass. Pour consulter un didacticiel qui explique comment déployer un groupe Greengrass contenant une fonction Lambda qui interagit avec le gestionnaire de flux, voirthe section called “Exportation de flux de données (console)”.

> **Note**
>
> Lorsque vous utilisez la console pour activer le gestionnaire de flux et déployer le groupe, la taille de la mémoire du gestionnaire de flux est définie sur 4 194 304 Ko (4 Go) par défaut. Nous vous recommandons de définir la taille de la mémoire sur au moins 128 000 Ko.

Pour vérifier si le gestionnaire de flux est activé (console)

1. DansAWS IoTVolet de navigation de la console sousGérer, DéveloppezAppareils Greengrass,,,,,,.Groupes (V1).
2. Choisissez le groupe cible.
3. Cliquez sur l'ongletTabulation de fonctions Lambda.
4. UNDERFonctions Lambda, SelectGestionnaire de flux, et choisissezModifier.
5. Vérifiez l'état activé ou désactivé. Tous les paramètres personnalisés du gestionnaire de flux qui sont configurés sont également affichés.

Pour activer ou désactiver le gestionnaire de flux lors de la création de groupe (console)

1. DansAWS IoTVolet de navigation de la console sousGérer, DéveloppezAppareils Greengrass,,,,,,.Groupes (V1).
2. Choisissez Create Group. Votre choix sur la page suivante détermine la façon dont vous configurez le gestionnaire de flux pour le groupe.
3. Continuez dans leNommez votre groupeet choisissez unNoyau Greengrass.
4. Choisissez Créer un groupe.
5. Sur la page de configuration de groupe, choisissez l'Fonctions Lambdaonglet, sélectionnezGestionnaire de flux, et choisissezModifier.

   - Pour activer le gestionnaire de flux avec les paramètres par défaut, choisissezActiver avec les paramètres par défaut.

   - Pour activer le gestionnaire de flux avec des paramètres personnalisés, choisissez Customize settings (Personnalisation des paramètres).

     1. Dans la pageConfiguration du gestionnaire de flux, choisissezActiver avec des paramètres personnalisés.
     2. Sous Custom settings (Paramètres personnalisés), entrez des valeurs pour les paramètres du gestionnaire de flux. Pour plus d'informations, consultez the section called “Paramètres du gestionnaire de flux”. Laissez les champs vides afin que AWS IoT Greengrass utilise ses valeurs par défaut.

   - Pour désactiver le gestionnaire de flux, choisissezDésactiver.

     1. Sur la page Configure stream manager (Configurer le gestionnaire de flux), choisissez Disable (Désactiver).

6. Choisissez Save (Enregistrer).
7. Continuez à parcourir les pages restantes pour créer votre groupe.
8. Dans la pageAppareils de client, téléchargez vos ressources de sécurité, consultez les informations, puis choisissezTerminer.

> **Note**
>
> Une fois le gestionnaire de flux activé, vous devez installer l'environnement d'exécution Java 8 sur l'appareil principal (noyau) avant de déployer votre groupe.

Pour activer ou désactiver le gestionnaire de flux pour un groupe existant (console)

1. DansAWS IoTVolet de navigation de la console sousGérer, DéveloppezAppareils Greengrass,,,,,,.Groupes (V1).
2. Choisissez le groupe cible.
3. Cliquez sur l'ongletTabulation de fonctions Lambda.
4. UNDERFonctions Lambda, SelectGestionnaire de flux, et choisissezModifier.
5. Vérifiez l'état activé ou désactivé. Tous les paramètres personnalisés du gestionnaire de flux qui sont configurés sont également affichés.

Pour modifier les paramètres du gestionnaire de flux (console)

1. DansAWS IoTVolet de navigation de la console sousGérer, DéveloppezAppareils Greengrass,,,,,,.Groupes (V1).
2. Choisissez le groupe cible.
3. Cliquez sur l'ongletTabulation de fonctions Lambda.
4. UNDERFonctions Lambda, SelectGestionnaire de flux, et choisissezModifier.
5. Vérifiez l'état activé ou désactivé. Tous les paramètres personnalisés du gestionnaire de flux qui sont configurés sont également affichés.
6. Choisissez Save (Enregistrer).

Configurer les paramètres du gestionnaire de flux (interface de ligne de commande)

DansAWS CLI, utilisez le système`GGStreamManager`Fonction Lambda pour configurer le gestionnaire de flux. Les Lambda sont des composants deAWS IoT GreengrassLogiciel Core. Pour le gestionnaire de flux et certaines autres fonctions Lambda, vous pouvez configurer la fonctionnalité Greengrass en gérant les`Function`et`FunctionDefinitionVersion`objets du groupe Greengrass. Pour plus d'informations, consultez the section called “Présentation du modèle d'objet de groupe”.

Vous pouvez utiliser l'API pour les tâches de gestion suivantes. Les exemples de cette section vous montrent comment utiliser laAWS CLI, mais vous pouvez également appeler leAWS IoT GreengrassAPI directement ou utilisez unAWSKIT SDK.

- Vérifier si le gestionnaire de flux est activé
- Activer, désactiver ou configurer le gestionnaire de flux

Les modifications prennent effet après le déploiement du groupe. Pour accéder à un didacticiel qui explique comment déployer un groupe Greengrass avec une fonction Lambda qui interagit avec le gestionnaire de flux, voirthe section called “Exportation de flux de données (interface de ligne de commande)”.

> **Tip**
>
> Pour savoir si le gestionnaire de flux est activé et s'exécute sur votre appareil principal, vous pouvez exécuter la commande suivante dans un terminal sur l'appareil.
>
> ```
> ps aux | grep -i 'streammanager'
> ```

Pour vérifier si le gestionnaire de flux est activé (interface de ligne de commande)

Le gestionnaire de flux est activé si la version de la définition de fonction déployée inclut le système.`GGStreamManager`Une fonction Lambda. Pour vérifier, procédez comme suit :

1. Obtenez les ID du groupe et de la version de groupe Greengrass cible. Cette procédure suppose qu'il s'agit du dernier groupe et de la dernière version du groupe. La requête suivante renvoie le groupe créé le plus récemment.

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

   Vous pouvez également procéder à une interrogation par nom. Les noms de groupe ne devant pas nécessairement être uniques, plusieurs groupes peuvent être renvoyés.

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

   > **Note**
   >
   > Vous trouverez également ces valeurs dansAWS IoTconsole L'ID du groupe s'affiche sur la page Paramètres du groupe. Les ID de version de groupe sont affichés sur leDéploiementsTabulation.

2. Copiez les valeurs `LatestVersion` et `Id` du groupe cible dans la sortie.
3. Obtenez la version de groupe la plus récente.

   - Remplacez *`group-id`* par la propriété `Id` que vous avez copiée.
   - Remplacez*`latest-group-version-id`*avec le`LatestVersion`que vous avez copié.

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

4. Depuis `FunctionDefinitionVersionArn` dans la sortie, copiez l'ID de la définition de fonction et l'ID de version de la définition de fonction :

   - L'ID de la définition de fonction est le GUID qui suit la`functions`dans l'ARN (Amazon Resource Name).
   - L'ID de version de la définition de fonction est le GUID qui suit le segment `versions` dans l'ARN.

```
arn:aws:greengrass:us-west-2:123456789012:/greengrass/definition/
functions/function-definition-id/versions/function-definition-version-id
```

5. Récupérez la version de la définition de fonction

   - Remplacez*function-definition-id*avec l'ID de la définition de fonction
   - Remplacez*function-definition-version-id*avec l'ID de version de la définition de fonction

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

Si le tableau `functions` dans la sortie inclut la fonction `GGStreamManager`, alors le gestionnaire de flux est activé. Toutes les variables d'environnement définies pour la fonction représentent des paramètres personnalisés pour le gestionnaire de flux.

Pour activer, désactiver ou configurer le gestionnaire de flux (interface de ligne de commande)

DansAWS CLI, utilisez le système`GGStreamManager`Fonction Lambda pour configurer le gestionnaire de flux. Les modifications prennent effet après le déploiement du groupe.

- Pour activer le gestionnaire de flux, incluez `GGStreamManager` dans le tableau `functions` de votre version de définition de fonction. Pour configurer des paramètres personnalisés, définissez des variables d'environnement pour les paramètres du gestionnaire de flux correspondants.
- Pour désactiver le gestionnaire de flux, supprimez `GGStreamManager` du tableau `functions` de votre version de définition de fonction.

Gestionnaire de flux avec les paramètres par défaut

L'exemple de configuration suivant active le gestionnaire de flux avec les paramètres par défaut. Il définit l'ID de la fonction arbitraire sur `streamManager`.

```
{
```

```
    "FunctionArn": "arn:aws:lambda:::function:GGStreamManager:1",
    "FunctionConfiguration": {
        "MemorySize": 4194304,
        "Pinned": true,
        "Timeout": 3
    },
    "Id": "streamManager"
}
```

> **Note**
>
> Pour`FunctionConfiguration`, vous pouvez connaître les éléments suivants :
>
> - `MemorySize`est défini sur 4 194 304 Ko (4 Go) avec les paramètres par défaut. Vous pouvez toujours modifier cette valeur. Nous vous conseillons de définir`MemorySize`jusqu'à 128 000 Ko au moins.
> - `Pinned` doit être défini sur `true`.
> - `Timeout` est requis par la version de définition de fonction, mais `GGStreamManager` ne l'utilise pas.

Gestionnaire de flux avec des paramètres personnalisés

L'exemple de configuration suivant active le gestionnaire de flux avec des valeurs personnalisées pour les paramètres de répertoire de stockage, de port du serveur et de taille de groupe de threads.

```
{
    "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 Greengrassutilise les valeurs par défaut pourParamètres du gestionnaire de fluxqui ne sont pas spécifiées en tant que variables d'environnement.

Gestionnaire de flux avec des paramètres personnalisés pour les exportations Amazon S3

L'exemple de configuration suivant active le gestionnaire de flux avec des valeurs personnalisées pour le répertoire de téléchargement et des paramètres de taille de téléchargement partitionné minimum.

```
{
    "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"
}
```

Pour activer, désactiver ou configurer le gestionnaire de flux (interface de ligne de commande)

1. Obtenez les ID du groupe et de la version de groupe Greengrass cible. Cette procédure suppose qu'il s'agit du dernier groupe et de la dernière version du groupe. La requête suivante renvoie le groupe créé le plus récemment.

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

Vous pouvez également procéder à une interrogation par nom. Les noms de groupe ne devant pas nécessairement être uniques, plusieurs groupes peuvent être renvoyés.

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

> **Note**
>
> Vous trouverez également ces valeurs dansAWS IoTconsole L'ID du groupe s'affiche sur la page Paramètres du groupe. Les ID de version de groupe sont affichés sur leDéploiementsTabulation.

2. Copiez les valeurs `LatestVersion` et `Id` du groupe cible dans la sortie.
3. Obtenez la version de groupe la plus récente.

   - Remplacez *`group-id`* par la propriété `Id` que vous avez copiée.
   - Remplacez*`latest-group-version-id`*avec le`LatestVersion`que vous avez copié.

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

4. Copier le`CoreDefinitionVersionArn`et tous les autres ARN de version issus de la sortie, à l'exception`FunctionDefinitionVersionArn`. Vous aurez besoin de ces valeurs ultérieurement lorsque vous créerez une version de groupe.
5. Depuis `FunctionDefinitionVersionArn` dans la sortie, copiez l'ID de la définition de fonction. L'ID est le GUID qui suit le segment `functions` dans l'ARN, comme illustré dans l'exemple suivant.

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

> **Note**
>
> Vous pouvez aussi créer une définition de fonction en exécutant la`create-function-definition`, puis copier l'ID à partir de la sortie.

6. Ajoutez une version de définition de fonction à la définition de fonction.

   - Remplacez*`function-definition-id`*avec le`Id`que vous avez copiée pour la définition de fonction.
   - Dans`functions`tableau, inclut toutes les autres fonctions que vous souhaitez rendre disponibles sur l'appareil principal Greengrass). Vous pouvez utiliser la commande `get-function-definition-version` pour obtenir la liste des fonctions existantes.

   Activer le gestionnaire de flux avec les paramètres par défaut

   L'exemple suivant active le gestionnaire de flux en incluant le`GGStreamManager`dans`functions`. Cet exemple utilise les valeurs par défaut pour les paramètres du gestionnaire de flux.

```
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**
>
> LemyLambdaFunctiondans les exemples représente une de vos fonctions Lambda définies par l'utilisateur.

Activer le gestionnaire de flux avec des paramètres personnalisés

L'exemple suivant active le gestionnaire de flux en incluant la fonction `GGStreamManager` dans le tableau `functions`. Tous les paramètres du gestionnaire de flux sont facultatifs, sauf si vous souhaitez modifier les valeurs par défaut. Cet exemple montre comment utiliser des variables d'environnement pour définir des valeurs personnalisées.

```
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**
>
> Pour`FunctionConfiguration`, vous pouvez connaître les éléments suivants :
>
> - `MemorySize`est défini sur 4 194 304 Ko (4 Go) avec les paramètres par défaut. Vous pouvez toujours modifier cette valeur. Nous vous conseillons de définir`MemorySize`jusqu'à 128 000 Ko au moins.
> - `Pinned` doit être défini sur `true`.
> - `Timeout` est requis par la version de définition de fonction, mais `GGStreamManager` ne l'utilise pas.

Désactiver le gestionnaire de flux

L'exemple suivant omet la fonction `GGStreamManager`, qui désactive le gestionnaire de flux.

```
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**
>
> Si vous ne souhaitez déployer aucune fonction Lambda, vous pouvez omettre entièrement la version de définition de fonction.

7. Copiez l'`Arn` de la version de définition de fonction à partir de la sortie.
8. Créez une version de groupe qui contient la fonction système Lambda

   - Remplacez *`group-id`* par l'`Id` du groupe.
   - Remplacez*`core-definition-version-arn`*avec le`CoreDefinitionVersionArn`que vous avez copié à partir de la version de groupe la plus récente
   - Remplacez*`function-definition-version-arn`*avec le`Arn`que vous avez copiée pour la nouvelle version de définition de fonction.
   - Remplacez les ARN des autres composants de groupe (par exemple `SubscriptionDefinitionVersionArn` ou `DeviceDefinitionVersionArn`) que vous avez copiés de la version de groupe la plus récente.
   - Supprimez tous les paramètres inutilisés. Par exemple, supprimez `--resource-definition-version-arn` si votre version de groupe ne contient aucune ressource.

```
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. Copiez la `Version` à partir de la sortie. Il s'agit de l'ID de la nouvelle version de groupe.
10. Déployez le groupe avec la nouvelle version de groupe.

- Remplacez *group-id* par l'`Id` que vous avez copié pour le groupe.
- Remplacez*group-version-id*avec le`Version`que vous avez copiée pour la nouvelle version de groupe.

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

Si vous souhaitez modifier à nouveau les paramètres du gestionnaire de flux, suivez cette procédure. Assurez-vous de créer une version de définition de fonction incluant le`GGStreamManager`fonction avec la configuration mise à jour. La version du groupe doit faire référence à tous les ARN de version de composant que vous souhaitez déployer sur l'appareil principal (noyau). Les modifications prennent effet après le déploiement du groupe.

Consulter aussi

- Gestion des flux de données
- the section called “Utiliser StreamManagerClient pour travailler avec des flux”
- the section called “Configurations d'exportation pour prises en chargeAWS Clouddestinations”
- the section called “Exportation de flux de données (console)”
- the section called “Exportation de flux de données (interface de ligne de commande)”

Utiliser StreamManagerClient pour travailler avec des flux

Fonctions Lambda définies par l'utilisateur exécutées sur leAWS IoT GreengrassCore peut utiliser le`StreamManagerClient`dans leAWS IoT GreengrassKit SDK Corepour créer des flux dansgestionnaire de fluxpuis interagissez avec les flux. Lorsqu'une fonction Lambda crée un flux, elle définit le paramètreAWS Clouddestinations, hiérarchisation et autres stratégies d'exportation et de conservation des données pour le flux. Pour envoyer des données au gestionnaire de flux, les fonctions Lambda ajoutent les données au flux. Si une destination d'exportation est définie pour le flux, le gestionnaire de flux exporte automatiquement le flux.

> **Note**
>
> Généralement, les clients du gestionnaire de flux sont des fonctions Lambda définies par l'utilisateur. Si votre analyse de rentabilisation l'exige, vous pouvez également autoriser les processus non Lambda exécutés sur le noyau Greengrass (par exemple, un conteneur Docker) à interagir avec le gestionnaire de flux. Pour plus d'informations, consultez the section called “Authentification client”.

Les extraits de cette rubrique vous montrent comment les clients appellent.`StreamManagerClient`méthodes pour travailler avec des flux Pour obtenir des détails d'implémentation concernant les méthodes et leurs arguments, utilisez les liens vers la référence SDK répertoriée après chaque extrait de code. Pour obtenir des didacticiels comprenant une fonction Python Lambda complète, reportez-vous à la sectionthe section called “Exportation de flux de données (console)”outhe section called “Exportation de flux de données (interface de ligne de commande)”.

Votre fonction Lambda doit instancier`StreamManagerClient`en dehors du gestionnaire de fonctions Si la fonction est instanciée dans le gestionnaire, elle crée un `client` et une connexion au gestionnaire de flux chaque fois qu'elle est appelée.

> **Note**
>
> Si vous effectuez une instanciation `StreamManagerClient` dans le gestionnaire, vous devez appeler explicitement la méthode `close()` lorsque le `client` termine son travail. Sinon, le `client` maintient la connexion ouverte et un autre thread actif jusqu'à ce que le script se termine.

`StreamManagerClient` prend en charge les opérations suivantes :

- the section called “Créer un flux de messages”
- the section called “Ajouter un message”
- the section called “Lire des messages”
- the section called “Afficher la liste des flux”
- the section called “Décrire le flux de messages”
- the section called “Met à jour le flux”

- the section called “Supprimer le flux de messages”

Créer un flux de messages

Pour créer un flux, une fonction Lambda définie par l'utilisateur appelle la méthode create et transmet un`MessageStreamDefinition`objet. Cet objet spécifie le nom unique du flux et définit la façon dont le gestionnaire de flux doit gérer les nouvelles données lorsque la taille maximale du flux est atteinte. Vous pouvez utiliser `MessageStreamDefinition` et ses types de données (tels que `ExportDefinition`, `StrategyOnFull` et `Persistence`) pour définir d'autres propriétés de flux. Il s'agit des licences suivantes :

- cibleAWS IoT Analytics, Kinesis Data Streams,AWS IoT SiteWise, et les destinations Amazon S3 pour les exportations automatiques. Pour plus d'informations, consultez the section called “Configurations d'exportation pour prises en chargeAWS Clouddestinations”.
- Exportez la priorité. Le gestionnaire de flux exporte les flux de priorité supérieure avant les flux de priorité inférieure.
- Taille maximale du lot et intervalle de lot pourAWS IoT Analytics, Kinesis Data Streams etAWS IoT SiteWisedestinations. Le gestionnaire de flux exporte les messages lorsque l'une ou l'autre des conditions est remplie.
- Durée de vie (TTL). Temps nécessaire pour garantir que les données du flux sont disponibles pour le traitement. Vous devez vous assurer que les données peuvent être consommées pendant cette période. Il ne s'agit pas d'une stratégie de suppression. Les données peuvent ne pas être supprimées immédiatement après la période de TTL.
- Persistance des flux. Choisissez d'enregistrer les flux dans le système de fichiers afin de conserver les données lors des redémarrages du noyau ou d'enregistrer les flux en mémoire.
- Numéro de séquence de début Spécifiez le numéro de séquence du message à utiliser comme message de départ dans l'exportation.

Pour plus d'informations sur`MessageStreamDefinition`, consultez la référence SDK de votre langue cible :

- MessageStreamDefinitiondans le kit SDK Java
- MessageStreamDefinitiondans le kit SDK Node.js
- MessageStreamDefinitiondans le kit SDK Python

> **Note**
>
> `StreamManagerClient`fournit également une destination cible que vous pouvez utiliser pour exporter des flux vers un serveur HTTP. Cette cible n'est destinée qu'à des fins de test. Il n'est pas stable ni pris en charge pour une utilisation dans des environnements de production.

Après la création d'un flux, vos fonctions Lambda peuventajouter des messagesvers le flux pour envoyer des données pour exportation etLire des messagesdepuis le flux pour le traitement local. Le nombre de flux que vous créez dépend de vos capacités matérielles et de votre analyse de rentabilisation. Une stratégie consiste à créer un flux pour chaque canal cible dansAWS IoT Analyticsou Kinesis, bien que vous puissiez définir plusieurs cibles pour un flux. Un flux a un cycle de vie durable.

Prérequis

Cette opération possède les critères suivants :

- MinimumAWS IoT GreengrassVersion Core : 1.10.0
- MinimumAWS IoT GreengrassVersion du kit SDK Core : Python : 1.5.0 | Java : 1.4.0 | Node.js : 1.6.0

> **Note**
>
> Création de flux avec unAWS IoT SiteWiseou la destination d'exportation Amazon S3 possède les critères suivants :
>
> - MinimumAWS IoT GreengrassVersion Core : 1.11.0
> - MinimumAWS IoT GreengrassVersion du kit SDK Core : Python : 1.6.0 | Java : 1.5.0 | Node.js : 1.7.0

Exemples

L'extrait de code suivant crée un flux nommé `StreamName`. Il définit les propriétés de flux dans le`MessageStreamDefinition`et les types de données subordonnés.

Python

```
client = StreamManagerClient()

try:
    client.create_message_stream(MessageStreamDefinition(
        name="StreamName",  # Required.
        max_size=268435456,  # Default is 256 MB.
        stream_segment_size=16777216,  # Default is 16 MB.
        time_to_live_millis=None,  # By default, no TTL is enabled.
        strategy_on_full=StrategyOnFull.OverwriteOldestData,  # Required.
        persistence=Persistence.File,  # Default is File.
        flush_on_write=False,  # Default is false.
        export_definition=ExportDefinition(  # Optional. Choose where/how the stream
 is exported to the AWS Cloud.
            kinesis=None,
            iot_analytics=None,
            iot_sitewise=None,
            s3_task_executor=None
        )
    ))
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
    # Properly handle errors.
```

Référence du kit 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 AWS Cloud.
                            new ExportDefinition()
                                    .withKinesis(null)
                                    .withIotAnalytics(null)
                                    .withIotSitewise(null)
                                    .withS3TaskExecutor(null)
                    )

    );
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

Référence du kit SDK Java :createMessageStream|MessageStreamDefinition

Node.js

```
const client = new StreamManagerClient();
client.onConnected(async () => {
    try {
        await client.createMessageStream(
            new MessageStreamDefinition()
                .withName("StreamName") // Required.
                .withMaxSize(268435456)  // Default is 256 MB.
                .withStreamSegmentSize(16777216)  // Default is 16 MB.
                .withTimeToLiveMillis(null)  // By default, no TTL is enabled.
                .withStrategyOnFull(StrategyOnFull.OverwriteOldestData)  //
 Required.
                .withPersistence(Persistence.File)  // Default is File.
                .withFlushOnWrite(false)  // Default is false.
                .withExportDefinition(  // Optional. Choose where/how the stream is
 exported to the AWS Cloud.
                    new ExportDefinition()
                        .withKinesis(null)
                        .withIotAnalytics(null)
                        .withIotSitewise(null)
                        .withS3TaskExecutor(null)
                )
        );
    } catch (e) {
        // Properly handle errors.
    }
});
```

```
client.onError((err) => {
    // Properly handle connection errors.
    // This is called only when the connection to the StreamManager server fails.
});
```

Référence du kit SDK Node.js :createMessageStream|MessageStreamDefinition

Pour plus d'informations sur la configuration des destinations d'exportation, consultezthe section called “Configurations d'exportation pour prises en chargeAWS Clouddestinations”.

Ajouter un message

Pour envoyer des données vers le gestionnaire de flux pour exportation, vos fonctions Lambda ajoutent les données au flux cible. La destination d'exportation détermine le type de données à transmettre à cette méthode.

Prérequis

Cette opération est soumise aux exigences suivantes :

- MinimumAWS IoT GreengrassVersion Core : 1.10.0
- MinimumAWS IoT GreengrassVersion du kit SDK Core : Python : 1.5.0 | Java : 1.4.0 | Node.js : 1.6.0

> **Note**
>
> Ajout de messages avec unAWS IoT SiteWiseou la destination d'exportation Amazon S3 possède les critères suivants :
>
> - MinimumAWS IoT GreengrassVersion Core : 1.11.0
> - MinimumAWS IoT GreengrassVersion du kit SDK Core : Python : 1.6.0 | Java : 1.5.0 | Node.js : 1.7.0

Exemples

AWS IoT Analyticsou des destinations d'exportation Kinesis Data Streams

L'extrait de code suivant ajoute un message au flux nommé `StreamName`. PourAWS IoT Analyticsou les destinations Kinesis Data Streams, vos fonctions Lambda ajoutent un blob de données.

Cet extrait de code possède les critères suivants :

- MinimumAWS IoT GreengrassVersion Core : 1.10.0
- MinimumAWS IoT GreengrassVersion du kit SDK Core : 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.
```

Référence du kit SDK Python :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.
}
```

Référence du kit SDK Java :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.
});
```

Référence du kit SDK Node.js :appendMessage

AWS IoT SiteWisedestinations d'exportation

L'extrait de code suivant ajoute un message au flux nommé `StreamName`.
PourAWS IoT SiteWisedestinations, vos fonctions Lambda ajoutent un objet sérialisé`PutAssetPropertyValueEntry`objet. Pour plus d'informations, consultez the section called “Exportation vers AWS IoT SiteWise”.

> **Note**
>
> Lorsque vous envoyez des données àAWS IoT SiteWise, vos données doivent répondre aux exigences du`BatchPutAssetPropertyValue`action. Pour de plus amples informations, veuillez consulter BatchPutasSetPropertyValue dans la référence de l'API AWS IoT SiteWise.

Cet extrait de code possède les critères suivants :

- MinimumAWS IoT GreengrassVersion Core : 1.11.0
- MinimumAWS IoT GreengrassVersion du kit SDK Core : 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.
```

Référence du kit 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.
}
```

Référence du kit SDK Java :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.
});
```

Référence du kit SDK Node.js :appendMessage|PutAssetPropertyValueEntry

Destinations d'exportation Amazon S3

L'extrait de code suivant ajoute une tâche d'exportation au flux nomméStreamName.
Pour les destinations Amazon S3, vos fonctions Lambda ajoutent un objet sérialiséS3ExportTaskDefinitionqui contient les informations sur le fichier d'entrée source et l'objet Amazon S3 cible. Si l'objet spécifié n'existe pas, Stream Manager le crée pour vous. Pour plus d'informations, consultez the section called “Exporter vers Amazon S3”.

Cet extrait de code possède les critères suivants :

- MinimumAWS IoT GreengrassVersion Core : 1.11.0
- MinimumAWS IoT GreengrassVersion du kit SDK Core : 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.
```

Référence du kit SDK Python :append_message|Définition de la tâche d'exportation S3

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.
}
```

Référence du kit SDK Java :appendMessage|Définition de la tâche d'exportation S3

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.
});
```

Référence du kit SDK Node.js :appendMessage|Définition de la tâche d'exportation S3

Lire des messages

Lire des messages à partir d'un flux.

Prérequis

Cette opération est soumise aux exigences suivantes :

- MinimumAWS IoT GreengrassVersion Core : 1.10.0
- MinimumAWS IoT GreengrassVersion du kit SDK Core : Python : 1.5.0 | Java : 1.4.0 | Node.js : 1.6.0

Exemples

L'extrait de code suivant lit les messages du flux nommé `StreamName`. La méthode read utilise un objet `ReadMessagesOptions` facultatif qui spécifie le numéro de séquence à partir duquel commencer la lecture, les nombres minimum et maximum à lire et un délai d'expiration pour la lecture des messages.

Python

```
client = StreamManagerClient()

try:
    message_list = client.read_messages(
        stream_name="StreamName",
        # By default, if no options are specified, it tries to read one message from
 the beginning of the stream.
        options=ReadMessagesOptions(
            desired_start_sequence_number=100,
```

```
            # Try to read from sequence number 100 or greater. By default, this is
 0.
            min_message_count=10,
            # Try to read 10 messages. If 10 messages are not available, then
 NotEnoughMessagesException is raised. By default, this is 1.
            max_message_count=100,  # Accept up to 100 messages. By default this is
 1.
            read_timeout_millis=5000
            # Try to wait at most 5 seconds for the min_messsage_count to be
 fulfilled. By default, this is 0, which immediately returns the messages or an
 exception.
        )
    )
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
    # Properly handle errors.
```

Référence du kit 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_messsage_count to
 be fulfilled. By default, this is 0, which immediately returns the messages or an
 exception.
                    .withReadTimeoutMillis(Duration.ofSeconds(5L).toMillis())
    );
} catch (StreamManagerException e) {
```

```
    // Properly handle exception.
}
```

Référence du kit SDK Java :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.
});
```

Référence du kit SDK Node.js :readMessages|ReadMessagesOptions

Afficher la liste des flux

Obtenez la liste des flux dans le gestionnaire de flux.

Prérequis

Cette opération est soumise aux exigences suivantes :

- MinimumAWS IoT GreengrassVersion Core : 1.10.0
- MinimumAWS IoT GreengrassVersion du kit SDK Core : Python : 1.5.0 | Java : 1.4.0 | Node.js : 1.6.0

Exemples

L'extrait de code suivant récupère une liste des flux (par nom) dans le gestionnaire de flux.

Python

```
client = StreamManagerClient()

try:
    stream_names = client.list_streams()
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
    # Properly handle errors.
```

Référence du kit SDK Python :list_streams

Java

```
try (final StreamManagerClient client =
 GreengrassClientBuilder.streamManagerClient().build()) {
    List<String> streamNames = client.listStreams();
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

Référence du kit SDK Java :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.
});
```

Référence du kit SDK Node.js :listStreams

Décrire le flux de messages

Obtenez des métadonnées sur un flux, en particulier la définition, la taille et l'état d'exportation du flux.

Prérequis

Cette opération est soumise aux exigences suivantes :

- MinimumAWS IoT GreengrassVersion Core : 1.10.0
- MinimumAWS IoT GreengrassVersion du kit SDK Core : Python : 1.5.0 | Java : 1.4.0 | Node.js : 1.6.0

Exemples

L'extrait de code suivant récupère des métadonnées sur le flux nommé `StreamName`, en particulier la définition, la taille et les statuts d'exportation du flux.

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.
```

Référence du kit 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.
}
```

Référence du kit SDK Java :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.
});
```

Référence du kit SDK Node.js :describeMessageStream

Met à jour le flux

Met à jour les propriétés d'un flux existant. Il est possible que vous souhaitiez mettre à jour un flux si vos besoins changent après la création du flux. Par exemple :

- Ajouter un nouveauConfiguration d'exportationpour unAWS Clouddestination.
- Augmentez la taille maximale d'un flux pour modifier la façon dont les données sont exportées ou conservées. Par exemple, la taille du flux associée à votre stratégie sur les paramètres complets peut entraîner la suppression ou le rejet des données avant que le gestionnaire de flux puisse les traiter.
- Suspendre et reprendre les exportations ; par exemple, si les tâches d'exportation sont longues et que vous souhaitez rationner vos données de chargement.

Vos fonctions Lambda suivent ce processus de haut niveau pour mettre à jour un flux :

1. Obtenez la description du flux.
2. Mettre à jour les propriétés cibles sur les propriétés correspondantes`MessageStreamDefinition`et des objets subordonnés.
3. Passez dans la mise à jour`MessageStreamDefinition`. Assurez-vous d'inclure les définitions d'objets complètes pour le flux mis à jour. Les propriétés non définies reprennent les valeurs par défaut.

   Vous pouvez spécifier le numéro de séquence du message à utiliser comme message de départ dans l'exportation.

Prérequis

Cette opération est soumise aux exigences suivantes :

- MinimumAWS IoT GreengrassVersion Core : 1.11.0
- MinimumAWS IoT GreengrassVersion du kit SDK Core : Python : 1.6.0 | Java : 1.5.0 | Node.js : 1.7.0

Exemples

L'extrait de code suivant met à jour le flux nommé`StreamName`. Il met à jour plusieurs propriétés d'un flux exporté vers 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.
```

Référence du kit SDK Python :Mettre à jour le flux de messages|MessageStreamDefinition

Java

```
try (final StreamManagerClient client =
 GreengrassClientBuilder.streamManagerClient().build()) {
    MessageStreamInfo messageStreamInfo = client.describeMessageStream(STREAM_NAME);
    // Update the message stream with new values.
    client.updateMessageStream(
        messageStreamInfo.getDefinition()
            .withStrategyOnFull(StrategyOnFull.RejectNewData) // Required. Updating
 Strategy on full to reject new data.
            // Max Size update should be greater than initial Max Size defined in
 Create Message Stream request
            .withMaxSize(536870912L) // Update Max Size to 512 MB.
            .withStreamSegmentSize(33554432L) // Update Segment Size to 32 MB.
            .withFlushOnWrite(true) // Update flush on write to true.
            .withPersistence(Persistence.Memory) // Update the persistence to
 Memory.
            .withTimeToLiveMillis(3600000L)  // Update TTL to 1 hour.
            .withExportDefinition(
                // Optional. Choose where/how the stream is exported to the AWS
 Cloud.
                messageStreamInfo.getDefinition().getExportDefinition().
                    // Updating Export definition to add a Kinesis Stream
 configuration.
                    .withKinesis(new ArrayList<KinesisConfig>() {{
                        add(new KinesisConfig()
                            .withIdentifier(EXPORT_IDENTIFIER)
                            .withKinesisStreamName("test"));
                        }})
            );
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

Référence du kit SDK Java :update_message_stream|MessageStreamDefinition

Node.js

```
const client = new StreamManagerClient();
client.onConnected(async () => {
    try {
        const messageStreamInfo = await c.describeMessageStream(STREAM_NAME);
        await client.updateMessageStream(
            messageStreamInfo.definition
                // Max Size update should be greater than initial Max Size defined
 in Create Message Stream request
                .withMaxSize(536870912)  // Default is 256 MB. Updating Max Size to
 512 MB.
                .withStreamSegmentSize(33554432)  // Default is 16 MB. Updating
 Segment Size to 32 MB.
                .withTimeToLiveMillis(3600000)  // By default, no TTL is enabled.
 Update TTL to 1 hour.
                .withStrategyOnFull(StrategyOnFull.RejectNewData)  // Required.
 Updating Strategy on full to reject new data.
                .withPersistence(Persistence.Memory)  // Default is File. Update the
 persistence to Memory
                .withFlushOnWrite(true)  // Default is false. Updating to true.
                .withExportDefinition(
                    // Optional. Choose where/how the stream is exported to the AWS
 Cloud.
                    messageStreamInfo.definition.exportDefinition
                        // Updating Export definition to add a Kinesis Stream
 configuration.
                        .withKinesis([new
 KinesisConfig().withIdentifier(uuidv4()).withKinesisStreamName(uuidv4())])
                )
        );
    } catch (e) {
        // Properly handle errors.
    }
});
client.onError((err) => {
    // Properly handle connection errors.
    // This is called only when the connection to the StreamManager server fails.
});
```

Référence du kit SDK Node.js :Mettre à jour le flux de messages|MessageStreamDefinition

Contraintes de mise à jour des flux

Les contraintes suivantes s'appliquent quand vous mettez à jour les flux. Sauf indication dans la liste suivante, les mises à jour prennent effet immédiatement.

- Vous ne pouvez pas mettre à jour la persistance d'un flux. Pour modifier ce comportement,supprimer le fluxetcréer un fluxqui définit la nouvelle politique de persistance.
- Vous pouvez mettre à jour la taille maximale d'un flux uniquement dans les conditions suivantes :
  - La taille maximale doit être supérieure ou égale à la taille actuelle du flux. Pour trouver ces informations,décrire le fluxpuis vérifiez l'état de stockage du fichier renvoyé`MessageStreamInfo`objet.
  - La taille maximale doit être supérieure ou égale à la taille du segment du flux.
- Vous pouvez mettre à jour la taille du segment de flux à une valeur inférieure à la taille maximale du flux. Le paramètre mis à jour s'applique aux nouveaux segments.
- Les mises à jour de la propriété de durée de vie (TTL) s'appliquent aux nouvelles opérations d'ajout. Si vous diminuez cette valeur, le gestionnaire de flux peut également supprimer des segments existants qui dépassent le TTL.
- Les mises à jour de la stratégie sur la propriété complète s'appliquent aux nouvelles opérations d'ajout. Si vous définissez la stratégie pour écraser les données les plus anciennes, le gestionnaire de flux peut également écraser les segments existants en fonction du nouveau paramètre.
- Les mises à jour de la propriété Flush on write s'appliquent aux nouveaux messages.
- Les mises à jour des configurations d'exportation s'appliquent aux nouvelles exportations. La demande de mise à jour doit inclure toutes les configurations d'exportation que vous souhaitez prendre en charge. Sinon, le gestionnaire de flux les supprime.
  - Lorsque vous mettez à jour une configuration d'exportation, spécifiez l'identifiant de la configuration d'exportation cible.
  - Pour ajouter une configuration d'exportation, spécifiez un identifiant unique pour la nouvelle configuration d'exportation.
  - Pour supprimer une configuration d'exportation, omettez la configuration d'exportation.
- Pourmise à journuméro de séquence de départ d'une configuration d'exportation dans un flux, vous devez spécifier une valeur inférieure au dernier numéro de séquence. Pour trouver ces informations,décrire le fluxpuis vérifiez l'état de stockage du fichier renvoyé`MessageStreamInfo`objet.

Supprimer le flux de messages

Supprime un flux. Lorsque vous supprimez un flux, toutes les données stockées dans le flux sont supprimées du disque.

Prérequis

Cette opération possède les critères suivants :

- MinimumAWS IoT GreengrassVersion Core : 1.10.0
- MinimumAWS IoT GreengrassVersion du kit SDK Core : Python : 1.5.0 | Java : 1.4.0 | Node.js : 1.6.0

Exemples

L'extrait de code suivant supprime le flux nommé `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.
```

Référence du kit SDK Python :deleteMessageStream

Java

```
try (final StreamManagerClient client =
 GreengrassClientBuilder.streamManagerClient().build()) {
    client.deleteMessageStream("StreamName");
} catch (StreamManagerException e) {
    // Properly handle exception.
```

```
}
```

Référence du kit SDK Java :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.
});
```

Référence du kit SDK Node.js :deleteMessageStream

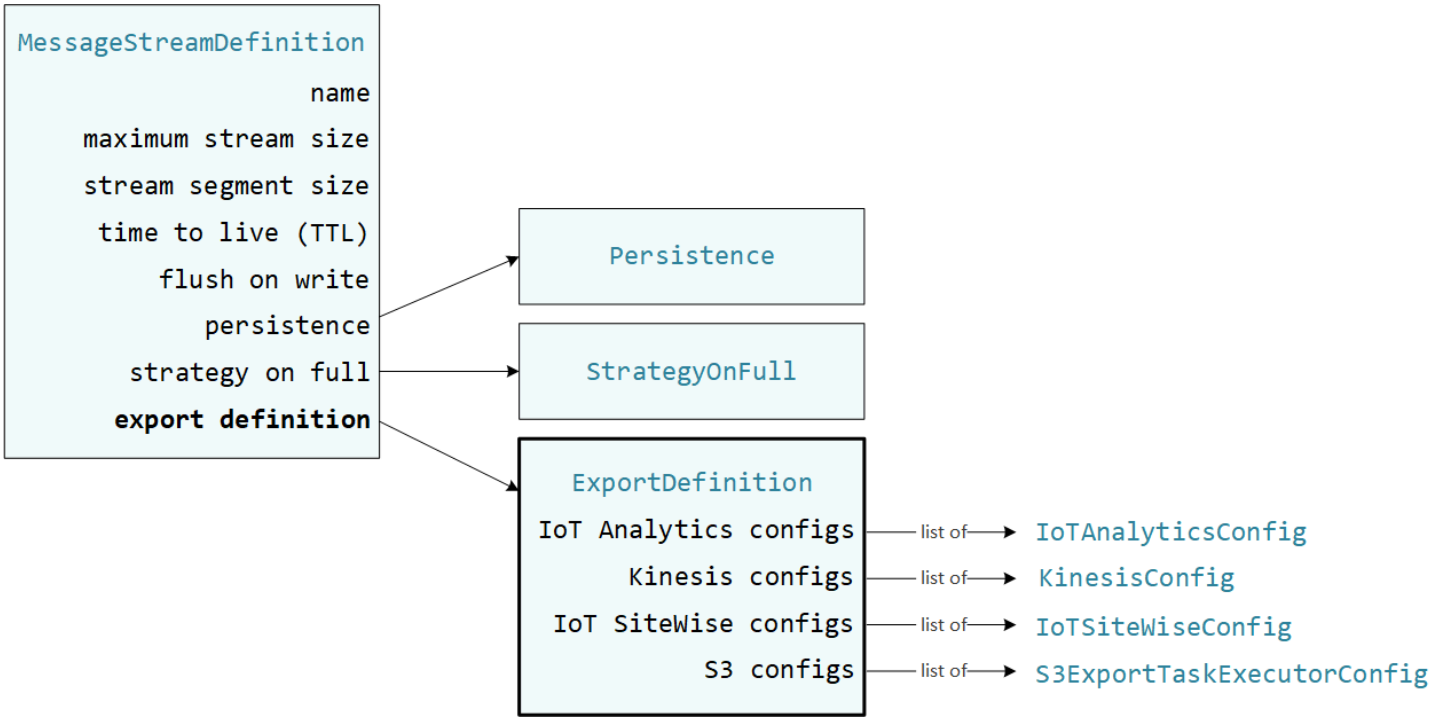
Consulter aussi

- Gestion des flux de données
- the section called “Configuration du gestionnaire de flux”
- the section called “Configurations d'exportation pour prises en chargeAWS Clouddestinations”
- the section called “Exportation de flux de données (console)”
- the section called “Exportation de flux de données (interface de ligne de commande)”
- `StreamManagerClient`dans leAWS IoT GreengrassRéférence du kit SDK principal :
  - Python
  - Java
  - Node.js

Configurations d'exportation pour prises en chargeAWS Clouddestinations

Utilisation des fonctions Lambda définies par l'utilisateur`StreamManagerClient`dans leAWS IoT GreengrassKit SDK principal pour interagir avec le gestionnaire de flux. Quand une fonction

Lambdacrée un fluxoumet à jour un flux, il passe un`MessageStreamDefinition`objet qui représente les propriétés du flux, y compris la définition d'exportation. Le`ExportDefinition`contient les configurations d'exportation définies pour le flux. Stream Manager utilise ces configurations d'exportation pour déterminer où et comment exporter le flux.



Vous pouvez définir zéro configuration d'exportation ou plus sur un flux, y compris plusieurs configurations d'exportation pour un seul type de destination. Par exemple, vous pouvez exporter un flux vers deuxAWS IoT Analyticset un flux de données Kinesis.

En cas d'échec des tentatives d'exportation, le gestionnaire de flux tente continuellement d'exporter des données vers leAWS Cloudà intervalles allant jusqu'à cinq minutes. Le nombre de nouvelles tentatives n'a pas de limite maximale.

> **Note**
>
> `StreamManagerClient`fournit également une destination cible que vous pouvez utiliser pour exporter des flux vers un serveur HTTP. Cette cible n'est destinée qu'à des fins de test. Il n'est pas stable ni pris en charge pour une utilisation dans des environnements de production.

prises en chargeAWS Clouddestinations

- Canaux AWS IoT Analytics
- Amazon Kinesis Data Streams
- AWS IoT SiteWisepropriétés des ressources
- Objets Amazon S3

Vous êtes responsable de la maintenance de cesAWS CloudAWS.

Canaux AWS IoT Analytics

Stream Manager prend en charge les exportations automatiques versAWS IoT Analytics.AWS IoT Analyticsvous permet d'effectuer des analyses avancées de vos données pour aider à prendre des décisions commerciales et à améliorer les modèles d'apprentissage automatique. Pour de plus amples informations, veuillez consulterPrésentation d'AWS IoT Analytics?dans leAWS IoT AnalyticsGuide de l'utilisateur.

DansAWS IoT GreengrassSDK Core, vos fonctions Lambda utilisent le`IoTAnalyticsConfig`pour définir la configuration d'exportation pour ce type de destination. Pour plus d'informations, consultez la référence SDK de votre langue cible :

- Configuration IoAnalyticsdans le kit SDK Python SDK
- Configuration IoAnalyticsdans le kit SDK Java
- Configuration IoAnalyticsdans le kit SDK Node.js

Prérequis

Cette destination d'exportation possède les critères suivants :

- Canaux cibles dansAWS IoT Analyticsdoivent se trouver dans la mêmeCompte AWSetRégion AWScomme le groupe Greengrass.
- Lethe section called “Rôle de groupe Greengrass”doit permettre à la`iotanalytics:BatchPutMessage`autorisation de cibler les chaînes. Par exemple :

```
{
    "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"
            ]
        }
    ]
}
```

Vous pouvez octroyer un accès précis ou conditionnel aux ressources, par exemple, en utilisant un caractère générique*méthode de dénomination. Pour de plus amples informations, veuillez consulterAjout et suppression de politiques IAMdans leIAM User Guide.

Exportation vers AWS IoT Analytics

Pour créer un flux qui exporte versAWS IoT Analytics, vos fonctions LambdaCréer un fluxavec une définition d'exportation comprenant une ou plusieurs`IoTAnalyticsConfig`objets. Cet objet définit les paramètres d'exportation, tels que le canal cible, la taille du lot, l'intervalle de lot et la priorité.

Lorsque vos fonctions Lambda reçoivent des données provenant d'appareils, ellesAjouter des messagesqui contiennent un blob de données vers le flux cible.

Le gestionnaire de flux exporte ensuite les données en fonction des paramètres de lot et de la priorité définis dans les configurations d'exportation du flux.

Amazon Kinesis Data Streams

Le gestionnaire de flux prend en charge les exportations automatiques vers Amazon Kinesis Data Streams. Kinesis Data Streams est couramment utilisé pour agréger des données à volume élevé et les charger dans un entrepôt de données ou un cluster de réduction de mappe. Pour de plus amples informations, veuillez consulterQu'est-ce qu'Amazon Kinesis Data Streams ?dans leManuel du développeur Amazon Kinesis.

DansAWS IoT GreengrassSDK Core, vos fonctions Lambda utilisent le`KinesisConfig`pour définir la configuration d'exportation pour ce type de destination. Pour plus d'informations, consultez la référence SDK de votre langue cible :

- Configuration Kinesisdans le kit SDK Python SDK
- Configuration Kinesisdans le kit SDK Java
- Configuration Kinesisdans le kit SDK Node.js

Prérequis

Cette destination d'exportation possède les critères suivants :

- Les flux cibles dans Kinesis Data Streams doivent se trouver dans le mêmeCompte AWSetRégion AWScomme le groupe Greengrass.
- Lethe section called “Rôle de groupe Greengrass”doit permettre à la`kinesis:PutRecords`autorisation de cibler les flux de données. Par exemple :

```
{
    "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"
            ]
        }
    ]
}
```

  Vous pouvez octroyer un accès précis ou conditionnel aux ressources, par exemple, en utilisant un caractère générique`*`méthode de dénomination. Pour de plus amples informations, veuillez consulterAjout et suppression de politiques IAMdans leIAM User Guide.

Exporter vers Kinesis Data Streams

Pour créer un flux qui exporte vers Kinesis Data Streams, votre Lambda fonctionneCréer un fluxavec une définition d'exportation comprenant une ou plusieurs`KinesisConfig`objets. Cet objet définit les paramètres d'exportation, tels que le flux de données cible, la taille du lot, l'intervalle de lot et la priorité.

Lorsque vos fonctions Lambda reçoivent des données provenant d'appareils, elles Ajouter des messages qui contiennent un blob de données vers le flux cible. Le gestionnaire de flux exporte ensuite les données en fonction des paramètres de lot et de la priorité définis dans les configurations d'exportation du flux.

Stream Manager génère un UUID aléatoire unique en tant que clé de partition pour chaque enregistrement téléchargé sur Amazon Kinesis.

AWS IoT SiteWisepropriétés des ressources

Stream Manager prend en charge les exportations automatiques versAWS IoT SiteWise.AWS IoT SiteWisevous permet de collecter, d'organiser et d'analyser à grande échelle les données provenant d'équipements industriels. Pour de plus amples informations, veuillez consulterPrésentation d'AWS IoT SiteWise?dans leAWS IoT SiteWiseGuide de l'utilisateur.

DansAWS IoT GreengrassSDK Core, vos fonctions Lambda utilisent le`IoTSiteWiseConfig`pour définir la configuration d'exportation pour ce type de destination. Pour plus d'informations, consultez la référence SDK de votre langue cible :

- Configuration IoT SiteWisedans le kit SDK Python SDK
- Configuration IoT SiteWisedans le kit SDK Java
- Configuration IoT SiteWisedans le kit SDK Node.js

> **Note**
>
> AWSfournit également lethe section called “IoT SiteWise”, une solution prédéfinie que vous pouvez utiliser avec des sources OPC-UA.

Prérequis

Cette destination d'exportation possède les critères suivants :

- Propriétés des ressources cibles dansAWS IoT SiteWisedoivent se trouver dans la mêmeCompte AWSetRégion AWScomme le groupe Greengrass.

> **Note**
>
> Pour obtenir la liste des régions quiAWS IoT SiteWisesupports, voirAWS IoT SiteWisePoints de terminaison et quotasdans leAWSRéférence générale.

- Lethe section called “Rôle de groupe Greengrass”doit permettre à laiotsitewise:BatchPutAssetPropertyValueautorisation de cibler les propriétés des ressources. L'exemple de stratégie suivant utilise leiotsitewise:assetHierarchyPathafin d'octroyer l'accès à une ressource racine cible et à ses enfants. Vous pouvez supprimer l'optionConditiondepuis la stratégie pour permettre l'accès à tous vosAWS IoT SiteWiseou spécifiez les ARN des actifs individuels.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
             "Effect": "Allow",
             "Action": "iotsitewise:BatchPutAssetPropertyValue",
             "Resource": "*",
             "Condition": {
                 "StringLike": {
                     "iotsitewise:assetHierarchyPath": [
                         "/root node asset ID",
                         "/root node asset ID/*"
                     ]
                 }
             }
        }
    ]
}
```

Vous pouvez octroyer un accès précis ou conditionnel aux ressources, par exemple, en utilisant un caractère générique*méthode de dénomination. Pour de plus amples informations, veuillez consulterAjout et suppression de politiques IAMdans leIAM User Guide.

Pour plus d'informations importantes sur la sécurité, consultez BatchPutAssetPropertyValue autorisationdans leAWS IoT SiteWiseGuide de l'utilisateur.

Exportation vers AWS IoT SiteWise

Pour créer un flux qui exporte versAWS IoT SiteWise, vos fonctions LambdaCréer un fluxavec une définition d'exportation comprenant une ou plusieurs`IoTSiteWiseConfig`objets. Cet objet définit les paramètres d'exportation, tels que la taille du lot, l'intervalle de lot et la priorité.

Lorsque vos fonctions Lambda reçoivent des données de propriété de ressources provenant d'appareils, elles ajoutent des messages contenant les données au flux cible. Les messages sont sérialisés en JSON`PutAssetPropertyValueEntry`objets contenant des valeurs de propriétés pour une ou plusieurs propriétés d'actifs. Pour de plus amples informations, veuillez consulterAjouter un messagepourAWS IoT SiteWisedestinations d'exportation.

> **Note**
>
> Lorsque vous envoyez des données àAWS IoT SiteWise, vos données doivent répondre aux exigences de la`BatchPutAssetPropertyValue`action. Pour de plus amples informations, veuillez consulter BatchPutasSetPropertyValue dans la référence de l'API AWS IoT SiteWise.

Le gestionnaire de flux exporte ensuite les données en fonction des paramètres de lot et de la priorité définis dans les configurations d'exportation du flux.

Vous pouvez ajuster les paramètres de votre gestionnaire de flux et la logique des fonctions Lambda pour concevoir votre stratégie d'exportation. Par exemple :

- Pour les exportations quasi en temps réel, définissez des paramètres de taille de lot et d'intervalle faibles et ajoutez les données au flux lors de leur réception.
- Pour optimiser le traitement par lots, atténuer les contraintes de bande passante ou réduire les coûts, vos fonctions Lambda peuvent regrouper les timestamp-quality-value (TQV) reçus pour une propriété d'actif unique avant d'ajouter les données au flux. Une stratégie consiste à regrouper des entrées pour un maximum de 10 combinaisons de propriétés et d'actifs différents, ou alias de propriété, dans un message au lieu d'envoyer plusieurs entrées pour la même propriété. Cela permet au gestionnaire de flux de rester dansAWS IoT SiteWisequotas.

Objets Amazon S3

Le gestionnaire de flux prend en charge les exportations automatiques vers Amazon S3. Vous pouvez utiliser Amazon S3 pour stocker et récupérer de grandes quantités de données. Pour de plus amples informations, veuillez consulterQu'est-ce qu'Amazon S3 ?dans leManuel du développeur Amazon Simple Storage Service.

DansAWS IoT GreengrassSDK Core, vos fonctions Lambda utilisent le`S3ExportTaskExecutorConfig`pour définir la configuration d'exportation pour ce type de destination. Pour plus d'informations, consultez la référence SDK de votre langue cible :

- Configuration de l'exécuteur de tâches d'exportation S3dans le kit SDK Python SDK
- Configuration de l'exécuteur de tâches d'exportation S3dans le kit SDK Java
- Configuration de l'exécuteur de tâches d'exportation S3dans le kit SDK Node.js

Prérequis

Cette destination d'exportation possède les critères suivants :

- Les compartiments Amazon S3 ciblés doivent se trouver dans le même compartimentCompte AWScomme le groupe Greengrass.
- Si l'icôneconteneurisation par défautpour le groupe Greengrass estConteneur Greengrass, vous devez définir le paramètreSTREAM_MANAGER_READ_ONLY_DIRSpour utiliser un répertoire de fichiers d'entrée situé sous`/tmp`ou n'est pas sur le système de fichiers racine.
- Si une fonction Lambda s'exécute dansConteneur Greengrassmode écrit les fichiers d'entrée dans le répertoire du fichier d'entrée, vous devez créer une ressource de volume local pour le répertoire et monter le répertoire dans le conteneur avec des autorisations d'écriture. Cela garantit que les fichiers sont écrits dans le système de fichiers racine et visibles en dehors du conteneur. Pour plus d'informations, consultez Accès aux ressources locales.
- Lethe section called “Rôle de groupe Greengrass”doit autoriser les autorisations suivantes sur les compartiments cibles. Par exemple :

```
{
    "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/*"
            ]
        }
    ]
}
```

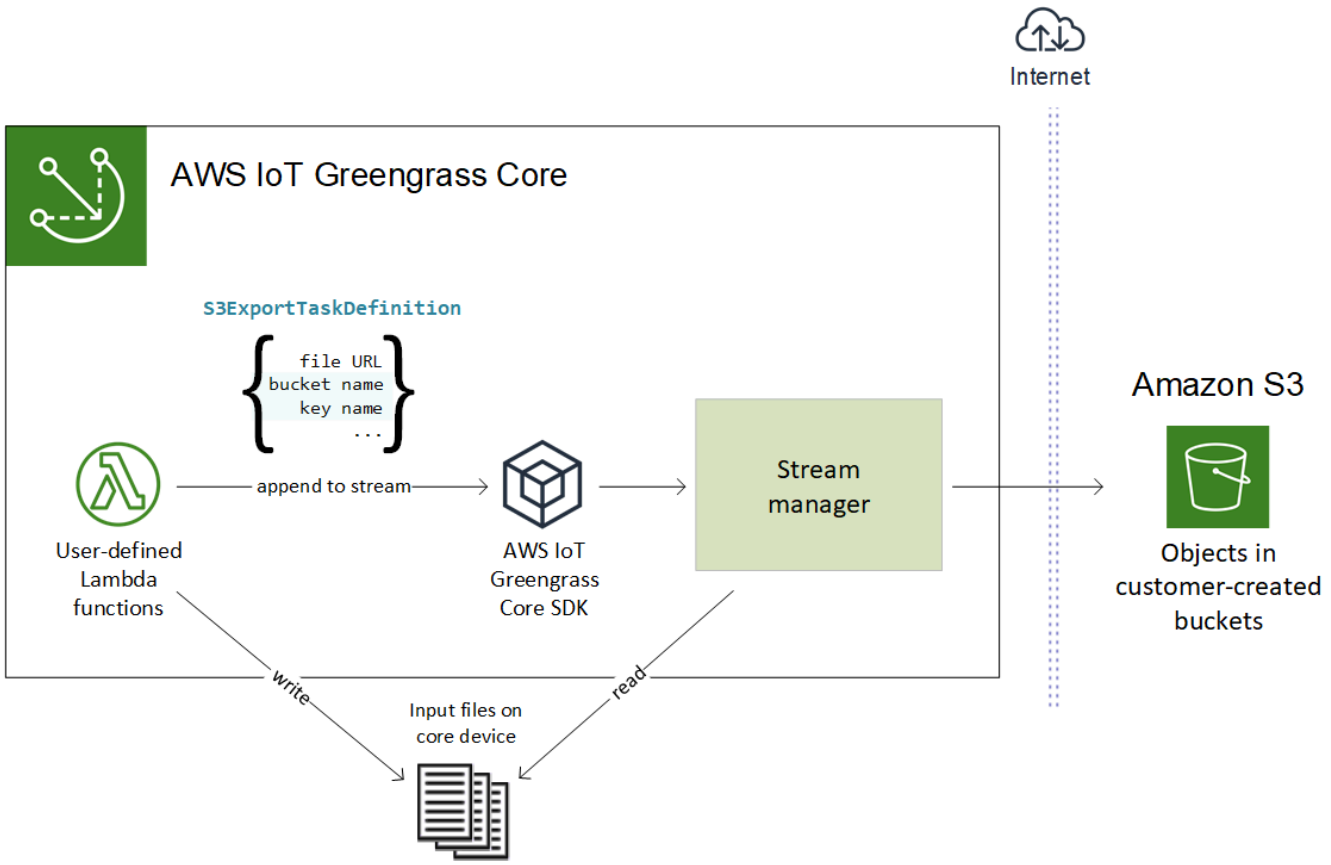
Vous pouvez octroyer un accès précis ou conditionnel aux ressources, par exemple, en utilisant un caractère générique*méthode de dénomination. Pour de plus amples informations, veuillez consulterAjout et suppression de politiques IAMdans leIAM User Guide.

Exporter vers Amazon S3

Pour créer un flux qui exporte vers Amazon S3, vos fonctions Lambda utilisent le`S3ExportTaskExecutorConfig`pour configurer la stratégie d'exportation. La stratégie définit les paramètres d'exportation, tels que le seuil de chargement partitionné et la priorité. Pour les exportations Amazon S3, Stream Manager télécharge les données qu'il lit à partir de fichiers locaux sur le périphérique principal. Pour lancer un chargement, vos fonctions Lambda ajoutent une tâche d'exportation au flux cible. La tâche d'exportation contient des informations sur le fichier d'entrée et l'objet Amazon S3 cible. Le gestionnaire de flux exécute les tâches dans la séquence dans laquelle elles sont ajoutées au flux.

> **Note**
>
> Le compartiment cible doit déjà exister dans votreCompte AWS. Si un objet pour la clé spécifiée n'existe pas, le gestionnaire de flux crée l'objet pour vous.

Ce flux de travail de haut niveau est illustré dans le diagramme suivant.



Le gestionnaire de flux utilise la propriété seuil de chargement partitionné,Taille de partie minimumet la taille du fichier d'entrée pour déterminer comment charger les données. Le seuil de téléchargement partitionné doit être supérieur ou égal à la taille d'article minimale. Si vous souhaitez télécharger des données en parallel, vous pouvez créer plusieurs flux.

Les clés qui spécifient vos objets Amazon S3 cibles peuvent inclure des clés validesDateTimeFormatter datetimeJavachaînes dans`!{timestamp:`*`value`*`}`espaces réservés. Vous pouvez utiliser ces espaces réservés à l'horodatage pour partitionner les données dans Amazon S3 en fonction de l'heure à laquelle les données du fichier d'entrée ont été téléchargées. Par exemple, le nom de clé suivant se résout en une valeur telle que`my-key/2020/12/31/data.txt`.

```
my-key/!{timestamp:YYYY}/!{timestamp:MM}/!{timestamp:dd}/data.txt
```

> **Note**
>
> Si vous souhaitez surveiller l'état d'exportation d'un flux, commencez par créer un flux d'état, puis configurez le flux d'exportation pour l'utiliser. Pour plus d'informations, consultez the section called “Surveillance des tâches d'exportation”.

Gérer les données d'entrée

Vous pouvez créer du code utilisé par les applications IoT pour gérer le cycle de vie des données d'entrée. L'exemple de flux de travail suivant montre comment utiliser les fonctions Lambda pour gérer ces données.

1. Un processus local reçoit des données provenant de périphériques ou de périphériques, puis les écrit dans des fichiers situés dans un répertoire sur le périphérique principal. Il s'agit des fichiers d'entrée du gestionnaire de flux.

   > **Note**
   >
   > Pour déterminer si vous devez configurer l'accès au répertoire des fichiers d'entrée, consultez leSTREAM_MANAGER_READ_ONLY_DIRS Paramètre .
   > Le processus dans lequel le gestionnaire de flux exécute hérite de toutes les autorisations de système de fichiers duidentité d'accès par défautpour le groupe. Le gestionnaire de flux doit avoir l'autorisation d'accéder aux fichiers d'entrée. Vous pouvez utiliser le plugin`chmod(1)`pour modifier l'autorisation des fichiers, si nécessaire.

2. Une fonction Lambda analyse le répertoire etajoute une tâche d'exportationvers le flux cible lorsqu'un nouveau fichier est créé. La tâche est sérialisée par JSON`S3ExportTaskDefinition`qui spécifie l'URL du fichier d'entrée, le compartiment et la clé Amazon S3 cible, ainsi que les métadonnées utilisateur facultatives.
3. Le gestionnaire de flux lit le fichier d'entrée et exporte les données vers Amazon S3 dans l'ordre des tâches ajoutées. Le compartiment cible doit déjà exister dans votreCompte AWS. Si un objet pour la clé spécifiée n'existe pas, le gestionnaire de flux crée l'objet pour vous.
4. La fonction Lambdalit des messagesà partir d'un flux d'état pour surveiller l'état de l'exportation. Une fois les tâches d'exportation terminées, la fonction Lambda peut supprimer les fichiers d'entrée correspondants. Pour plus d'informations, consultez the section called “Surveillance des tâches d'exportation”.

Surveillance des tâches d'exportation

Vous pouvez créer du code que les applications IoT utilisent pour surveiller l'état de vos exportations Amazon S3. Vos fonctions Lambda doivent créer un flux d'état, puis configurer le flux d'exportation pour écrire des mises à jour d'état dans le flux d'état. Un flux d'état unique peut recevoir des mises à jour de statut provenant de plusieurs flux exportés vers Amazon S3.

Premiers,Créer un fluxà utiliser comme flux d'état. Vous pouvez configurer les stratégies de taille et de rétention du flux afin de contrôler la durée de vie des messages d'état. Par exemple :

- Définir`Persistence`pour`Memory`si vous ne voulez pas stocker les messages d'état.
- Définir`StrategyOnFull`pour`OverwriteOldestData`afin que les nouveaux messages d'état ne soient pas perdus.

Ensuite, créez ou mettez à jour le flux d'exportation pour utiliser le flux d'état. Spécifiquement, définissez la propriété de configuration d'état du flux`S3ExportTaskExecutorConfig`configuration d'exportation. Cela indique au gestionnaire de flux d'écrire des messages d'état concernant les tâches d'exportation vers le flux d'état. Dans`StatusConfig`, spécifiez le nom du flux d'état et le niveau de verbosité. Les valeurs prises en charge suivantes vont des valeurs les moins détaillées (`ERROR`) au plus verbeux (`TRACE`). La valeur par défaut est `INFO`.

- `ERROR`
- `WARN`
- `INFO`
- `DEBUG`
- `TRACE`

L'exemple de flux de travail suivant montre comment les fonctions Lambda peuvent utiliser un flux d'état pour surveiller l'état de l'exportation.

1. Comme décrit dans le flux de travail précédent, une fonction Lambdaajoute une tâche d'exportationvers un flux configuré pour écrire des messages d'état concernant les tâches d'exportation vers un flux d'état. L'opération d'ajout renvoie un numéro de séquence qui représente l'ID de la tâche.

2. Une fonction Lambdalit des messagesde manière séquentielle à partir du flux d'état, puis filtre les messages en fonction du nom du flux et de l'ID de tâche ou en fonction d'une propriété de tâche d'exportation à partir du contexte de message. Par exemple, la fonction Lambda peut filtrer en fonction de l'URL du fichier d'entrée de la tâche d'exportation, qui est représentée par le`S3ExportTaskDefinition`objet dans le contexte du message.

   Les codes d'état suivants indiquent qu'une tâche d'exportation a atteint son état terminé :

   - `Success`. Le téléchargement s'est terminé avec succès.
   - `Failure`. Le gestionnaire de flux a rencontré une erreur, par exemple, le compartiment spécifié n'existe pas. Une fois le problème résolu, vous pouvez ajouter à nouveau la tâche d'exportation au flux.
   - `Canceled`. La tâche a été abandonnée parce que la définition du flux ou de l'exportation a été supprimée, ou le time-to-live (TTL) de la tâche a expiré.

   > **Note**
   >
   > La tâche peut également avoir le statut de`InProgress`ou`Warning`. Le gestionnaire de flux émet des avertissements lorsqu'un événement renvoie une erreur qui n'affecte pas l'exécution de la tâche. Par exemple, l'échec du nettoyage d'un chargement partiel interrompu renvoie un avertissement.

3. Une fois les tâches d'exportation terminées, la fonction Lambda peut supprimer les fichiers d'entrée correspondants.

L'exemple suivant montre comment une fonction Lambda peut lire et traiter les messages d'état.

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.
```

Référence du kit 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;
                     } 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.
}
```

Référence du kit SDK Java :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) => {
```

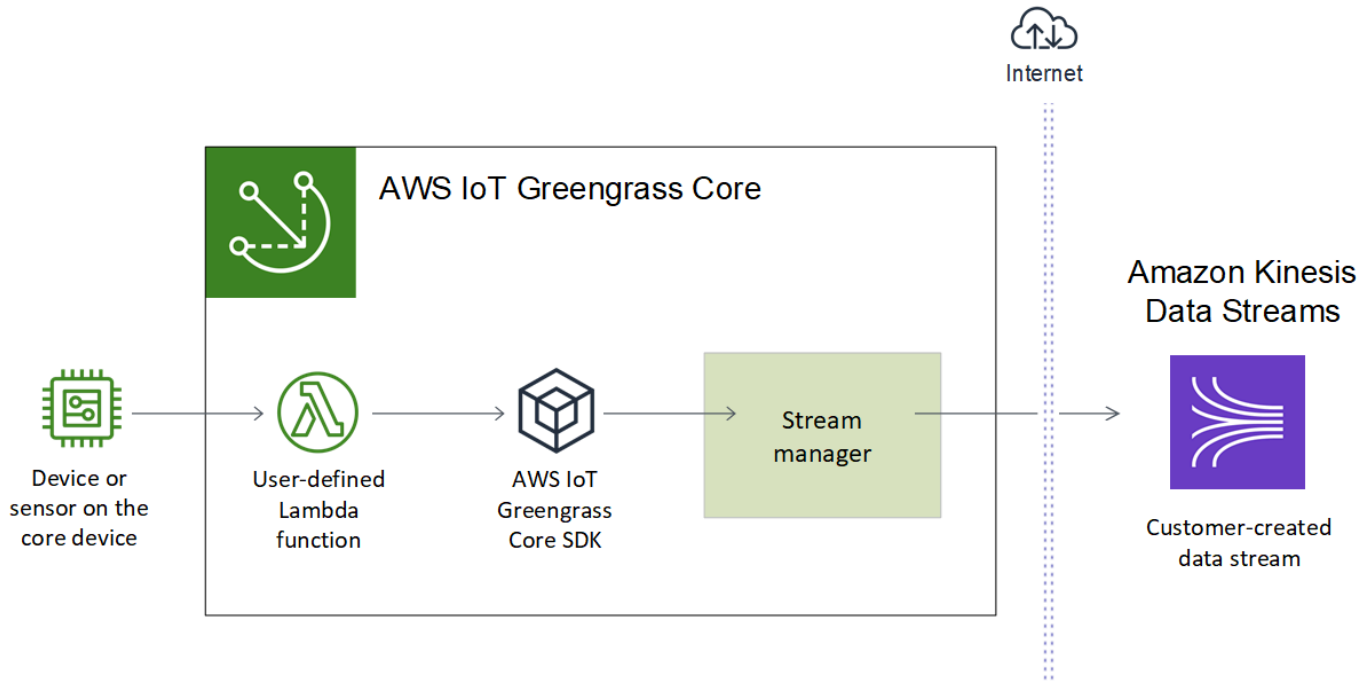
```
                    // Deserialize the status message first.
                    const statusMessage =
 util.deserializeJsonBytesToObj(message.payload, StatusMessage);
                    // Check the status of the status message. If the status is
 'Success', the file was successfully uploaded to 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.
});
```

Référence du kit SDK Node.js :readMessages|StatusMessage

Exportation de flux de données versAWS Cloud(console)

Ce didacticiel vous montre comment utiliserAWS IoTconsole pour configurer et déployer uneAWS IoT Greengrassgroupe avec le gestionnaire de flux activé. Le groupe contient une fonction Lambda définie par l'utilisateur qui écrit dans un flux du gestionnaire de flux, ce flux étant ensuite exporté automatiquement vers leAWS Cloud.

Le gestionnaire de flux rend l'ingestion, le traitement et l'exportation de flux de données volumineux plus efficaces et plus fiables. Dans ce didacticiel, vous créez une`TransferStream`Fonction Lambda qui consomme des données IoT. La fonction Lambda utilise la fonctionAWS IoT GreengrassKit SDK principal pour créer un flux dans le gestionnaire de flux, puis lire et écrire dans ce flux. Le gestionnaire de flux exporte ensuite le flux vers Kinesis Data Streams. Le schéma suivant illustre ce flux de travail.



L'objectif de ce didacticiel est de montrer comment les fonctions Lambda définies par l'utilisateur utilisent le`StreamManagerClient`l'objet dans l'AWS IoT GreengrassKit SDK principal pour interagir avec le gestionnaire de flux. Par souci de simplicité, la fonction Lambda Python que vous créez pour ce didacticiel génère des données d'appareil simulées.

Prérequis

Pour suivre ce didacticiel, vous devez disposer des éléments suivants :

- Un groupe Greengrass et un service principal Greengrass (version 1.10 ou ultérieure). Pour plus d'informations sur la création d'un groupe et d'un service principal Greengrass, consultezCommencer avec AWS IoT Greengrass. Le didacticiel Mise en route comprend également les étapes d'installation du logiciel AWS IoT Greengrass Core.

  > **Note**
  >
  > Gestionnaire de flux n'est pas pris en charge sur OpenWrt Distributions.

- L'environnement d'exécution Java 8 (JDK 8) installé sur l'appareil principal (noyau).
  - Pour les distributions basées sur Debian (y compris Raspbian) ou Ubuntu, exécutez la commande suivante :

    ```
    sudo apt install openjdk-8-jdk
    ```

  - Pour les distributions basées sur Red Hat (y compris Amazon Linux), exécutez la commande suivante :

    ```
    sudo yum install java-1.8.0-openjdk
    ```

    Pour de plus amples informations, veuillez consulter How to download and install prebuilt OpenJDK packages sur le site web OpenJDK.

- AWS IoT GreengrassKit SDK principal pour Python v1.5.0 ou version ultérieure. Pour utiliser`StreamManagerClient`dans leAWS IoT GreengrassSDK principal pour Python, vous devez :
  - Installer Python 3.7 ou version ultérieure sur l'appareil principal (noyau).
  - Incluez le SDK et ses dépendances dans votre package de déploiement de fonctions Lambda. Des instructions sont fournies dans ce didacticiel.

  > **Tip**
  >
  > Vous pouvez utiliser `StreamManagerClient` avec Java ou NodeJS. Pour obtenir un exemple de code, veuillez consulterAWS IoT GreengrassKit SDK de développement logiciel pour JavaetAWS IoT GreengrassKit SDK de développement logiciel pour Node.jssur GitHub.

- Un flux de destination nommé**`MyKinesisStream`**créé dans Amazon Kinesis Data Streams dansRégion AWScomme votre groupe Greengrass. Pour de plus amples informations, veuillez consulterCréer un fluxdans leGuide du développeur Amazon Kinesis.

  > **Note**
  >
  > Dans ce didacticiel, le gestionnaire de flux exporte les données vers Kinesis Data Streams, ce qui entraîne des frais sur votreCompte AWS. Pour de plus amples informations sur la tarification, consultezTarification de Kinesis Data Streams.
  > Pour éviter des frais, vous pouvez exécuter ce didacticiel sans créer de flux de données Kinesis. Dans ce cas, vous consultez les journaux pour voir si le gestionnaire de flux a tenté d'exporter le flux vers Kinesis Data Streams.

- Une stratégie IAM ajoutée àthe section called “Rôle de groupe Greengrass”qui permet au`kinesis:PutRecords`action sur le flux de données cible, comme illustré dans l'exemple suivant :

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kinesis:PutRecords"
            ],
            "Resource": [
                "arn:aws:kinesis:region:account-id:stream/MyKinesisStream"
            ]
        }
    ]
}
```

Le didacticiel contient les étapes détaillées suivantes :

1. Créer un package de déploiement de fonction Lambda
2. Création d'une fonction Lambda
3. Ajouter une fonction au groupe
4. Activer le gestionnaire de flux

5. Configurer la journalisation locale
6. Déployer le groupe
7. Tester l'application

Le didacticiel devrait prendre environ 20 minutes.

Étape 1 : Créer un package de déploiement de fonction Lambda

Au cours de cette étape, vous créez un package de déploiement de fonction Lambda qui contient le code de fonction Python et ses dépendances. Vous téléchargez ce package ultérieurement lorsque vous créez leFonction Lambda dansAWS Lambda. La fonction Lambda utilise la fonctionAWS IoT GreengrassKit SDK principal pour créer des flux locaux et interagir avec ces derniers.

> **Note**
>
> Vos fonctions Lambda définies par l'utilisateur doivent utiliser leAWS IoT GreengrassKit SDK Corepour interagir avec le gestionnaire de flux. Pour de plus amples informations sur les conditions requises pour le gestionnaire de flux Greengrass, veuillez consulter les conditions requises pour le gestionnaire de flux Greengrass.

1. Télécharger leAWS IoT GreengrassKit SDK de développement logiciel pour Pythonv1.5.0 ou version ultérieure.
2. Décompressez le package téléchargé pour obtenir le kit SDK. Le kit SDK est représenté par le dossier `greengrasssdk`.
3. Installez les dépendances de package à inclure avec le kit SDK dans votre package de déploiement de fonction Lambda.

    1. Accédez au répertoire SDK qui contient le fichier `requirements.txt`. Ce fichier répertorie les dépendances.
    2. Installez les dépendances du kit SDK. Par exemple, exécutez la commande `pip` suivante pour les installer dans le répertoire en cours :

        ```
        pip install --target . -r requirements.txt
        ```

4. Enregistrez la fonction de code Python suivante dans un fichier local nommé `transfer_stream.py`.

> **Tip**
>
> Pour obtenir un exemple de code qui utilise Java et NodeJS, veuillez consulterAWS IoT GreengrassKit SDK de développement logiciel pour JavaetAWS IoT GreengrassKit SDK de développement logiciel pour Node.jssur GitHub.

```
import asyncio
import logging
import random
import time

from greengrasssdk.stream_manager import (
    ExportDefinition,
    KinesisConfig,
    MessageStreamDefinition,
    ReadMessagesOptions,
    ResourceNotFoundException,
    StrategyOnFull,
    StreamManagerClient,
)


# This example creates a local stream named "SomeStream".
# It starts writing data into that stream and then stream manager automatically
 exports
# the data to a customer-created Kinesis data stream named "MyKinesisStream".
# This example runs forever until the program is stopped.

# The size of the local stream on disk will not exceed the default (which is 256
 MB).
# Any data appended after the stream reaches the size limit continues to be
 appended, and
# stream manager deletes the oldest data until the total stream size is back under
 256 MB.
# The Kinesis data stream in the cloud has no such bound, so all the data from this
 script is
# uploaded to Kinesis and you will be charged for that usage.


def main(logger):
```

```
    try:
        stream_name = "SomeStream"
        kinesis_stream_name = "MyKinesisStream"

        # Create a client for the StreamManager
        client = StreamManagerClient()

        # Try deleting the stream (if it exists) so that we have a fresh start
        try:
            client.delete_message_stream(stream_name=stream_name)
        except ResourceNotFoundException:
            pass

        exports = ExportDefinition(
            kinesis=[KinesisConfig(identifier="KinesisExport" + stream_name,
kinesis_stream_name=kinesis_stream_name)]
        )
        client.create_message_stream(
            MessageStreamDefinition(
                name=stream_name,
strategy_on_full=StrategyOnFull.OverwriteOldestData, export_definition=exports
            )
        )

        # Append two messages and print their sequence numbers
        logger.info(
            "Successfully appended message to stream with sequence number %d",
            client.append_message(stream_name, "ABCDEFGHIJKLMNO".encode("utf-8")),
        )
        logger.info(
            "Successfully appended message to stream with sequence number %d",
            client.append_message(stream_name, "PQRSTUVWXYZ".encode("utf-8")),
        )

        # Try reading the two messages we just appended and print them out
        logger.info(
            "Successfully read 2 messages: %s",
            client.read_messages(stream_name,
ReadMessagesOptions(min_message_count=2, read_timeout_millis=1000)),
        )

        logger.info("Now going to start writing random integers between 0 and 1000
to the stream")
```

```
        # Now start putting in random data between 0 and 1000 to emulate device
 sensor input
        while True:
            logger.debug("Appending new random integer to stream")
            client.append_message(stream_name, random.randint(0,
 1000).to_bytes(length=4, signed=True, byteorder="big"))
            time.sleep(1)

    except asyncio.TimeoutError:
        logger.exception("Timed out while executing")
    except Exception:
        logger.exception("Exception while running")


def function_handler(event, context):
    return


logging.basicConfig(level=logging.INFO)
# Start up this sample code
main(logger=logging.getLogger())
```

5. Compressez les éléments suivants dans un fichier nommé `transfer_stream_python.zip`. Il s'agit du package de déploiement de votre fonction Lambda.

   - transfer_stream.py. Logique d'application.
   - greengrasssdk. Bibliothèque requise pour toutes les fonctions Lambda Python Greengrass qui publient des messages MQTT.

     Opérations du gestionnaire de fluxsont disponibles dans la version 1.5.0 ou ultérieure duAWS IoT GreengrassKit SDK de développement logiciel pour Python.
   - Les dépendances que vous avez installées pourAWS IoT GreengrassKit SDK principal pour Python (par exemple,`cbor2`annuaires).

   Lorsque vous créez le fichier `zip`, incluez uniquement ces éléments, et non le dossier conteneur.

Étape 2 : Création d'une fonction Lambda

Au cours de cette étape, vous utilisez leAWS Lambdapour créer une fonction Lambda et pour la configurer afin qu'elle utilise votre package de déploiement. Vous publiez ensuite une version de fonction et créez un alias.

1. Créez d'abord la fonction Lambda.

   a. Dans AWS Management Console, choisissez Services et ouvrez la console AWS Lambda.

   b. ChoisissezCréation de fonctionpuisCréer à partir de zéro.

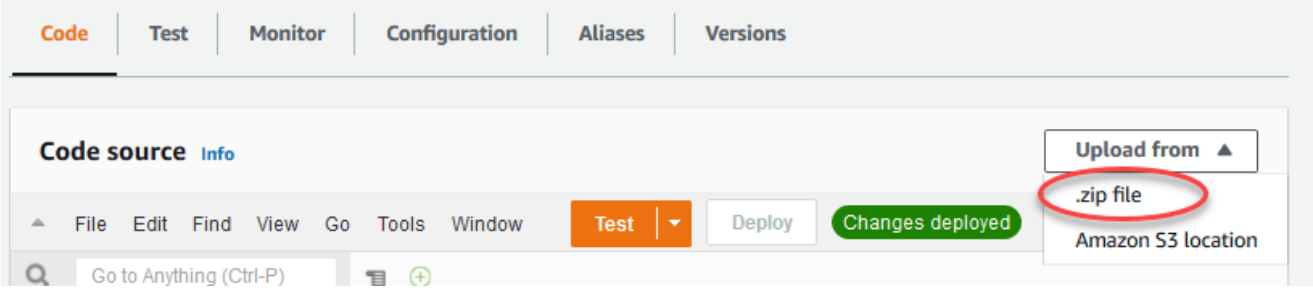
   c. Dans la section Informations de base, spécifiez les valeurs suivantes :

      - Sous Nom de la fonction, saisissez **`TransferStream`**.
      - Pour Runtime, sélectionnez Python 3.7.
      - PourAutorisations, conservez le paramètre par défaut. Cela crée un rôle d'exécution qui accorde des autorisations Lambda de base. Ce rôle n'est pas utilisé parAWS IoT Greengrass.

   d. Dans le bas de la page, choisissez Create function.

2. Enregistrez ensuite le gestionnaire et téléchargez le package de déploiement de votre fonction Lambda.

   a. Dans la pageCodeonglet, sousCode source, choisissezChargement à partir de. Dans le menu déroulant, choisissezfichier .zip.



   b. ChoisissezCharger, puis choisissez votre`transfer_stream_python.zip`package de déploiement. Ensuite, choisissez Enregistrer.

   c. Dans la pageCodepour la fonction, sousParamètres Runtime, choisissezModifier, puis entrez les valeurs suivantes.

      - Pour Runtime, sélectionnez Python 3.7.
      - Pour Handler (Gestionnaire), entrez **`transfer_stream.function_handler`**.

d. Choisissez Save (Enregistrer).

> **Note**
>
> LeTestsur leAWS Lambdane fonctionne pas avec cette fonction. LeAWS IoT GreengrassLe SDK principal ne contient pas de modules nécessaires pour exécuter vos fonctions Greengrass Lambda indépendamment dans leAWS Lambdaconsole Ces modules (par exemple,`greengrass_common`) sont fournis aux fonctions après leur déploiement dans votre cœur Greengrass.

3. Publiez maintenant la première version de votre fonction Lambda et créez unealias pour la version.

> **Note**
>
> Les groupes Greengrass peuvent référencer une fonction Lambda par alias (recommandé) ou par version. L'utilisation d'un alias facilite la gestion des mises à jour de code, car vous n'avez pas besoin de changer votre table d'abonnement ou votre définition de groupe lorsque le code de fonction est mis à jour. Au lieu de cela, il vous suffit de pointer l'alias vers la nouvelle version de fonction.

a. Dans le menu Actions, sélectionnez Publier une nouvelle version.

b. Dans Description de la version, saisissez **`First version`**, puis choisissez Publish.

c. Dans la pageTransferStream : 1page de configuration, depuis la pageActions, choisissezCréer un alias.

d. Sur la page Create a new alias, utilisez les valeurs suivantes :

- Pour Name (Nom), saisissez **`GG_TransferStream`**.
- Pour Version, choisissez 1.

> **Note**
>
> AWS IoT Greengrassne prend pas en charge les alias Lambda pour$LATESTversions.

e. Sélectionnez Create (Créer).

Vous pouvez désormais ajouter la fonction Lambda à votre groupe Greengrass.

Étape 3 : Ajouter une fonction Lambda au groupe Greengrass

Au cours de cette étape, vous ajoutez la fonction Lambda au groupe, puis configurez son cycle de vie et ses variables d'environnement. Pour plus d'informations, consultez the section called “Contrôle de l'exécution de la fonction Greengrass Lambda”.

1. DansAWS IoTVolet de navigation de la console sousGérer, développezAppareils Greengrass, puisGroupes (V1).
2. Choisissez le groupe cible.
3. Sur la page de configuration de groupe, choisissezFonctions LambdaOnglet.
4. UNDERFonctions My Lambda, choisissezAddition.
5. Dans la pageAjouter une fonction Lambda, choisissez l'Fonction Lambdapour votre fonction Lambda.
6. PourVersion Lambda, choisissezAlias : GG_TransferStream.

   Maintenant, configurez les propriétés qui déterminent le comportement de la fonction Lambda dans le groupe Greengrass.
7. DansConfiguration de fonction Lambda, effectuez les modifications suivantes :

   - Définissez Limite de mémoire sur 32 Mo.
   - PourPinned, choisissezVrai.

   > **Note**
   >
   > UNlongue durée(ouépinglé) La fonction Lambda démarre automatiquement aprèsAWS IoT Greengrassdémarre et continue à fonctionner dans son propre conteneur. Cela contraste avec unA laFonction Lambda, qui démarre quand elle est appelée et qui s'arrête lorsqu'il n'y a plus de tâches à exécuter. Pour plus d'informations, consultez the section called “Configuration du cycle de vie”.
8. ChoisissezAjouter une fonction Lambda.

Étape 4 : Activer le gestionnaire de flux

Dans cette étape, vous vous assurez que le gestionnaire de flux est activé.

1. Sur la page de configuration de groupe, choisissezFonctions LambdaOnglet.
2. UNDERFonctions Lambda système, sélectionnezGestionnaire de flux, puis l'état. Si l'état est désactivé, choisissez Modifier. Ensuite, choisissez Activer et Enregistrer. Vous pouvez utiliser les paramètres par défaut pour ce didacticiel. Pour plus d'informations, consultez the section called “Configuration du gestionnaire de flux”.

> **Note**
>
> Lorsque vous utilisez la console pour activer le gestionnaire de flux et déployer le groupe, la taille de la mémoire du gestionnaire de flux est définie par défaut sur 4 194 304 Ko (4 Go). Nous vous recommandons de définir la taille de la mémoire sur au moins 128 000 Ko.

Étape 5 : Configurer la journalisation locale

Au cours de cette étape, vous configurezAWS IoT Greengrasscomposants système, fonctions Lambda définies par l'utilisateur et connecteurs du groupe pour qu'ils écrivent des journaux dans le système de fichiers de l'appareil principal (noyau). Vous pouvez utiliser les journaux pour résoudre les problèmes que vous rencontrez. Pour plus d'informations, consultez the section called “Surveillance avec les journaux AWS IoT Greengrass”.

1. Sous Configuration des journaux locaux, vérifiez si la journalisation locale est configurée.
2. Si les journaux ne sont pas configurés pour les composants système Greengrass ou les fonctions Lambda définies par l'utilisateur, choisissezModifier.
3. ChoisissezNiveau de journalisation des fonctions Lambda utilisateuretNiveau de journalisation du système Greengrass.
4. Conservez les valeurs par défaut de niveau de journalisation et de limite d'espace disque, puis choisissez Enregistrer.

Étape 6 : Déploiement du groupe Greengrass

Déployer le groupe sur l'appareil principal (noyau)

1. Vérifiez que lesAWS IoT Greengrasscore est en cours d'exécution. Dans la fenêtre de terminal de votre Raspberry Pi, exécutez les commandes suivantes, si nécessaire.

   a. Pour vérifier si le démon est en cours d'exécution :

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

   Si la sortie contient une entrée `root` pour `/greengrass/ggc/packages/`*`ggc-version`*`/bin/daemon`, le démon est en cours d'exécution.

   > **Note**
   >
   > La version du chemin d'accès dépend de la version du logiciel AWS IoT Greengrass Core installée sur votre appareil principal.

   b. Pour démarrer le démon :

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

2. Sur la page de configuration de groupe, choisissezDéploiement.

3. a. DansFonctions Lambda, sous l'ongletFonctions Lambda systèmesection, sélectionnezDétecteur IPet choisissezModifier.

   b. DansModifier les paramètres du détecteur IPboîte de dialogue, sélectionnezDétecter et remplacer automatiquement les points de terminaison des courtiers MQTT.

   c. Choisissez Save (Enregistrer).

   Les appareils peuvent ainsi acquérir automatiquement des informations de connectivité pour le noyau, telles que l'adresse IP, le DNS et le numéro de port. La détection automatique est recommandée, mais AWS IoT Greengrass prend également en charge les points de terminaison spécifiés manuellement. Vous êtes uniquement invité à indiquer la méthode de découverte lors du déploiement initial du groupe.

> **Note**
>
> Si vous y êtes invité, autorisez la création deRôle de service Greengrasset associez-le à votreCompte AWSdans l'actuelRégion AWS. Ce rôle permetAWS IoT Greengrasspour accéder à vos ressources dansAWSServices .

La page Déploiements indique l'horodatage, l'ID de version et l'état du déploiement. Une fois terminé, le statut affiché pour le déploiement doit afficher l'étatTerminé.

Pour bénéficier d'une aide à la résolution des problèmes, consultez Résolution des problèmes.

Étape 7 : Tester l'application

LeTransferStreamLa fonction Lambda génère des données d'appareil simulées. Elle écrit des données dans un flux que le gestionnaire de flux exporte vers le flux de données Kinesis cible.

1. Dans la console Amazon Kinesis, sousKinesis, choisissezMyKinesisStream.

   > **Note**
   >
   > Si vous avez exécuté le didacticiel sans flux de données Kinesis cible, recherchez dans le fichier journal le gestionnaire de flux (`GGStreamManager`). S'il contient `export stream MyKinesisStream doesn't exist` dans un message d'erreur, le test est réussi. Cette erreur signifie que le service a essayé d'effectuer une exportation vers le flux mais que le flux n'existe pas.

2. Dans la pageMyKinesisStreampage, choisissezSurveillance. Si le test réussit, vous devriez voir des données dans les graphiques PutRecords . Selon votre connexion, l'affichage des données peut prendre une minute.

   > **Important**
   >
   > Lorsque vous avez terminé le test, supprimez le flux de données Kinesis pour éviter d'entraîner des frais supplémentaires.

Vous pouvez aussi exécuter la commande suivante pour arrêter le démon Greengrass. Cela empêche le noyau d'envoyer des messages jusqu'à ce que vous soyez prêt à continuer le test.

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

3. Supprimez lesTransferStreamFonction Lambda depuis le noyau.

   a. DansAWS IoTVolet de navigation de la console sousGérer, développezAppareils Greengrass, puisGroupes (V1).

   b. UNDERGroupes Greengrass, choisissez votre groupe.

   c. Dans la pageLambda, choisissez les points de suspension (...) pour leTransferStreamfonction, puis choisissezFonction Supprimer.

   d. Sous Actions, choisissez Deploy (Déployer).

Pour afficher les informations de journalisation ou résoudre les problèmes liés aux flux, recherchez les fonctions `GGStreamManager` et `TransferStream` dans les journaux. Vous devez disposer d'autorisations `root` pour pouvoir lire les journaux AWS IoT Greengrass sur le système de fichiers.

- `TransferStream` écrit les entrées de journal dans *`greengrass-root`*`/ggc/var/log/user/`*`region`*`/`*`account-id`*`/TransferStream.log`.
- `GGStreamManager` écrit les entrées de journal dans *`greengrass-root`*`/ggc/var/log/system/GGStreamManager.log`.

Si vous avez besoin de plus d'informations de dépannage, vous pouvezdéfinir le niveau de journalisationpourJournaux Lambda utilisateurpourJournaux de débogagepuis déployez à nouveau le groupe.

Consulter aussi

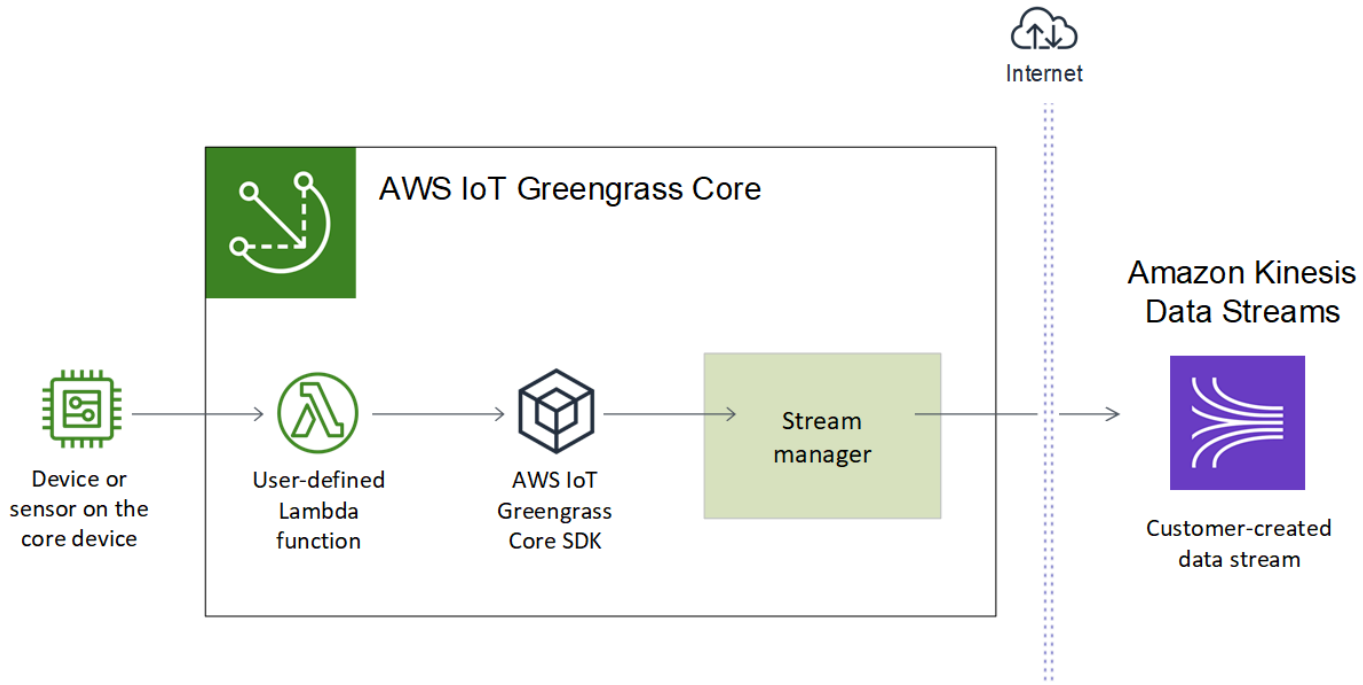
- Gestion des flux de données
- the section called “Configuration du gestionnaire de flux”
- the section called “Utiliser StreamManagerClient pour travailler avec des flux”
- the section called “Configurations d'exportation pour prises en chargeAWS Clouddestinations”

- the section called “Exportation de flux de données (interface de ligne de commande)”

Exportation de flux de données vers leAWS Cloud(CLI)

Ce didacticiel vous montre comment utiliser leAWS CLIpour configurer et déployer unAWS IoT Greengrassgroupe avec le gestionnaire de flux activé. Le groupe contient une fonction Lambda définie par l'utilisateur qui écrit dans un flux du gestionnaire de flux, ce flux étant ensuite exporté automatiquement vers le groupe deAWS Cloud.

Le gestionnaire de flux facilite et fiabilise l'ingestion, le traitement et l'exportation de flux de données volumineux plus efficaces et plus fiables. Dans ce didacticiel, vous créez une`TransferStream`Une fonction Lambda qui consomme des données IoT. La fonction Lambda utilise la fonctionAWS IoT GreengrassInstaller le kit principal pour créer un flux dans le gestionnaire de flux, puis lire et écrire dans ce flux. Le gestionnaire de flux exporte ensuite le flux vers Kinesis Data Streams. Le schéma suivant illustre ce flux de travail.



L'objectif de ce didacticiel est de montrer comment les fonctions Lambda définies par l'utilisateur utilisent le`StreamManagerClient`dans leAWS IoT GreengrassKit SDK pour interagir avec le gestionnaire de flux. Par souci de simplicité, la fonction Python Lambda dans le cadre de ce didacticiel génère des données d'appareil simulées pour ce didacticiel.

Lorsque vous utilisez leAWS IoT Greengrass, qui inclut les commandes Greengrass dans leAWS CLI, pour créer un groupe, le gestionnaire de flux est désactivé par défaut. Pour activer

Stream Manager sur votre cœur, vouscréer une version de définition de fonctionqui inclut le systèmeGGStreamManagerFonction Lambda et une version de groupe qui fait référence à la nouvelle version de définition de fonction. Vous déployez ensuite le groupe.

Prérequis

Pour suivre ce didacticiel, vous devez disposer des éléments suivants :

- Un groupe Greengrass et un service principal Greengrass (version). Pour plus d'informations sur la création d'un groupe et un service principal Greengrass, consultez pour plus d'informationsCommencer avec AWS IoT Greengrass. Le didacticiel Mise en route comprend également les étapes d'installation du logiciel AWS IoT Greengrass Core.

  > **Note**
  >
  > Gestionnaire de flux n'est pas pris en charge sur OpenWrt Distributions.

- L'environnement d'exécution Java 8 (JDK 8) installé sur l'appareil principal (noyau).
  - Pour les distributions basées sur Debian (y compris Raspbian) ou Ubuntu, exécutez la commande suivante :

    ```
    sudo apt install openjdk-8-jdk
    ```

  - Pour les distributions basées sur Red Hat (y compris Amazon Linux), exécutez la commande suivante :

    ```
    sudo yum install java-1.8.0-openjdk
    ```

    Pour de plus amples informations, veuillez consulter How to download and install prebuilt OpenJDK packages sur le site web OpenJDK.

- AWS IoT GreengrassKit SDK pour Python v1.5.0 ou version ultérieure. Pour utiliserStreamManagerClientdans leAWS IoT GreengrassSDK principal pour Python, vous devez :
  - Installer Python Installer Python sur Installer Python ou
  - Incluez le SDK et ses dépendances dans votre package de déploiement de fonctions Lambda. Des instructions sont fournies dans ce didacticiel.

> **Tip**
>
> Vous pouvez utiliser `StreamManagerClient` avec Java ou NodeJS. Pour obtenir un exemple de code, consultez leAWS IoT GreengrassKit SDK for JavaetAWS IoT GreengrassKit SDK pour Node.jssur GitHub.

- Un flux de destination nommé**`MyKinesisStream`**créé dans Amazon Amazon Kinesis Data Streams dans le mêmeRégion AWSen tant que groupe Greengrass. Pour de plus amples informations, veuillez consulterCréer un fluxdans leAmazon Kinesis Developer Guide.

> **Note**
>
> Dans ce didacticiel, le gestionnaire de flux exporte les données vers Kinesis Data Streams, ce qui entraîne des frais sur votreCompte AWS. Pour plus d'informations sur la tarification, consultez leTarification Kinesis Data Streams.
> Pour éviter des frais, vous pouvez exécuter ce didacticiel sans créer de flux de données Kinesis. Dans ce cas, vous consultez les journaux pour voir si le gestionnaire de flux a tenté d'exporter le flux vers Kinesis Data Streams.

- Une stratégie IAM ajoutée à lathe section called “Rôle de groupe Greengrass”qui permet au`kinesis:PutRecords`action sur le flux de données cible, comme illustré dans l'exemple suivant :

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kinesis:PutRecords"
            ],
            "Resource": [
                "arn:aws:kinesis:region:account-id:stream/MyKinesisStream"
            ]
        }
    ]
}
```

- L'AWS CLI installée et configurée sur votre ordinateur. Pour de plus amples informations, veuillez consulterInstallation deAWS Command Line InterfaceetConfiguration deAWS CLIdans leAWS Command Line InterfaceGuide de l'utilisateur.

  Les exemples de commandes de ce didacticiel sont écrits pour Linux et d'autres systèmes Unix. Si vous utilisez Windows, consultezSpécification des valeurs des paramètres pour l'AWSInterface de ligne de commandepour plus d'informations sur les différences de syntaxe.

  Si la commande contient une chaîne JSON, le didacticiel fournit un exemple qui possède le fichier JSON sur une seule ligne. Sur certains systèmes, il peut être plus efficace de modifier et exécuter des commandes à l'aide de ce format.

Le didacticiel contient les étapes détaillées suivantes :

1. Créer un package de déploiement de fonction Lambda
2. Création d'une fonction Lambda
3. Créer une version et une définition de fonction
4. Créer une définition et une version de l'enregistreur
5. Obtenir l'ARN de votre version de définition du noyau
6. Créer une version de groupe
7. Créer un déploiement
8. Tester l'application

Le didacticiel devrait prendre environ 30 minutes.

Étape 1 : Créer un package de déploiement de fonction Lambda

Dans cette étape, vous créez un package de déploiement de fonction Lambda qui contient le code de fonction Python et les dépendances. Vous téléchargez ce package ultérieurement lorsque vous créez lefonction Lambda dansAWS Lambda. La fonction Lambda utilise la fonctionAWS IoT GreengrassInstaller le kit principal pour créer des flux locaux et interagir avec ces derniers.

> **Note**
>
> Vos fonctions Lambda définies par l'utilisateur doivent utiliser leAWS IoT GreengrassKit SDK Corepour interagir avec le gestionnaire de flux. Pour de plus amples informations sur les conditions requises pour le gestionnaire de flux Greengrass, veuillez consulter les conditions requises pour le gestionnaire de flux Greengrass.

1. Télécharger leAWS IoT GreengrassKit SDK pour Pythonv1.5.0 ou version ultérieure.
2. Décompressez le package téléchargé pour obtenir le kit SDK. Le kit SDK est représenté par le dossier `greengrasssdk`.
3. Installer Installer Installer les dépendances de package Installer les dépendances de package à inclure dans le kit de déploiement de votre fonction Lambda.

   1. Accédez au répertoire SDK qui contient le fichier `requirements.txt`. Ce fichier répertorie les dépendances.
   2. Installez les dépendances du kit SDK. Par exemple, exécutez la commande `pip` suivante pour les installer dans le répertoire en cours :

      ```
      pip install --target . -r requirements.txt
      ```
4. Enregistrez la fonction de code Python suivante dans un fichier local nommé `transfer_stream.py`.

   > **Tip**
   >
   > Pour obtenir un exemple de code qui utilise Java et NodeJS, consultez leAWS IoT GreengrassKit SDK for JavaetAWS IoT GreengrassKit SDK pour Node.jssur GitHub.

   ```
   import asyncio
   import logging
   import random
   import time

   from greengrasssdk.stream_manager import (
       ExportDefinition,
       KinesisConfig,
   ```

```
    MessageStreamDefinition,
    ReadMessagesOptions,
    ResourceNotFoundException,
    StrategyOnFull,
    StreamManagerClient,
)


# This example creates a local stream named "SomeStream".
# It starts writing data into that stream and then stream manager automatically
 exports
# the data to a customer-created Kinesis data stream named "MyKinesisStream".
# This example runs forever until the program is stopped.

# The size of the local stream on disk will not exceed the default (which is 256
 MB).
# Any data appended after the stream reaches the size limit continues to be
 appended, and
# stream manager deletes the oldest data until the total stream size is back under
 256 MB.
# The Kinesis data stream in the cloud has no such bound, so all the data from this
 script is
# uploaded to Kinesis and you will be charged for that usage.


def main(logger):
    try:
        stream_name = "SomeStream"
        kinesis_stream_name = "MyKinesisStream"

        # Create a client for the StreamManager
        client = StreamManagerClient()

        # Try deleting the stream (if it exists) so that we have a fresh start
        try:
            client.delete_message_stream(stream_name=stream_name)
        except ResourceNotFoundException:
            pass

        exports = ExportDefinition(
            kinesis=[KinesisConfig(identifier="KinesisExport" + stream_name,
 kinesis_stream_name=kinesis_stream_name)]
        )
        client.create_message_stream(
```

```
            MessageStreamDefinition(
                name=stream_name,
 strategy_on_full=StrategyOnFull.OverwriteOldestData, export_definition=exports
            )
        )

        # Append two messages and print their sequence numbers
        logger.info(
            "Successfully appended message to stream with sequence number %d",
            client.append_message(stream_name, "ABCDEFGHIJKLMNO".encode("utf-8")),
        )
        logger.info(
            "Successfully appended message to stream with sequence number %d",
            client.append_message(stream_name, "PQRSTUVWXYZ".encode("utf-8")),
        )

        # Try reading the two messages we just appended and print them out
        logger.info(
            "Successfully read 2 messages: %s",
            client.read_messages(stream_name,
 ReadMessagesOptions(min_message_count=2, read_timeout_millis=1000)),
        )

        logger.info("Now going to start writing random integers between 0 and 1000
 to the stream")
        # Now start putting in random data between 0 and 1000 to emulate device
 sensor input
        while True:
            logger.debug("Appending new random integer to stream")
            client.append_message(stream_name, random.randint(0,
 1000).to_bytes(length=4, signed=True, byteorder="big"))
            time.sleep(1)

    except asyncio.TimeoutError:
        logger.exception("Timed out while executing")
    except Exception:
        logger.exception("Exception while running")


def function_handler(event, context):
    return


logging.basicConfig(level=logging.INFO)
```

```
# Start up this sample code
main(logger=logging.getLogger())
```

5. Compressez les éléments suivants dans un fichier nommé `transfer_stream_python.zip`. Il s'agit du package de déploiement de votre fonction Lambda.

   - transfer_stream.py. Logique d'application.
   - greengrasssdk. Installer la bibliothèque requise pour toutes les fonctions Python Greengrass Lambda qui publient des messages MQTT.

     Opérations du gestionnaire de fluxsont disponibles dans la version 1.5.0 ou ultérieure duAWS IoT GreengrassKit SDK pour Python.
   - Les dépendances que vous avez installées pour leAWS IoT GreengrassKit SDK pour Python (par exemple, le`cbor2`annuaires).

   Lorsque vous créez le fichier `zip`, incluez uniquement ces éléments, et non le dossier conteneur.

Étape 2 : Création d'une fonction Lambda

1. Créez un rôle IAM afin de pouvoir transmettre l'ARN du rôle lorsque vous créez un rôle IAM pour vous permettre de transmettre l'ARN du rôle

   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 Greengrassn'utilise pas ce rôle car les autorisations pour vos fonctions Greengrass Lambda sont spécifiées dans le rôle de groupe Greengrass n'utilise pas ce rôle de groupe. Dans le cadre de ce didacticiel, vous créez un rôle vide.

2. Copiez la `Arn` à partir de la sortie.
3. Utilisez l'API AWS Lambda pour créer la fonction `TransferStream`. La commande suivante suppose que le fichier ZIP se trouve dans le répertoire actuel.

   - Remplacez *`role-arn`* par l'`Arn` que vous avez copié.

```
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. Publiez une version de la fonction.

```
aws lambda publish-version --function-name TransferStream --description 'First
 version'
```

5. Créez un alias pour la version publiée.

   Les groupes Greengrass peuvent référencer une fonction Lambda par alias (recommandé) ou par version. L'utilisation d'un alias facilite la gestion des mises à jour de code, car vous n'avez pas besoin de changer votre table d'abonnement ou la définition de groupe lorsque le code de fonction est mis à jour. À la place, il vous suffit de pointer l'alias vers la nouvelle version de fonction.

```
aws lambda create-alias --function-name TransferStream --name GG_TransferStream --
function-version 1
```

> **Note**
>
> AWS IoT Greengrassne prend pas en charge les alias Lambda pour$LATESTversions.

6. Copiez la `AliasArn` à partir de la sortie. Vous utilisez cette valeur lorsque vous configurez la fonction pour AWS IoT Greengrass.

Maintenant, vous êtes prêt à configurer la fonction pour AWS IoT Greengrass.

Étape 3 : Créer une version et une définition de fonction

Cette étape crée une version de définition de fonction qui fait référence au système.`GGStreamManager`Fonction Lambda et fonction définie par l'utilisateur`TransferStream`fonction Lambda. Pour activer le gestionnaire de flux lorsque vous utilisez leAWS IoT Greengrass, la version de votre définition de fonction doit inclure le`GGStreamManager`.

1. Créez une définition de fonction avec une version initiale qui contient les fonctions système et les fonctions Lambda définies par l'utilisateur.

   La version de définition suivante active le gestionnaire de flux avec leréglages des paramètres. Pour configurer des paramètres personnalisés, vous devez définir des variables d'environnement pour les paramètres du gestionnaire de flux correspondants. Pour obtenir un exemple, veuillez consulter .the section called “Activer, désactiver ou configurer le gestionnaire de flux”.AWS IoT Greengrassutilise les paramètres par défaut pour tous les paramètres qui sont omis.`MemorySize`doit être au moins`128000`.`Pinned`doit être défini sur`true`.

   > **Note**
   >
   > UNlongue durée(ouépinglé) La fonction Lambda démarre automatiquement aprèsAWS IoT Greengrassdémarre et continue à fonctionner dans son propre conteneur. Cela contraste avec unsur demandeFonction Lambda, qui démarre quand elle est appelée et

qui s'arrête lorsqu'il n'y a plus de tâches à exécuter. Pour plus d'informations, consultez the section called “Configuration du cycle de vie”.

- Remplacez*arbitrary-function-id*avec un nom pour la fonction, tel que**stream-manager**.
- Remplacez*alias ARN*avec le`AliasArn`que vous avez copié lorsque vous avez créé l'alias de l'`TransferStream`fonction Lambda.

JSON expanded

```
aws greengrass create-function-definition --name MyGreengrassFunctions --
initial-version '{
    "Functions": [
        {
            "Id": "arbitrary-function-id",
            "FunctionArn": "arn:aws:lambda:::function:GGStreamManager:1",
            "FunctionConfiguration": {
                "MemorySize": 128000,
                "Pinned": true,
                "Timeout": 3
            }
        },
        {
            "Id": "TransferStreamFunction",
            "FunctionArn": "alias-arn",
            "FunctionConfiguration": {
                "Executable": "transfer_stream.function_handler",
                "MemorySize": 16000,
                "Pinned": true,
                "Timeout": 5
            }
        }
    ]
}'
```

JSON single

```
aws greengrass create-function-definition \
--name MyGreengrassFunctions \
--initial-version '{"Functions": [{"Id": "arbitrary-function-
id","FunctionArn": "arn:aws:lambda:::function:GGStreamManager:1",
 "FunctionConfiguration": {"Environment": {"Variables":
{"STREAM_MANAGER_STORE_ROOT_DIR": "/data","STREAM_MANAGER_SERVER_PORT":
 "1234","STREAM_MANAGER_EXPORTER_MAX_BANDWIDTH": "20000"}},"MemorySize":
 128000,"Pinned": true,"Timeout": 3}},{"Id": "TransferStreamFunction",
 "FunctionArn": "alias-arn", "FunctionConfiguration": {"Executable":
 "transfer_stream.function_handler", "MemorySize": 16000,"Pinned":
 true,"Timeout": 5}}]}'
```

> **Note**
>
> `Timeout` est requis par la version de définition de fonction, mais `GGStreamManager` ne l'utilise pas. Pour plus d'informations sur`Timeout`et d'autres paramètres au niveau du groupe, voirthe section called “Contrôle de l'exécution de la fonction Greengrass Lambda”.

2. Copiez la `LatestVersionArn` à partir de la sortie. Vous utilisez cette valeur pour ajouter la version de définition de fonction à la version de groupe que vous déployez pour le noyau.

Étape 4 : Créer une définition et une version de l'enregistreur

Configurez les paramètres de journalisation du groupe. Dans le cadre de ce didacticiel, vous configurezAWS IoT Greengrassles composants système, les fonctions Lambda définies par l'utilisateur et les connecteurs pour écrire des journaux dans le système de fichiers de l'appareil principal (noyau). Vous pouvez utiliser les journaux pour résoudre les problèmes que vous rencontrez. Pour plus d'informations, consultez the section called “Surveillance avec les journaux AWS IoT Greengrass”.

1. Créez une définition d'enregistreur qui inclut une version initiale.

JSON Expanded

```
aws greengrass create-logger-definition --name "LoggingConfigs" --initial-
version '{
    "Loggers": [
        {
            "Id": "1",
            "Component": "GreengrassSystem",
            "Level": "INFO",
            "Space": 10240,
            "Type": "FileSystem"
        },
        {
            "Id": "2",
            "Component": "Lambda",
            "Level": "INFO",
            "Space": 10240,
            "Type": "FileSystem"
        }
    ]
}'
```

JSON Single-line

```
aws greengrass create-logger-definition \
    --name "LoggingConfigs" \
    --initial-version '{"Loggers":
[{"Id":"1","Component":"GreengrassSystem","Level":"INFO","Space":10240,"Type":"FileSyste
{"Id":"2","Component":"Lambda","Level":"INFO","Space":10240,"Type":"FileSystem"}]}'
```

2. Copiez l'élément `LatestVersionArn` de la définition de l'enregistreur à partir de la sortie. Vous utilisez cette valeur pour ajouter la version de la définition de l'enregistreur à la version de groupe que vous déployez sur le noyau.

Étape 5 : Obtenir l'ARN de votre version de définition du noyau

Obtenez l'ARN de la version de définition du noyau pour ensuite l'ajouter à votre nouvelle version de groupe. Pour que vous puissiez déployer une version de groupe, cette dernière doit référencer une version de définition du noyau contenant exactement un noyau.

1. Obtenez les ID du groupe et de la version de groupe Greengrass cible. Cette procédure suppose qu'il s'agit du dernier groupe et de la dernière version de groupe les plus récents La requête suivante renvoie le groupe créé le plus récemment.

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

   Vous pouvez également procéder à une interrogation par nom. Les noms de groupe ne devant pas nécessairement être uniques, plusieurs groupes peuvent être renvoyés.

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

   > **Note**
   >
   > Vous pouvez également trouver ces valeurs dans leAWS IoTconsole L'ID du groupe s'affiche sur la page Paramètres du groupe. Les ID de version de groupe sont affichés dans leDéploiementsOnglet.

2. Copiez l'`Id` du groupe cible à partir de la sortie. Vous utilisez cela pour obtenir la version de définition du noyau et lorsque vous déployez le groupe.
3. Copiez l'élément `LatestVersion` à partir de la sortie (ID de la dernière version ajoutée au groupe). Vous utilisez cela pour obtenir la version de la définition du noyau.
4. Obtenir l'ARN de la version de la définition du noyau :

   a. Obtenez la version de groupe.

      - Remplacez *`group-id`* par l'`Id` que vous avez copié pour le groupe.
      - Remplacez*`group-version-id`*avec le`LatestVersion`que vous avez copiés pour le groupe.

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

   b. Copiez la `CoreDefinitionVersionArn` à partir de la sortie. Vous utilisez cette valeur pour ajouter la version de définition du noyau à la version de groupe que vous déployez sur le noyau.

Étape 6 : Créer une version de groupe

Maintenant, vous êtes prêt à créer une version de groupe contenant tous les éléments que vous souhaitez déployer. Pour ce faire, vous devez créer une version de groupe qui fait référence à la version cible de chaque type de composant. Pour ce didacticiel, vous incluez une version de définition du noyau, une version de définition de fonction et une version de définition de l'enregistreur.

1. Créer une version de groupe.

   - Remplacez *`group-id`* par l'`Id` que vous avez copié pour le groupe.
   - Remplacez*`core-definition-version-arn`*avec le`CoreDefinitionVersionArn`que vous avez copié pour la version de définition du noyau.
   - Remplacez*`function-definition-version-arn`*avec le`LatestVersionArn`que vous avez copié pour votre nouvelle version de définition de fonction.
   - Remplacez*`logger-definition-version-arn`*avec le`LatestVersionArn`que vous avez copié pour votre nouvelle version de définition de l'enregistreur.

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

2. Copiez la `Version` à partir de la sortie. Il s'agit de l'ID de la nouvelle version de groupe.

Étape 7 : Créer un déploiement

Déployer le groupe sur l'appareil principal (noyau)

1. Vérifiez que le`AWS IoT Greengrass`core est en cours d'exécution. Dans la fenêtre de terminal de votre Raspberry Pi, exécutez les commandes suivantes, si nécessaire.

   a. Pour vérifier si le démon est en cours d'exécution :

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

Si la sortie contient une entrée `root` pour `/greengrass/ggc/packages/`*`ggc-version`*`/bin/daemon`, le démon est en cours d'exécution.

> **Note**
>
> La version du chemin d'accès dépend de la version du logiciel AWS IoT Greengrass Core installée sur votre appareil principal.

b. Pour démarrer le démon :

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

2. Créer un déploiement.

   - Remplacez *`group-id`* par l'`Id` que vous avez copié pour le groupe.
   - Remplacez*`group-version-id`*avec le`Version`que vous avez copié pour la nouvelle version du groupe.

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

3. Copiez la `DeploymentId` à partir de la sortie.
4. Obtenir le statut du déploiement.

   - Remplacez *`group-id`* par l'`Id` que vous avez copié pour le groupe.
   - Remplacez *`deployment-id`* par l'`DeploymentId` que vous avez copié pour le déploiement.

```
aws greengrass get-deployment-status \
--group-id group-id \
--deployment-id deployment-id
```

Si le statut est`Success`, le déploiement a réussi. Pour bénéficier d'une aide à la résolution des problèmes, consultez Résolution des problèmes.

Étape 8 : Tester l'application

LeTransferStreamLa fonction Lambda génère des données d'appareil simulées. Elle écrit des données dans un flux que le gestionnaire de flux exporte vers le flux de données Kinesis cible.

1. Dans la console Amazon Kinesis, sousKinesis data flux de données, choisissezMyKinesisStream.

   > **Note**
   >
   > Si vous avez exécuté le didacticiel sans flux de données Kinesis cible, recherchez dans le fichier journal le gestionnaire de flux (`GGStreamManager`). S'il contient `export stream MyKinesisStream doesn't exist` dans un message d'erreur, le test est réussi. Cette erreur signifie que le service a essayé d'effectuer une exportation vers le flux mais que le flux n'existe pas.

2. Dans la pageMyKinesisStreampage, choisissezSurveillance. Si le test réussit, vous devriez voir des données dans les graphiques PutRecords . Selon votre connexion, l'affichage des données peut prendre une minute.

   > **Important**
   >
   > Lorsque vous avez terminé le test, supprimez le flux de données Kinesis pour éviter d'entraîner des frais supplémentaires.
   > Vous pouvez aussi exécuter la commande suivante pour arrêter le démon Greengrass. Cela empêche le noyau d'envoyer des messages jusqu'à ce que vous soyez prêt à continuer le test.
   >
   > ```
   > cd /greengrass/ggc/core/
   > sudo ./greengrassd stop
   > ```

3. Supprimez leTransferStreamUne fonction Lambda depuis le cœur.

   a. Suivez l'the section called “Créer une version de groupe” pour créer une nouvelle version de groupe, mais supprimez l'option `--function-definition-version-arn` dans la commande `create-group-version`. Vous pouvez également créer une version de définition de fonction qui n'inclut pas leTransferStreamfonction Lambda.

> **Note**
>
> En omettant le système`GGStreamManager`Installer la fonction Lambda à partir de la version du groupe déployé, vous désactivez la gestion de flux sur le noyau.

b. Suivez l'the section called “Créer un déploiement” pour déployer la nouvelle version du groupe.

Pour afficher les informations de journalisation ou résoudre les problèmes liés aux flux, recherchez les fonctions `GGStreamManager` et `TransferStream` dans les journaux. Vous devez disposer d'autorisations `root` pour pouvoir lire les journaux AWS IoT Greengrass sur le système de fichiers.

- `TransferStream` écrit les entrées de journal dans *`greengrass-root`*`/ggc/var/log/user/`*`region`*`/`*`account-id`*`/TransferStream.log`.
- `GGStreamManager` écrit les entrées de journal dans *`greengrass-root`*`/ggc/var/log/system/GGStreamManager.log`.

Si vous avez besoin de plus d'informations de dépannage, vous pouvez définir le niveau de journalisation `Lambda` sur `DEBUG`, puis créer et déployer une nouvelle version de groupe.

Consulter aussi

- Gestion des flux de données
- the section called “Utiliser StreamManagerClient pour travailler avec des flux”
- the section called “Configurations d'exportation pour prises en chargeAWS Clouddestinations”
- the section called “Configuration du gestionnaire de flux”
- the section called “Exportation de flux de données (console)”
- AWS Identity and Access ManagementCommandes (IAM)dans leAWS CLIRéférence des commandes
- AWS Lambdacommandesdans leAWS CLIRéférence des commandes
- AWS IoT Greengrasscommandesdans leAWS CLIRéférence des commandes

Déployer des secrets sur AWS IoT Greengrass Core

Cette fonction est disponible pourAWS IoT GreengrassCore v1.7 et versions ultérieures.

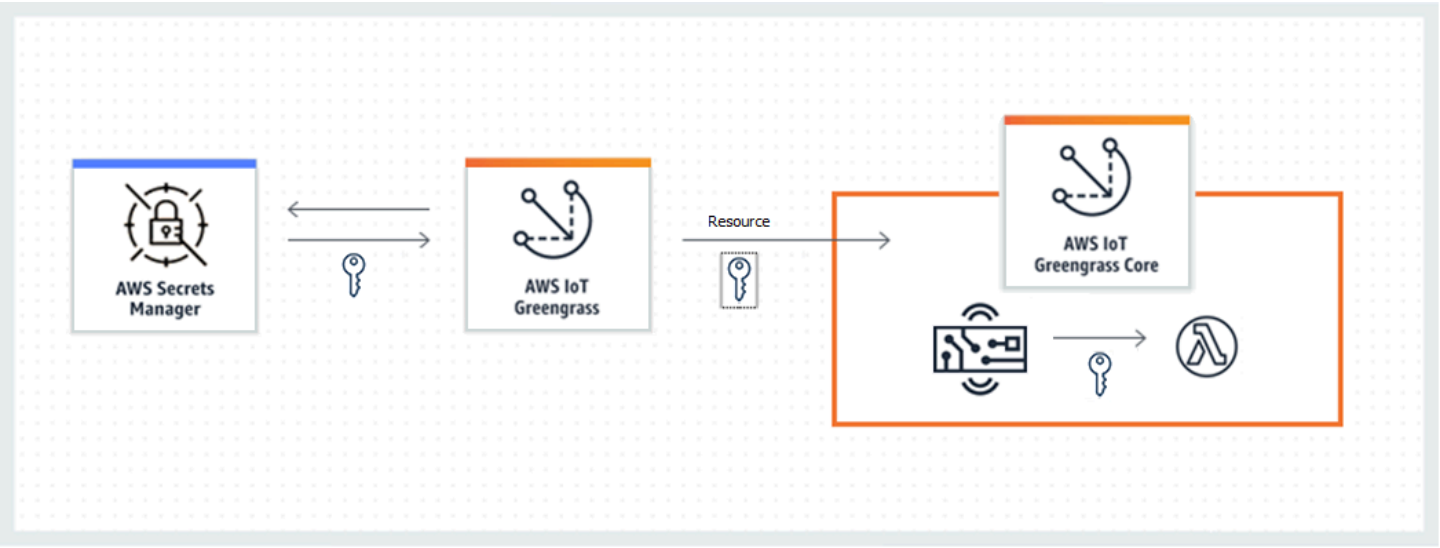
AWS IoT Greengrass vous permet de vous authentifier auprès des services et applications des appareils Greengrass sans codage en dur de mots de passe, jetons ou d'autres secrets.

AWS Secrets Managerest un service que vous pouvez utiliser pour stocker et gérer vos secrets en toute sécurité dans le cloud.AWS IoT Greengrassétend Secrets Manager aux périphériques principaux de Greengrass, de sorte que votreconnecteursLes fonctions Lambda peuvent utiliser des secrets locaux pour interagir avec des services et des applications. Par exemple, le connecteur Twilio Notifications utilise un jeton d'authentification stocké localement.

Pour intégrer un secret dans un groupe Greengrass, vous devez créer une ressource de groupe qui référence le secret de Secrets Manager. Cette ressource de secret référence le secret du cloud par ARN. Pour découvrir comment créer, gérer et utiliser les ressources de secret, consultez the section called “Utiliser les ressources de secret”.

AWS IoT Greengrass chiffre vos secrets en transit et au repos. Pendant le déploiement de groupe,AWS IoT Greengrassextrait le secret de Secrets Manager et crée une copie chiffrée locale dans le noyau Greengrass. Après avoir effectué la rotation des secrets de votre cloud dans Secrets Manager, redéployez le groupe pour propager les valeurs mises à jour dans le noyau.

Le schéma suivant montre le processus de haut niveau consistant à déployer un secret sur le noyau. Les secrets sont chiffrés au repos et en transit.



L'utilisation de AWS IoT Greengrass pour stocker vos secrets en local offre les avantages suivants :

- Dissocié du code (pas codé en dur). Cela prend en charge les informations d'identification gérées de manière centralisée et permet de protéger les données sensibles du risque de divulgation.
- Disponible pour les scénarios hors ligne. Les connecteurs et les fonctions peuvent accéder en toute sécurité aux logiciels et services locaux lorsqu'ils sont déconnectés d'Internet.
- Accès contrôlé aux secrets. Seuls les connecteurs et fonctions autorisés dans le groupe peuvent accéder à vos secrets. AWS IoT Greengrass utilise le chiffrement de clé privée pour sécuriser vos secrets. Les secrets sont chiffrés au repos et en transit. Pour plus d'informations, consultez the section called “Chiffrement des secrets”.
- Rotation contrôlée. Après avoir effectué la rotation des secrets dans Secrets Manager, redéployez le groupe Greengrass pour mettre à jour les copies locales de vos secrets. Pour plus d'informations, consultez the section called “Création et gestion des secrets”.

> **Important**
>
> AWS IoT Greengrass ne met pas automatiquement à jour les valeurs des secrets locaux après la rotation des versions du cloud. Pour mettre à jour les valeurs locales, vous devez redéployer le groupe.

Chiffrement des secrets

AWS IoT Greengrass chiffre les secrets au repos et en transit.

> **Important**
>
> Assurez-vous que vos fonctions Lambda définies par l'utilisateur gèrent les secrets en toute sécurité et ne consignent aucune donnée sensible stockée dans le secret. Pour de plus amples informations, veuillez consulterAtténuation des risques liés à la journalisation et au débogage de votre fonction Lambdadans leAWS Secrets ManagerGuide de l'utilisateur. Bien que cette documentation se réfère spécifiquement aux fonctions de rotation, la recommandation s'applique également aux fonctions Lambda de Greengrass.

Chiffrement en transit

AWS IoT Greengrass utilise le protocole TLS (Transport Layer Security) pour chiffrer les communications sur Internet et le réseau local. Cela a pour effet de protéger les secrets en transit,

ce qui se produit lorsque les secrets sont récupérés à partir de Secrets Manager et déployés dans le noyau. Pour les suites de chiffrement TLS prises en charge, consultez the section called “Prise en charge des suites de chiffrement TLS”.

Chiffrement au repos

AWS IoT Greengrass utilise la clé privée spécifiée dans `config.json` pour le chiffrement des secrets qui sont stockés sur le noyau. Par conséquent, le stockage sécurisé de la clé privée est essentiel pour protéger les secrets locaux. Dans le AWS modèle de responsabilité partagée, il est de la responsabilité du client de garantir un stockage sécurisé de la clé privée sur l'appareil principal.

AWS IoT Greengrass prend en charge deux modes de stockage de clé privée :

- l'utilisation des modules de sécurité matérielle. Pour plus d'informations, consultez the section called “Intégration de sécurité matérielle”.

  > **Note**
  >
  > Actuellement,AWS IoT Greengrassseul est pris en charge lePKCS #1 v1.5mécanisme de remplissage pour le chiffrement et le déchiffrement des secrets locaux lors de l'utilisation de clés privées matérielles. Si vous suivez les instructions fournies par le fournisseur pour générer manuellement des clés privées matérielles, veillez à choisir PKCS #1 v1.5.AWS IoT Greengrassne prend pas en charge le rembourrage OAEP (Optimal Asymmetric Encryption Padded).

- l'utilisation des autorisations de système de fichiers (par défaut).

La clé privée est utilisée pour sécuriser la clé de données, qui est utilisée pour chiffrer les secrets locaux. La clé de données effectue une rotation avec chaque déploiement de groupe.

LeAWS IoT Greengrasscore est la seule entité qui a accès à la clé privée. Les connecteurs Greengrass ou les fonctions Lambda qui sont rattachés à une ressource secrète obtiennent la valeur de la clé secrète à partir de la base.

Prérequis

Vous trouverez ci-dessous les exigences minimales requises pour la prise en charge des secrets locaux :

- Vous devez utiliserAWS IoT GreengrassCore v1.7 ou version ultérieure.
- Pour obtenir les valeurs des secrets locaux, vos fonctions Lambda définies par l'utilisateur doivent utiliserAWS IoT GreengrassCore SDK v1.3.0 ou version ultérieure.
- La clé privée utilisée pour le chiffrement des secrets locaux doit être spécifiée dans le fichier de configuration Greengrass. Par défaut, AWS IoT Greengrass utilise la clé privée principale stockée dans le système de fichiers. Pour fournir votre propre clé privée, consultez the section called “Spécifier la clé privée pour chiffrer un secret”. Seul le type de clé RSA est pris en charge.

  > **Note**
  >
  > Actuellement,AWS IoT Greengrassseul est pris en charge lePKCS #1 v1.5mécanisme de remplissage pour le chiffrement et le déchiffrement des secrets locaux lors de l'utilisation de clés privées matérielles. Si vous suivez les instructions fournies par le fournisseur pour générer manuellement des clés privées matérielles, veillez à choisir PKCS #1 v1.5.AWS IoT Greengrassne prend pas en charge le rembourrage OAEP (Optimal Asymmetric Encryption Padded).

- AWS IoT Greengrass doit recevoir l'autorisation d'obtenir vos valeurs secrètes. Cela permet à AWS IoT Greengrass d'extraire les valeurs pendant le déploiement de groupe. Si vous utilisez le rôle de service Greengrass par défaut, AWS IoT Greengrass possède déjà l'accès aux secrets dont les noms commencent par greengrass-. Pour personnaliser l'accès, consultez the section called “Autoriser AWS IoT Greengrass à obtenir les valeurs secrètes”.

  > **Note**
  >
  > Nous vous recommandons d'utiliser cette convention d'affectation de noms pour identifier les secrets auxquels AWS IoT Greengrass est autorisé à accéder, même si vous personnalisez les autorisations. La console utilise différentes autorisations pour lire vos secrets, il est donc possible que vous puissiez sélectionner des secrets dans la console que AWS IoT Greengrass n'est pas autorisé à récupérer. Utiliser une convention de dénomination peut vous aider à éviter un conflit d'autorisation, qui se traduit par une erreur de déploiement.

Spécifier la clé privée pour chiffrer un secret

Dans cette procédure, vous devez fournir le chemin d'accès à une clé privée utilisée pour le chiffrement secret local. Il doit s'agir d'une clé RSA avec une longueur minimale de 2 048 bits. Pour de plus amples informations sur les clés privées utilisées sur leAWS IoT GreengrassCore, veuillez consulterthe section called "Mandataires de sécurité".

AWS IoT Greengrass prend en charge deux modes de stockage de clé privée : le stockage basé sur le matériel ou le stockage basé sur le système de fichiers (par défaut). Pour plus d'informations, consultez the section called "Chiffrement des secrets".

Suivez cette procédure uniquement si vous souhaitez modifier la configuration par défaut, qui utilise la clé privée principale dans le système de fichiers. Ces étapes sont rédigées en supposant que vous avez créé votre groupe et noyau, comme indiqué dans le Module 2 du didacticiel Démarrez.

1. Ouvrez le fichier `config.json` situé dans le répertoire */greengrass-root*`/config`.

   > **Note**
   >
   > *greengrass-root* indique le chemin d'installation du logiciel AWS IoT Greengrass Core sur votre appareil. Généralement, il s'agit du répertoire `/greengrass`.

2. Dans l'objet `crypto.principals.SecretsManager`, pour la propriété `privateKeyPath`, entrez le chemin d'accès de votre clé privée :

   - Si votre clé privée est stockée dans le système de fichiers, indiquez le chemin d'accès absolu de la clé. Par exemple :

     ```
     "SecretsManager" : {
       "privateKeyPath" : "file:///somepath/hash.private.key"
     }
     ```

   - Si votre clé privée est stockée dans un module de sécurité matériel (HSM), spécifiez le chemin d'accès à l'aide du schéma d'URI RFC 7512 PKCS#11. Par exemple :

     ```
     "SecretsManager" : {
       "privateKeyPath" : "pkcs11:object=private-key-label;type=private"
     }
     ```

     Pour plus d'informations, consultez the section called "Configuration de sécurité matérielle".

> **Note**
>
> Actuellement,AWS IoT Greengrassseul est pris en charge lePKCS #1 v1.5mécanisme de remplissage pour le chiffrement et le déchiffrement des secrets locaux lors de l'utilisation de clés privées matérielles. Si vous suivez les instructions fournies par le fournisseur pour générer manuellement des clés privées matérielles, veillez à choisir PKCS #1 v1.5.AWS IoT Greengrassne prend pas en charge le rembourrage OAEP (Optimal Asymmetric Encryption Padded).

Autoriser AWS IoT Greengrass à obtenir les valeurs secrètes

Dans cette procédure, vous pouvez ajouter une stratégie en ligne au rôle de service Greengrass qui permet à AWS IoT Greengrass d'obtenir les valeurs de vos secrets.

Suivez cette procédure uniquement si vous souhaitez accorder à AWS IoT Greengrass des autorisations personnalisées pour accéder à vos secrets ou si votre rôle de service Greengrass n'inclut pas la stratégie gérée `AWSGreengrassResourceAccessRolePolicy`. `AWSGreengrassResourceAccessRolePolicy` accorde l'accès aux secrets dont les noms commencent par greengrass-.

1. Exécutez la commande de l'interface en ligne de commande pour obtenir l'ARN du rôle de service Greengrass :

   ```
   aws greengrass get-service-role-for-account --region region
   ```

   L'ARN renvoyé contient le nom du rôle.

   ```
   {
     "AssociatedAt": "time-stamp",
     "RoleArn": "arn:aws:iam::account-id:role/service-role/role-name"
   }
   ```

   Vous utilisez l'ARN ou le nom dans l'étape suivante.

2. Ajouter une stratégie en ligne qui autorise l'action `secretsmanager:GetSecretValue`. Pour obtenir des instructions, consultezAjout et suppression de stratégies IAMdans leIAM User Guide.

Vous pouvez octroyer un accès précis en répertoriant explicitement des secrets ou en utilisant un schéma d'attribution de nom de caractère générique *, ou vous pouvez accorder un accès conditionnel aux secrets balisés ou versionnés. Par exemple, la stratégie suivante permet à AWS IoT Greengrass de lire uniquement les secrets spécifiés.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue"
            ],
            "Resource": [
                "arn:aws:secretsmanager:region:account-id:secret:greengrass-
SecretA-abc",
                "arn:aws:secretsmanager:region:account-id:secret:greengrass-
SecretB-xyz"
            ]
        }
    ]
}
```

> **Note**
>
> Si vous utilisez une clé AWS KMS gérée par le client pour chiffrer les secrets, votre rôle de service Greengrass doit également autoriser l'action `kms:Decrypt`.

Pour de plus amples informations sur les stratégies IAM pour Secrets Manager, veuillez consulterAuthentification et contrôle d'accès pourAWS Secrets ManageretActions, ressources et clés de contexte pouvant être utilisées dans une stratégie IAM ou une stratégie de secret pourAWS Secrets Managerdans leAWS Secrets ManagerGuide de l'utilisateur.

Consulter aussi

- Qu'est-ce que AWS Secrets Manager ? dans le Guide de l'utilisateur AWS Secrets Manager
- PIÈCES #1 : Cryptage RSA version 1.5

Utilisation des ressources de secret

AWS IoT Greengrass utilise des ressources de secret pour intégrer les secrets d'AWS Secrets Manager dans un groupe Greengrass. Une ressource secrète est une référence à un secret du Secrets Manager. Pour plus d'informations, veuillez consulter Déployer des secrets sur Core.

Sur le périphérique AWS IoT Greengrass principal, les connecteurs et les fonctions Lambda peuvent utiliser la ressource secrète pour s'authentifier auprès des services et des applications, sans avoir à coder en dur des mots de passe, des jetons ou d'autres informations d'identification.

Création et gestion des secrets

Dans un groupe Greengrass, une ressource secrète fait référence à l'ARN d'un secret Secrets Manager. Lorsque la ressource secrète est déployée vers le cœur, la valeur du secret est cryptée et mise à la disposition des connecteurs affiliés et des fonctions Lambda. Pour plus d'informations, veuillez consulter the section called “Chiffrement des secrets”.

Vous utilisez Secrets Manager pour créer et gérer les versions cloud de vos secrets. Vous utilisez AWS IoT Greengrass pour créer, gérer et déployer vos ressources de secret.

> **Important**
>
> Nous vous recommandons de suivre la meilleure pratique qui consiste à alterner vos secrets dans Secrets Manager. Ensuite, déployez le groupe Greengrass pour mettre à jour les copies locales de vos secrets. Pour plus d'informations, consultez la section Rotation de vos AWS Secrets Manager secrets dans le guide de AWS Secrets Manager l'utilisateur.

Pour rendre un secret disponible sur le noyau Greengrass

1. Créez un secret dans Secrets Manager. Il s'agit de la version cloud de votre secret, qui est stockée et gérée de manière centralisée dans Secrets Manager. Les tâches de gestion comprennent la rotation des valeurs secrètes et l'application des stratégies de ressources.
2. Créez une ressource de secret dans AWS IoT Greengrass. Il s'agit d'un type de ressource de groupe qui référence le secret du cloud par ARN. Vous ne pouvez référencer un secret qu'une seule fois par groupe.
3. Configurez votre connecteur ou votre fonction Lambda. Vous devez associer la ressource à un connecteur ou à une fonction en spécifiant des paramètres ou propriétés correspondants.

Cela leur permet d'obtenir la valeur de la ressource de secret localement déployée. Pour plus d'informations, veuillez consulter the section called “Utilisation des secrets locaux”.

4. Déploiement du groupe Greengrass. Pendant le déploiement, AWS IoT Greengrass récupère la valeur du secret du cloud et crée (ou met à jour) le secret local dans le noyau.

Secrets Manager enregistre un événement AWS CloudTrail chaque fois qu'il AWS IoT Greengrass récupère une valeur secrète. AWS IoT Greengrassn'enregistre aucun événement lié au déploiement ou à l'utilisation de secrets locaux. Pour plus d'informations sur la journalisation de Secrets Manager, consultez la section Surveiller l'utilisation de vos AWS Secrets Manager secrets dans le Guide de AWS Secrets Manager l'utilisateur.

Inclure des étiquettes intermédiaires dans les ressources de secret

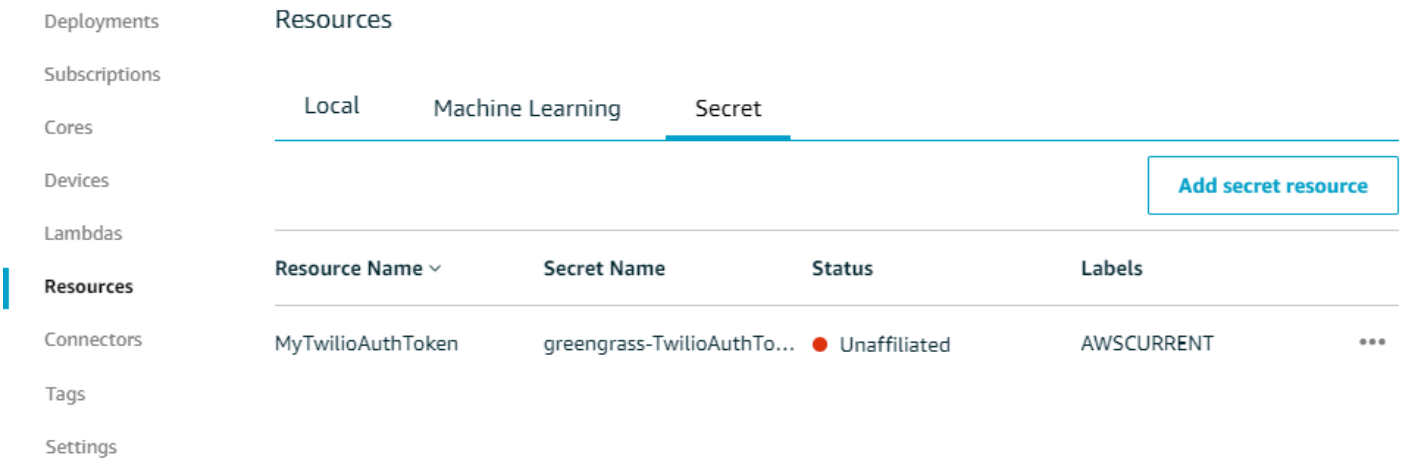
Secrets Manager utilise des étiquettes intermédiaires pour identifier des versions spécifiques d'une valeur secrète. Les étiquettes de mise en scène peuvent être définies par le système ou définies par l'utilisateur. Secrets Manager attribue l'`AWSCURRENT`étiquette à la version la plus récente de la valeur secrète. Les étiquettes intermédiaires sont couramment utilisées pour gérer la rotation des secrets. Pour plus d'informations sur le versionnement de Secrets Manager, consultez la section Termes et concepts clés du Guide de l'AWS Secrets Managerutilisateur. AWS Secrets Manager

Les ressources secrètes incluent toujours l'étiquette `AWSCURRENT` de préparation, et elles peuvent éventuellement inclure d'autres étiquettes de préparation si elles sont requises par une fonction ou un connecteur Lambda. Pendant le déploiement de groupe, AWS IoT Greengrass récupère les valeurs des étiquettes intermédiaires qui sont référencées dans le groupe, puis crée ou met à jour les valeurs correspondantes sur le noyau.

Créer et gérer les ressources de secret (console)

Création des ressources de secret (console)

Dans la AWS IoT Greengrass console, vous pouvez créer et gérer des ressources secrètes depuis l'onglet Secrets de la page Ressources du groupe. Pour les didacticiels concernant la création d'une ressource de secret et son ajout à un groupe, consultez the section called “Comment créer une ressource de secret (console)” et the section called “Démarrer avec les connecteurs (console)”.



> **Note**
>
> La console vous permet également de créer une ressource secrète et secrète lorsque vous configurez un connecteur ou une fonction Lambda. Vous pouvez le faire depuis la page Configurer les paramètres du connecteur ou depuis la page Ressources de la fonction Lambda.

Gestion des ressources de secret (console)

Les tâches de gestion des ressources secrètes de votre groupe Greengrass incluent l'ajout de ressources secrètes au groupe, la suppression de ressources secrètes du groupe et la modification de l'ensemble des étiquettes intermédiaires incluses dans une ressource secrète.

Si vous pointez vers un secret différent de celui de Secrets Manager, vous devez également modifier tous les connecteurs qui utilisent le secret :

1. Sur la page de configuration de groupe, choisissez Connecteurs.
2. Dans le menu contextuel du connecteur, choisissez Modifier.
3. La page Edit parameters (Modifier les paramètres) affiche un message vous informant que l'ARN du secret a changé. Pour confirmer le changement, choisissez Enregistrer.

Si vous supprimez un secret dans Secrets Manager, supprimez la ressource secrète correspondante du groupe ainsi que des connecteurs et des fonctions Lambda qui le référencent. Sinon, lors du déploiement en groupe, AWS IoT Greengrass renvoie une erreur indiquant que le secret est introuvable. Mettez également à jour votre code de fonction Lambda si nécessaire.

Créer et gérer les ressources de secret (interface de ligne de commande)

Création des ressources de secret (interface de ligne de commande)

Dans l'API AWS IoT Greengrass, un secret est un type de ressource de groupe. L'exemple suivant crée une définition de ressource avec une version initiale qui inclut une ressource de secret nommée `MySecretResource`. Pour obtenir un didacticiel concernant la création d'une ressource de secret et son ajout à une version de groupe, consultez the section called “Démarrer avec les connecteurs (CLI)”.

La ressource secrète fait référence à l'ARN du secret Secrets Manager correspondant et inclut deux étiquettes intermédiaires en plus de`AWSCURRENT`, qui sont toujours incluses.

```
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"
                    ]
                }
            }
        }
    ]
}'
```

Gestion des ressources de secret (interface de ligne de commande)

Les tâches de gestion des ressources secrètes de votre groupe Greengrass incluent l'ajout de ressources secrètes au groupe, la suppression de ressources secrètes du groupe et la modification de l'ensemble des étiquettes intermédiaires incluses dans une ressource secrète.

Dans l'API AWS IoT Greengrass, ces modifications sont mises en œuvre à l'aide des versions.

L'AWS IoT GreengrassAPI utilise des versions pour gérer les groupes. Les versions étant immuables, pour ajouter ou modifier des composants de groupe (par exemple, les appareils clients, les fonctions et les ressources du groupe), vous devez créer des versions de composants nouveaux ou mis à jour. Ensuite, vous créez et déployez une version de groupe contenant la version cible de chaque composant. Pour en savoir plus sur les groupes, voirthe section called “Groupes AWS IoT Greengrass”.

Par exemple, pour modifier l'ensemble des étiquettes intermédiaires d'une ressource de secret :

1. Créez une version de définition de ressource qui contient la ressource de secret mise à jour. L'exemple suivant ajoute une troisième étiquette intermédiaire à la ressource de secret à partir de la section précédente.

   > **Note**
   >
   > Pour ajouter d'autres ressources à la version, incluez-les dans la grappe `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. Si l'ID de la ressource de secret est modifié, mettez à jour les connecteurs et les fonctions qui utilisent la ressource de secret. Dans les nouvelles versions, mettez à jour le paramètre ou la

propriété qui correspond à l'ID de ressource. Si l'ARN du secret est modifié, vous devez également mettre à jour le paramètre correspondant pour tous les connecteurs qui utilisent le secret.

> **Note**
>
> L'ID de ressource est un identifiant arbitraire qui est fourni par le client.

3. Créez une version de groupe qui contient la version cible de chaque composant que vous souhaitez envoyer vers le noyau.
4. Déploiement de la version de groupe.

Pour obtenir un didacticiel qui montre comment créer et déployer des ressources de secret, des connecteurs et des fonctions, consultez the section called “Démarrer avec les connecteurs (CLI)”.

Si vous supprimez un secret dans Secrets Manager, supprimez la ressource secrète correspondante du groupe ainsi que des connecteurs et des fonctions Lambda qui le référencent. Sinon, lors du déploiement en groupe, AWS IoT Greengrass renvoie une erreur indiquant que le secret est introuvable. Mettez également à jour votre code de fonction Lambda si nécessaire. Vous pouvez supprimer un secret local en déployant une version de définition de ressource qui ne contient pas la ressource secrète correspondante.

Utilisation de secrets locaux dans les connecteurs et les fonctions Lambda

Les connecteurs Greengrass et les fonctions Lambda utilisent des secrets locaux pour interagir avec les services et les applications. La valeur `AWSCURRENT` est utilisée par défaut, mais les valeurs des autres étiquettes intermédiaires incluses dans la ressource de secret sont également disponibles.

Les connecteurs et les fonctions doivent être configurés avant de pouvoir accéder aux secrets locaux. Cela associe la ressource de secret au connecteur ou à la fonction.

Connecteurs

Si un connecteur requiert l'accès à un secret local, il fournit des paramètres que vous configurez avec les informations dont il a besoin pour accéder au secret.

- Pour savoir comment procéder dans la AWS IoT Greengrass console, consultezthe section called “Démarrer avec les connecteurs (console)”.
- Pour savoir comment procéder avec l'interface de ligne de commande AWS IoT Greengrass, consultez the section called “Démarrer avec les connecteurs (CLI)”.

Pour en savoir plus sur les exigences des connecteurs individuels, consultez the section called “AWS- connecteurs Greengrass fournis”.

La logique pour accéder et utiliser le secret est intégré dans le connecteur.

Fonctions Lambda

Pour autoriser une fonction Greengrass Lambda à accéder à un secret local, vous devez configurer les propriétés de la fonction.

- Pour savoir comment procéder dans la AWS IoT Greengrass console, consultezthe section called “Comment créer une ressource de secret (console)”.
- Pour ce faire, dans l'API AWS IoT Greengrass, vous fournissez les informations suivantes dans la propriété `ResourceAccessPolicies`.
  - `ResourceId` : l'ID de la ressource de secret dans le groupe Greengrass. Il s'agit de la ressource qui référence l'ARN du secret Secrets Manager correspondant.
  - `Permission` : type d'accès que la fonction a à la ressource. Seule l'autorisation `ro` (en lecture seule) est prise en charge pour les ressources de secret.

L'exemple suivant crée une fonction Lambda qui peut accéder à la ressource `MyApiKey` secrète.

```
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
                }
```

```
            }
        }
    ]
}'
```

Pour accéder aux secrets locaux lors de l'exécution, les fonctions Greengrass Lambda appellent la `get_secret_value` fonction depuis le `secretsmanager` client dans le SDK AWS IoT Greengrass Core (v1.3.0 ou version ultérieure).

L'exemple suivant montre comment utiliser le SDK AWS IoT Greengrass Core pour Python pour obtenir un secret. Il transmet le nom du secret à la `get_secret_value` fonction. `SecretId`peut être le nom ou l'ARN du secret du Secrets Manager (pas la ressource secrète).

```
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")
```

Pour les secrets de type texte, la fonction `get_secret_value` renvoie une chaîne. Pour les secrets de type binaire, elle renvoie une chaîne encodée en base64.

> **⚠ Important**
>
> Assurez-vous que les fonctions Lambda définies par l'utilisateur gèrent les secrets de manière sécurisée et n'enregistrent aucune donnée sensible stockée dans le secret. Pour plus d'informations, consultez la section Atténuer les risques liés à la journalisation et au débogage de votre fonction Lambda dans AWS Secrets Managerle guide de l'utilisateur. Bien que cette documentation fasse spécifiquement référence aux fonctions de rotation, la recommandation s'applique également aux fonctions Greengrass Lambda.

La valeur actuelle du secret est renvoyée par défaut. Il s'agit de la version à laquelle l'étiquette intermédiaire `AWSCURRENT` est attachée. Pour accéder à une autre version, transmettez le nom de l'étiquette intermédiaire correspondante pour l'argument `VersionStage` facultatif. Par exemple :

```
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")
```

Pour obtenir un autre exemple de fonction qui appelle `get_secret_value`, consultez Création d'un package de déploiement de fonctions Lambda.

Comment créer une ressource de secret (console)

Cette fonctionnalité est disponible pour AWS IoT Greengrass Core v1.7 et versions ultérieures.

Ce didacticiel montre comment utiliser la AWS Management Console pour ajouter une ressource de secret à un groupe Greengrass. Une ressource de secret est une référence à un secret de AWS Secrets Manager. Pour plus d'informations, veuillez consulter Déployer des secrets sur Core.

Sur le périphérique AWS IoT Greengrass principal, les connecteurs et les fonctions Lambda peuvent utiliser la ressource secrète pour s'authentifier auprès des services et des applications, sans avoir à coder en dur des mots de passe, des jetons ou d'autres informations d'identification.

Dans ce didacticiel, vous commencez par créer une information secrète dans la console AWS Secrets Manager. Ensuite, dans la AWS IoT Greengrass console, vous ajoutez une ressource

secrète à un groupe Greengrass depuis la page Ressources du groupe. Cette ressource secrète fait référence au secret du Secrets Manager. Plus tard, vous associez la ressource secrète à une fonction Lambda, ce qui permet à la fonction d'obtenir la valeur du secret local.

> **Note**
>
> La console vous permet également de créer une ressource secrète et secrète lorsque vous configurez un connecteur ou une fonction Lambda. Vous pouvez le faire depuis la page Configurer les paramètres du connecteur ou depuis la page Ressources de la fonction Lambda.
> Seuls les connecteurs qui contiennent les paramètres des secrets peuvent accéder aux secrets. Pour un didacticiel qui montre comment le connecteur Twilio Notifications utilise un jeton d'authentification stocké localement, voirthe section called “Démarrer avec les connecteurs (console)”.

Le didacticiel contient les étapes détaillées suivantes :

1. Création d'un secret dans le Gestionnaire de Secrets
2. Ajouter une ressource de secret à un groupe
3. Création d'un package de déploiement de fonctions Lambda
4. Création d'une fonction Lambda
5. Ajouter la fonction au groupe
6. Attacher la ressource de secret à la fonction
7. Ajouter des abonnements au groupe
8. Déployer le groupe
9. the section called “Test de la fonction Lambda ”

Le didacticiel devrait prendre environ 20 minutes.

Prérequis

Pour suivre ce didacticiel, vous devez disposer des éléments suivants :

- Un groupe Greengrass et un noyau Greengrass (v1.7 ou version ultérieure). Pour savoir comment créer un groupe et un service principal Greengrass, consultez Commencer avec AWS IoT

Greengrass. Le didacticiel Mise en route comprend également les étapes d'installation du logiciel AWS IoT Greengrass Core.

- AWS IoT Greengrass doit être configuré pour prendre en charge les secrets locaux. Pour plus d'informations, consultez les exigences liées aux ressources de secret.

> **Note**
>
> Cette exigence inclut l'autorisation d'accéder aux secrets de votre Gestionnaire de Secrets. Si vous utilisez le rôle de service Greengrass par défaut, Greengrass est autorisé à obtenir les valeurs des secrets dont le nom commence par greengrass-.

- Pour obtenir les valeurs des secrets locaux, vos fonctions Lambda définies par l'utilisateur doivent AWS IoT Greengrass utiliser le SDK Core v1.3.0 ou version ultérieure.

Étape 1 : Création d'un secret du Gestionnaire de Secrets

Au cours de cette étape, vous utilisez la console AWS Secrets Manager pour créer un secret.

1. Connectez-vous à la console AWS Secrets Manager.

   > **Note**
   >
   > Pour plus d'informations sur ce processus, voir Étape 1 : Création et stockage de votre secret AWS Secrets Manager dans le guide de AWS Secrets Manager l'utilisateur.

2. Choisissez Store a new secret (Stocker un nouveau secret).
3. Sous Choisir le type de secret, choisissez Autre type de secret.
4. Dans Spécifier les paires clé/valeur à stocker pour ce secret :
   - Pour Clé, entrez **`test`**.
   - Pour le champ Value (Valeur), entrez **`abcdefghi`**.
5. Gardez aws/secretsmanager sélectionné pour la clé de chiffrement, puis choisissez Next.

   > **Note**
   >
   > Vous n'êtes pas débité AWS KMS si vous utilisez la clé AWS gérée par défaut créée par Secrets Manager dans votre compte.

6. Pour Secret name, entrez **`greengrass-TestSecret`** et choisissez Next.

   > **Note**
   >
   > Par défaut, le rôle de service Greengrass permet d'AWS IoT Greengrassobtenir la valeur des secrets dont le nom commence par greengrass -. Pour plus d'informations, consultez les exigences liées aux secrets.

7. Ce didacticiel ne nécessite pas de rotation. Choisissez donc désactiver la rotation automatique, puis cliquez sur Suivant.
8. Dans la page Révision, passez en revue vos paramètres, puis choisissez Stocker.

   Ensuite, vous créez une ressource de secret dans votre groupe Greengrass qui référence le secret.

Étape 2 : Ajouter une ressource de secret à un groupe Greengrass

Au cours de cette étape, vous configurez une ressource de groupe qui fait référence au secret de Secrets Manager.

1. Dans le volet de navigation de la AWS IoT console, sous Gérer, développez les appareils Greengrass, puis choisissez Groups (V1).
2. Choisissez le groupe auquel vous souhaitez ajouter la ressource de secret.
3. Sur la page de configuration du groupe, choisissez l'onglet Ressources, puis faites défiler la page jusqu'à la section Secrets. La section Secrets affiche les ressources secrètes appartenant au groupe. Vous pouvez ajouter, modifier et supprimer des ressources secrètes dans cette section.

   > **Note**
   >
   > La console vous permet également de créer une ressource secrète et secrète lorsque vous configurez un connecteur ou une fonction Lambda. Vous pouvez le faire depuis la page Configurer les paramètres du connecteur ou depuis la page Ressources de la fonction Lambda.

4. Choisissez Ajouter dans la section Secrets.
5. Sur la page Ajouter une ressource secrète, entrez **`MyTestSecret`** le nom de la ressource.

6. Sous Secret, choisissez greengrass- TestSecret.
7. Dans la section Sélectionner les étiquettes (facultatif), AWSCURRENT l'étiquette intermédiaire représente la dernière version du secret. Cette étiquette est toujours incluse dans une ressource de secret.

   > **Note**
   >
   > Ce didacticiel nécessite uniquement l' AWSCURRENT étiquette. Vous pouvez éventuellement inclure des étiquettes requises par votre fonction ou connecteur Lambda.

8. Choisissez Add resource (Ajouter ressource).

Étape 3 : Création d'un package de déploiement de fonctions Lambda

Pour créer une fonction Lambda, vous devez d'abord créer un package de déploiement de fonction Lambda contenant le code de la fonction et les dépendances. Les fonctions Lambda de Greengrass nécessitent le SDK AWS IoT Greengrass principal pour des tâches telles que la communication avec les messages MQTT dans l'environnement principal et l'accès aux secrets locaux. Ce didacticiel crée une fonction Python afin que vous utilisiez la version Python du SDK dans le package de déploiement.

> **Note**
>
> Pour obtenir les valeurs des secrets locaux, vos fonctions Lambda définies par l'utilisateur doivent AWS IoT Greengrass utiliser le SDK Core v1.3.0 ou version ultérieure.

1. Sur la page de téléchargement du SDK AWS IoT Greengrass Core, téléchargez le SDK AWS IoT Greengrass Core pour Python sur votre ordinateur.
2. Décompressez le package téléchargé pour obtenir le kit SDK. Le kit SDK est représenté par le dossier `greengrasssdk`.
3. Enregistrez la fonction de code Python suivante dans un fichier local nommé `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` fonction prend en charge le nom ou l'ARN du secret du Secrets Manager pour la `SecretId` valeur. Cet exemple utilise le nom du secret. Pour cet exemple, AWS IoT Greengrass renvoie la paire clé-valeur : `{"test":"abcdefghi"}`.

> **Important**
>
> Assurez-vous que les fonctions Lambda définies par l'utilisateur gèrent les secrets de manière sécurisée et n'enregistrent aucune donnée sensible stockée dans le secret. Pour plus d'informations, consultez la section Atténuer les risques liés à la journalisation et au débogage de votre fonction Lambda dans AWS Secrets Managerle guide de l'utilisateur. Bien que cette documentation fasse spécifiquement référence aux fonctions de rotation, la recommandation s'applique également aux fonctions Greengrass Lambda.

4. Compressez les éléments suivants dans un fichier nommé `secret_test_python.zip`. Lorsque vous créez le fichier ZIP, insérez uniquement le code et ses dépendances, pas le dossier dans lequel il se trouve.

   - secret_test.py. Logique d'application.
   - greengrasssdk. Bibliothèque requise pour toutes les fonctions Lambda de Python Greengrass.

Il s'agit de votre package de déploiement de fonctions Lambda.

Étape 4 : Créer une fonction Lambda

Au cours de cette étape, vous allez utiliser la AWS Lambda console pour créer une fonction Lambda et la configurer pour utiliser votre package de déploiement. Vous publiez ensuite une version de fonction et créez un alias.

1. Créez d'abord la fonction Lambda.

   a. Dans AWS Management Console, choisissez Services et ouvrez la console AWS Lambda.

   b. Choisissez Créer une fonction, puis sélectionnez Auteur à partir de zéro.

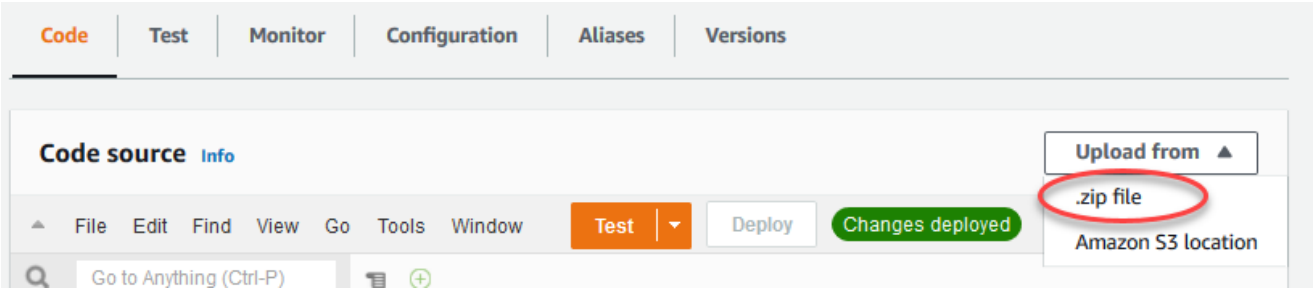
   c. Dans la section Informations de base, spécifiez les valeurs suivantes :

      - Sous Nom de la fonction, saisissez **`SecretTest`**.
      - Pour Runtime, sélectionnez Python 3.7.
      - Pour les autorisations, conservez le paramètre par défaut. Cela crée un rôle d'exécution qui accorde des autorisations Lambda de base. Ce rôle n'est pas utilisé parAWS IoT Greengrass.

   d. Dans le bas de la page, choisissez Create function.

2. Enregistrez ensuite le gestionnaire et téléchargez le package de déploiement de votre fonction Lambda.

   a. Dans l'onglet Code, sous Source du code, choisissez Télécharger depuis. Dans le menu déroulant, sélectionnez le fichier .zip.



   b. Choisissez Upload, puis choisissez votre package `secret_test_python.zip` de déploiement. Ensuite, choisissez Enregistrer.

c. Dans l'onglet Code de la fonction, sous Paramètres d'exécution, choisissez Modifier, puis entrez les valeurs suivantes.

- Pour Runtime, sélectionnez Python 3.7.
- Pour Handler (Gestionnaire), entrez **`secret_test.function_handler`**.

d. Choisissez Save (Enregistrer).

> **Note**
>
> Le bouton Test de la AWS Lambda console ne fonctionne pas avec cette fonction. Le SDK AWS IoT Greengrass principal ne contient pas les modules nécessaires pour exécuter vos fonctions Greengrass Lambda de manière indépendante dans la console. AWS Lambda Ces modules (par exemple`greengrass_common`) sont fournis aux fonctions après leur déploiement sur votre noyau Greengrass.

3. À présent, publiez la première version de votre fonction Lambda et créez un alias pour cette version.

> **Note**
>
> Les groupes Greengrass peuvent référencer une fonction Lambda par alias (recommandé) ou par version. L'utilisation d'un alias facilite la gestion des mises à jour du code, car vous n'avez pas à modifier votre table d'abonnement ou la définition de groupe lorsque le code de fonction est mis à jour. Au lieu de cela, il vous suffit de pointer l'alias vers la nouvelle version de la fonction.

a. Dans le menu Actions, sélectionnez Publier une nouvelle version.

b. Dans Description de la version, saisissez **`First version`**, puis choisissez Publish.

c. Sur la page de configuration SecretTest: 1, dans le menu Actions, choisissez Créer un alias.

d. Sur la page Create a new alias, utilisez les valeurs suivantes :

- Pour Name (Nom), saisissez **`GG_SecretTest`**.
- Pour Version, choisissez 1.

> **Note**
>
> AWS IoT Greengrassne prend pas en charge les alias Lambda pour les versions $LATEST.

e. Sélectionnez Create (Créer).

Vous êtes maintenant prêt à ajouter la fonction Lambda à votre groupe Greengrass et à joindre la ressource secrète.

Étape 5 : ajouter la fonction Lambda au groupe Greengrass

Au cours de cette étape, vous allez ajouter la fonction Lambda au groupe Greengrass dans la console. AWS IoT

1. Sur la page de configuration du groupe, choisissez l'onglet Fonctions Lambda.
2. Dans la section Mes fonctions Lambda, choisissez Ajouter.
3. Pour la fonction Lambda, choisissez. SecretTest
4. Pour la version de la fonction Lambda, choisissez l'alias de la version que vous avez publiée.

Configurez ensuite le cycle de vie de la fonction Lambda.

1. Dans la section Configuration de la fonction Lambda, effectuez les mises à jour suivantes.

> **Note**
>
> Nous vous recommandons d'exécuter votre fonction Lambda sans conteneurisation, sauf si votre analyse de rentabilisation l'exige. Cela permet d'accéder au processeur graphique et à la caméra de votre appareil sans configurer les ressources de l'appareil. Si vous exécutez sans conteneurisation, vous devez également accorder un accès root à vos fonctions LambdaAWS IoT Greengrass.

a. Pour exécuter sans conteneurisation :

- Pour Utilisateur et groupe du système, sélectionnez**`Another user ID/group ID`**. Dans le champ ID utilisateur du système, entrez**`0`**. Pour l'ID du groupe de systèmes, entrez**`0`**.

  Cela permet à votre fonction Lambda de s'exécuter en tant que root. Pour plus d'informations sur l'exécution en tant que root, consultezthe section called “Définition de l'identité d'accès par défaut pour les fonctions Lambda dans un groupe”.

  > **Tip**
  >
  > Vous devez également mettre à jour votre `config.json` fichier pour accorder un accès root à votre fonction Lambda. Pour la procédure, voirthe section called “Exécution d'une fonction Lambda en tant que root”.

- Pour la conteneurisation de la fonction Lambda, sélectionnez Aucun conteneur.

  Pour plus d'informations sur l'exécution sans conteneurisation, consultez. the section called “Considérations à prendre en compte lors du choix de la conteneurisation des fonctions Lambda”

- Pour Expiration, entrez **`10 seconds`**.
- Pour Épinglé, choisissez True.

  Pour plus d'informations, veuillez consulter the section called “Configuration du cycle de vie”.

- Sous Paramètre supplémentaire, pour l'accès en lecture au répertoire /sys, choisissez Enabled.

b. Pour exécuter plutôt en mode conteneurisé :

> **Note**
>
> Nous vous déconseillons de l'exécuter en mode conteneurisé, sauf si votre analyse de rentabilisation l'exige.

- Pour Utilisateur et groupe du système, choisissez Utiliser le groupe par défaut.
- Pour la conteneurisation de la fonction Lambda, choisissez Utiliser le groupe par défaut.

- Pour Limite de mémoire, entrez **`1024 MB`**.
- Pour Expiration, entrez **`10 seconds`**.
- Pour Épinglé, choisissez True.

  Pour plus d'informations, veuillez consulter the section called “Configuration du cycle de vie”.
- Sous Paramètres supplémentaires, pour l'accès en lecture au répertoire /sys, sélectionnez Activé.

2. Choisissez Ajouter une fonction Lambda.

Associez ensuite la ressource secrète à la fonction.

Étape 6 : associer la ressource secrète à la fonction Lambda

Au cours de cette étape, vous associez la ressource secrète à la fonction Lambda de votre groupe Greengrass. Cela associe la ressource à la fonction, ce qui permet à la fonction d'obtenir la valeur du secret local.

1. Sur la page de configuration du groupe, choisissez l'onglet Fonctions Lambda.
2. Choisissez la SecretTestfonction.
3. Sur la page de détails de la fonction, sélectionnez Ressources.
4. Accédez à la section Secrets et choisissez Associer.
5. Choisissez MyTestSecret, puis choisissez Associer.

Étape 7 : Ajouter des abonnements au groupe Greengrass

Au cours de cette étape, vous ajoutez des abonnements qui permettent à AWS IoT la fonction Lambda d'échanger des messages. Un abonnement permet à AWS IoT d'appeler la fonction et à la fonction d'envoyer des données de sortie vers AWS IoT.

1. Sur la page de configuration du groupe, choisissez l'onglet Abonnements, puis choisissez Ajouter un abonnement.
2. Créez un abonnement qui permet à AWS IoT de publier des messages à la fonction.

   Sur la page de configuration du groupe, choisissez l'onglet Abonnements, puis choisissez Ajouter un abonnement.

3. Sur la page Créer un abonnement, configurez la source et la cible comme suit :

   a. Dans Type de source, sélectionnez Fonction Lambda, puis IoT Cloud.

   b. Dans Type de cible, choisissez Service, puis choisissez SecretTest.

   c. Dans le filtre de rubrique, entrez**`secrets/input`**, puis choisissez Créer un abonnement.

4. Ajoutez un second abonnement. Choisissez l'onglet Abonnements, choisissez Ajouter un abonnement et configurez la source et la cible comme suit :

   a. Dans Type de source, choisissez Services, puis sélectionnez SecretTest.

   b. Dans Type de cible, choisissez la fonction Lambda, puis sélectionnez IoT Cloud.

   c. Dans le filtre de rubrique, entrez**`secrets/output`**, puis choisissez Créer un abonnement.

Étape 8 : Déployer le groupe Greengrass

Déployer le groupe sur l'appareil principal (noyau) Pendant le déploiementAWS IoT Greengrass, extrait la valeur du secret depuis Secrets Manager et crée une copie cryptée locale sur le noyau.

1. Assurez-vous que le AWS IoT Greengrass noyau fonctionne. Dans la fenêtre de terminal de votre Raspberry Pi, exécutez les commandes suivantes, si nécessaire.

   a. Pour vérifier si le démon est en cours d'exécution :

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

   Si la sortie contient une entrée `root` pour `/greengrass/ggc/packages/`*`ggc-version`*`/bin/daemon`, le démon est en cours d'exécution.

   > **Note**
   >
   > La version du chemin d'accès dépend de la version du logiciel AWS IoT Greengrass Core installée sur votre appareil principal.

   b. Pour démarrer le daemon :

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

2. Sur la page de configuration du groupe, choisissez Deploy.

3. a. Dans l'onglet Fonctions Lambda, sous la section Fonctions Lambda du système, sélectionnez Détecteur IP et choisissez Modifier.

   b. Dans la boîte de dialogue Modifier les paramètres du détecteur IP, sélectionnez Détecter et remplacer automatiquement les points de terminaison du broker MQTT.

   c. Choisissez Save (Enregistrer).

   Les appareils peuvent ainsi acquérir automatiquement des informations de connectivité pour le noyau, telles que l'adresse IP, le DNS et le numéro de port. La détection automatique est recommandée, mais AWS IoT Greengrass prend également en charge les points de terminaison spécifiés manuellement. Vous êtes uniquement invité à indiquer la méthode de découverte lors du déploiement initial du groupe.

   > **Note**
   >
   > Si vous y êtes invité, autorisez la création du rôle de service Greengrass et associez-le à votre rôle Compte AWS dans le service actuel. Région AWS Ce rôle permet d'accéder AWS IoT Greengrass à vos ressources dans les AWS services.

   La page Déploiements indique l'horodatage, l'ID de version et l'état du déploiement. Une fois terminé, le statut affiché pour le déploiement doit être Terminé.

   Pour bénéficier d'une aide à la résolution des problèmes, consultez Résolution des problèmes.

Test de la fonction Lambda

1. Sur la page d'accueil de la AWS IoT console, choisissez Test.
2. Pour S'abonner à la rubrique, utilisez les valeurs suivantes, puis choisissez S'abonner.

Propriété	Valeur
rubrique abonnement	secrets/sortie
Affichage de la charge utile MQTT	Affichage des charges utiles sous forme de chaînes

3. Pour Publier dans le sujet, utilisez les valeurs suivantes, puis choisissez Publier pour appeler la fonction.

Propriété	Valeur
Sujet	secrets/entrée
Message	Conserver le message par défaut. La publication d'un message appelle la fonction Lambda, mais la fonction décrite dans ce didacticiel ne traite pas le corps du message.

En cas de réussite, la fonction publie un message « Success ».

Consulter aussi

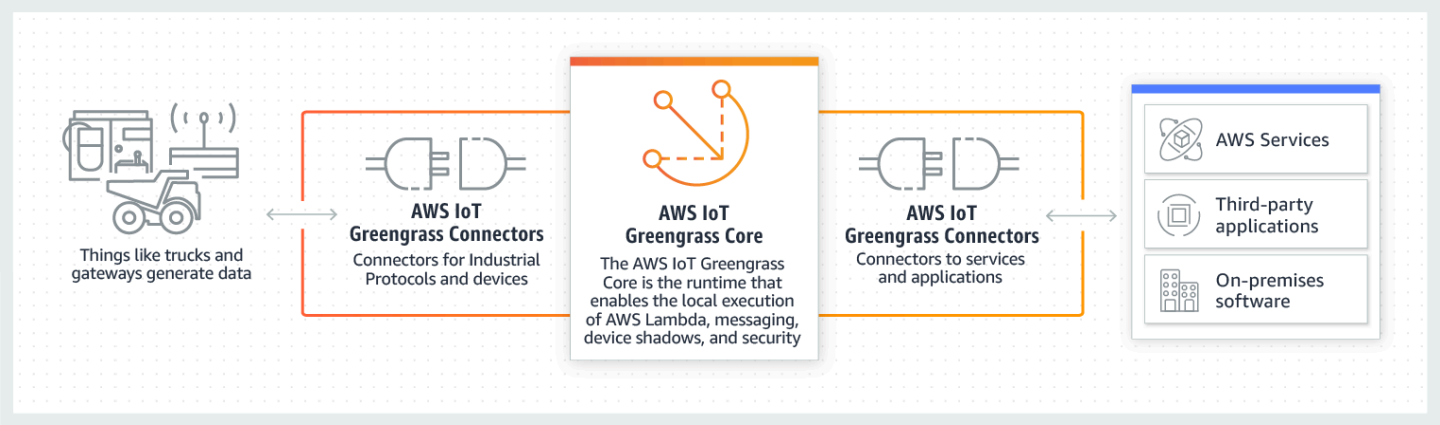
- Déployer des secrets sur Core

Intégrer à des services et protocoles à l'aide de connecteurs Greengrass

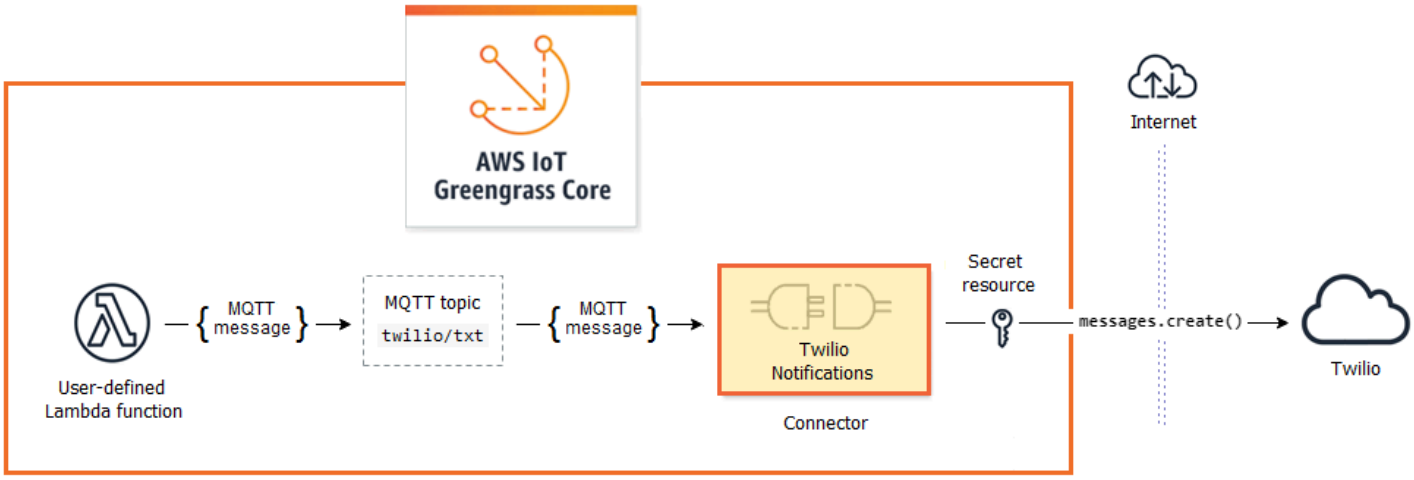
Cette fonction est disponible uniquement pour lesAWS IoT GreengrassCore v1.7 et version 1.7.

Connecteurs dansAWS IoT Greengrasssont des modules prédéfinis qui rendent plus efficace l'interaction avec l'infrastructure locale, les protocoles de périphérique,AWSet d'autres services cloud. En utilisant des connecteurs, vous pouvez consacrer moins de temps à apprendre des nouveaux protocoles et des API et consacrer plus de temps à lalogiquequi compte pour votre entreprise.

Le schéma suivant montre où les connecteurs peuvent s'intégrer dans le documentAWS IoT Greengrass paysage.



De nombreux connecteurs utilisent des messages MQTT pour communiquer avec les appareils clients et les fonctions Lambda Greengrass du groupe ou avecAWS IoTet le service shadow local. Dans l'exemple suivant, le connecteur Twilio Notifications reçoit des messages MQTT à partir d'une fonction Lambda définie par l'utilisateur, utilise une référence locale d'un secret à partir deAWS Secrets Manageret appelle l'API Twilio.



Pour accéder aux didacticiel permettant de créer cette solution, consultez the section called “Démarrer avec les connecteurs (console)” et the section called “Démarrer avec les connecteurs (CLI)”.

Les connecteurs Greengrass peuvent vous aider à développer des fonctionnalités de périphérique ou à créer des appareils à usage unique. À l'aide de connecteurs, vous pouvez :

- Mettre en œuvre une logique métier réutilisable.
- Interagir avec les services de cloud et locaux, y compris AWS et les services tiers.
- Intégrer et traiter des données d'appareil.
- Activer device-to-device les appels à des rubriques MQTT et les fonctions Lambda définies par l'utilisateur.

AWS fournit un ensemble d'exemples de connecteurs Greengrass qui simplifient les interactions avec les services courants et les sources de données. Ces modules réutilisables prédéfinis activent des scénarios pour la journalisation et les diagnostics, le réapprovisionnement, le traitement des données industrielles, l'alarme et la messagerie. Pour plus d'informations, consultez the section called “AWS-connecteurs Greengrass fournis”.

Prérequis

Pour utiliser des connecteurs, gardez ces points à l'esprit :

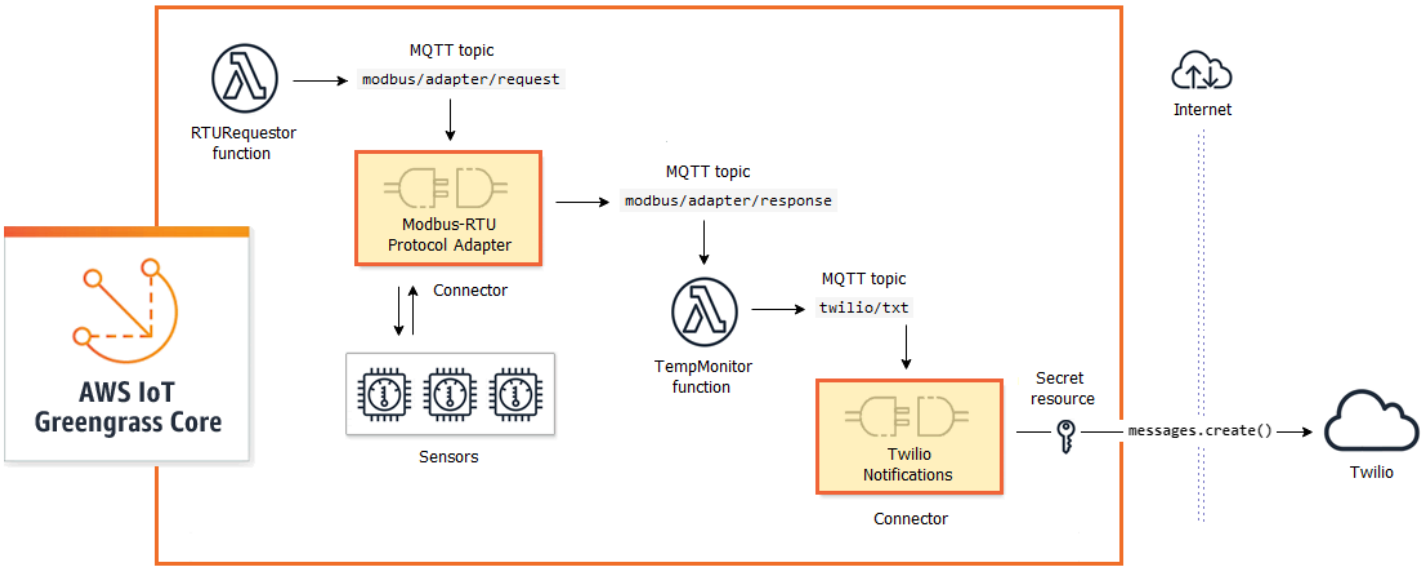
- Chaque connecteur que vous utilisez est soumis à des exigences que vous devez respecter. Ces exigences peuvent inclure la version minimale du logiciel AWS IoT Greengrass Core, les conditions préalables pour l'appareil, les autorisations requises et les limites. Pour plus d'informations, consultez the section called “AWS- connecteurs Greengrass fournis”.
- Un groupe Greengrass peut contenir seulement une seule instance configurée d'un connecteur donné. Cependant, vous pouvez utiliser l'instance dans plusieurs abonnements. Pour plus d'informations, consultez the section called “Paramètres de configuration”.
- Lorsque la conteneurisation par défaut du groupe Greengrass est définie sur No container (Aucun conteneur), les connecteurs du groupe doivent s'exécuter sans conteneurisation. Pour rechercher des connecteurs prenant en charge le mode No container (Aucun conteneur) veuillez consulter the section called “AWS- connecteurs Greengrass fournis”.

Utilisation des connecteurs Greengrass

Connecteur Aest un type de composant de groupe. Comme les autres composants de groupe, tels que les appareils clients et les fonctions Lambda définies par l'utilisateur, vous ajoutez des connecteurs à des groupes, vous configurez leurs paramètres et vous les déployez dans leAWS IoT Greengrasscore. Connecteurs utilisés dans l'environnement principal.

Vous pouvez déployer certains connecteurs en tant que simples applications autonomes. Par exemple, le connecteur Device Defender lit les métriques du système à partir de l'appareil principal et les envoie àAWS IoT Device Defenderpour analyse.

Vous pouvez ajouter d'autres connecteurs en tant que blocs de construction dans des solutions plus volumineuses. La solution de l'exemple suivant utilise le connecteur de l'adaptateur de protocole Modbus-RTU pour traiter les messages à partir de capteurs et le connecteur Twilio Notifications pour déclencher des messages Twilio.



Les solutions incluent souvent les fonctions Lambda définies par l'utilisateur qui restent en regard des connecteurs et traitent les données que le connecteur envoie ou reçoit. Dans cet exemple, le document TempMonitorreçoit des données de Modbus-RTU Protocol Adapter, exécute une logique métier, puis envoie les données à Twilio Notifications.

Pour créer et déployer une solution, vous suivez cette procédure générale :

1. Mappez les flux de données de haut niveau. Identifiez les sources de données, les canaux de données, les services, les protocoles et les ressources dont vous avez besoin. Dans l'exemple de solution, cela inclut les données sur le protocole Modbus RTU, le port série physique Modbus et Twilio.
2. Identifiez les connecteurs à inclure dans la solution, et ajoutez-les à votre groupe. L'exemple de solution utilise Modbus-RTU Protocol Adapter et Twilio Notifications. Pour vous aider à trouver les connecteurs qui s'appliquent à votre scénario, et pour en savoir plus sur leurs besoins individuels, consultez the section called “AWS- connecteurs Greengrass fournis”.
3. Déterminez si les fonctions Lambda définies par l'utilisateur, les périphériques clients ou les ressources sont nécessaires, puis créez-les et ajoutez-les au groupe. Cela peut inclure des fonctions qui contiennent une logique métier ou traitent des données dans un format requis par une autre entité dans la solution. L'exemple de solution utilise des fonctions pour envoyer des demandes RTU Modbus et lancer des notifications Twilio. Il inclut également une ressource d'appareil local pour le port série Modbus RTU et une ressource de secret pour le jeton d'authentification de Twilio.

> **Note**
>
> Les ressources de secret référencent des mots de passe, des jetons et d'autres secrets de AWS Secrets Manager. Les secrets peuvent être utilisés par les connecteurs et les fonctions Lambda pour s'authentifier auprès des services et des applications. Par défaut, AWS IoT Greengrass peut accéder aux secrets dont les noms commencent par « greengrass- ». Pour plus d'informations, consultez Déployer des secrets sur Core.

4. Créez des abonnements qui autorisent les entités de la solution à échanger des messages MQTT. Si un connecteur est utilisé dans un abonnement, le connecteur et la source ou la cible du message doivent utiliser la syntaxe de rubriques prédéfinie prise en charge par le connecteur. Pour plus d'informations, consultez the section called “Entrées et sorties”.
5. Déployer le groupe vers le noyau Greengrass.

Pour plus d'informations sur la création et le déploiement d'un connecteur, consultez les didacticiels suivants :

- the section called “Démarrer avec les connecteurs (console)”
- the section called “Démarrer avec les connecteurs (CLI)”

Paramètres de configuration

Beaucoup de connecteurs fournissent des paramètres qui vous permettent de personnaliser le comportement ou la sortie. Ces paramètres sont utilisés lors de l'initialisation, de l'exécution, ou à d'autres moments dans le cycle de vie du connecteur.

Les types de paramètres et leur utilisation varient en fonction du connecteur. Par exemple, le connecteur SNS dispose d'un paramètre qui configure la valeur par défaut de la rubrique SNS et Device Defender dispose d'un paramètre qui configure la fréquence de collecte des données.

Une version de groupe peut contenir plusieurs connecteurs, mais une seule instance d'un connecteur donné à la fois. Cela signifie que chaque connecteur du groupe ne peut avoir qu'une seule configuration active. Cependant, l'instance de connecteur peut être utilisée dans plusieurs abonnements dans le groupe. Par exemple, vous pouvez créer des abonnements qui autorisent de nombreux appareils à envoyer des données au connecteur Kinesis Firehose.

Paramètres utilisés pour accéder aux ressources d'un groupe

Les connecteurs Greengrass utilisent des ressources de groupe pour accéder au système de fichiers, aux ports, aux périphériques et aux autres ressources locales sur l'appareil principal. Si un connecteur nécessite l'accès à une ressource de groupe, il fournit alors des paramètres de configuration associés.

Les ressources de groupe comprennent :

- Des ressources locales. Répertoires, fichiers, ports, connecteurs et périphériques qui sont présents sur l'appareil principal Greengrass.
- Des ressources de Machine Learning. Modèles de machine Learning qui sont formés dans le cloud et déployés sur le noyau pour l'inférence locale.
- Des ressources de secret. Copies locales et chiffrées de mots de passe, clés, jetons d'accès ou texte arbitraire à partir de AWS Secrets Manager. Les connecteurs peuvent accéder en toute sécurité à ces secrets locaux et les utiliser pour s'authentifier auprès des services ou de l'infrastructure locale.

Par exemple, les paramètres de Device Defender autorisent l'accès aux métriques du système sur l'hôte.`/proc`et les paramètres des notifications Twilio de activent l'accès à un jeton d'authentification Twilio stocké localement.

Mise à jour des paramètres du connecteur

Les paramètres sont configurés lorsque le connecteur est ajouté à un groupe Greengrass. Vous pouvez modifier les valeurs des paramètres après l'ajout du connecteur.

- Dans la console : Sur la page de configuration de groupe, ouvrezConnecteurs, et dans le menu contextuel du connecteur, choisissezModifier.

  > **Note**
  >
  > Si le connecteur utilise une ressource de secret qui est ultérieurement modifiée pour référencer un autre secret, vous devez modifier les paramètres du connecteur et confirmer la modification.

- Dans l'API : Créez une autre version du connecteur qui définit la nouvelle configuration.

LeAWS IoT GreengrassL'API utilise des versions pour gérer les groupes. Les versions étant immuables, vous devez créer des versions de composants nouveaux ou mis à jour pour ajouter ou modifier des composants de groupetels que les appareils clients, les fonctions et les ressources du groupe. Ensuite, vous devez créer et déployer une version de groupe qui contient la version cible de chaque composant.

Lorsque vous apportez des modifications à la configuration du connecteur, vous devez déployer le groupe pour propager les modifications au noyau.

Entrées et sorties

De nombreux connecteurs Greengrass peuvent communiquer avec d'autres entités par l'envoi et la réception de messages MQTT. Cette communication est contrôlée par les abonnements qui permettent à un connecteur d'échanger des données avec des fonctions Lambda, des appareils clients et d'autres connecteurs du groupe Greengrass, ou avecAWS IoTet le service shadow local. Pour autoriser cette communication, vous devez créer des abonnements dans le groupe auquel le connecteur appartient. Pour plus d'informations, consultez the section called “Abonnements gérés dans le flux de travail de messagerie MQTT”.

Les connecteurs peuvent être des abonnés aux messages, des éditeurs de messages ou les deux. Chaque connecteur définit les rubriques MQTT qu'il publie ou auquel il s'abonne. Ces rubriques prédéfinies doivent être utilisées dans les abonnements où le connecteur est une source ou une cible de message. Pour les didacticiels qui incluent des étapes pour la configuration d'abonnements d'un connecteur, consultez the section called “Démarrer avec les connecteurs (console)” et the section called “Démarrer avec les connecteurs (CLI)”.

> **Note**
>
> De nombreux connecteurs intègrent également des modes de communication pour interagir avec les services de cloud ou locaux. Ils varient selon le connecteur et peuvent exiger que vous configuriez des paramètres ou que vous ajoutiez des autorisations au rôle de groupe. Pour plus d'informations sur les exigences de connecteur, consultez the section called “AWS-connecteurs Greengrass fournis”.

Rubriques d'entrée

La plupart des connecteurs reçoivent des données en entrée sur les rubriques MQTT. Certains connecteurs s'abonnent à plusieurs rubriques pour les données d'entrée. Par exemple, le connecteur Serial Stream prend en charge deux rubriques :

- `serial/+/read/#`
- `serial/+/write/#`

Pour ce connecteur, les demandes de lecture et écriture sont envoyées à la rubrique correspondante. Lorsque vous créez des abonnements, assurez-vous d'utiliser la rubrique qui correspond à votre implémentation.

Les caractères + et # des exemples précédents sont des caractères génériques. Ces caractères génériques autorisent les abonnés à recevoir des messages sur plusieurs rubriques et les éditeurs à personnaliser les rubriques dans lesquelles ils publient.

- Le caractère générique + peut apparaître n'importe où dans la hiérarchie des rubriques. Il peut être remplacé par un élément de la hiérarchie.

  Par exemple, pour la rubrique `sensor/+/input`, les messages peuvent être publiés dans les rubriques `sensor/id-123/input`, mais pas dans `sensor/group-a/id-123/input`.

- Le caractère générique # peut apparaître uniquement à la fin de la hiérarchie des rubriques. Il peut être remplacé par un zéro ou plusieurs éléments de hiérarchie.

  Par exemple, pour une rubrique `sensor/#`, les messages peuvent être publiés dans `sensor/`, `sensor/id-123` et `sensor/group-a/id-123` mais pas dans `sensor`.

Les caractères génériques sont valides uniquement lorsque vous vous abonnez à des rubriques. Les messages ne peuvent pas être publiés dans des rubriques qui contiennent des caractères génériques. Vérifiez la documentation du connecteur pour plus d'informations sur ses exigences de rubriques d'entrée ou de sortie. Pour plus d'informations, consultez the section called “AWS-connecteurs Greengrass fournis”.

Prise en charge de la conteneurisation

Par défaut, la plupart des connecteurs s'exécutent sur le noyau Greengrass dans un environnement d'exécution isolé géré par AWS IoT Greengrass. Ces environnements d'exécution, appelés conteneurs, assurent l'isolement entre les connecteurs et le système hôte, ce qui offre plus de sécurité pour l'hôte et le connecteur.

Cependant, cette conteneurisation Greengrass n'est pas prise en charge dans certains environnements, par exemple en cas d'exécution deAWS IoT Greengrassdans un conteneur Docker ou sur des noyaux Linux plus anciens sans cgroups. Dans ces environnements, les connecteurs doivent s'exécuter en mode Aucun conteneur. Pour rechercher des connecteurs prenant en charge le mode No container (Aucun conteneur) veuillez consulter the section called “AWS- connecteurs Greengrass fournis”. Certains connecteurs s'exécutent en mode natif et certains connecteurs vous permettent de définir le mode d'isolement.

Vous pouvez également définir le mode d'isolement sur Aucun conteneur dans les environnements prenant en charge la conteneurisation Greengrass, mais nous vous recommandons d'utiliser le mode conteneur Greengrass lorsque cela est possible.

> **Note**
>
> Le paramètre de conteneurisation par défaut pour le groupe Greengrass ne s'applique pas aux connecteurs.

Mise à niveau des versions du connecteur

Les fournisseurs de connecteurs peuvent publier de nouvelles versions d'un connecteur qui ajoutent des fonctions, corrigent des problèmes ou améliorent les performances. Pour de plus amples informations sur les versions disponibles et les modifications associées, veuillez consulter la documentation de chaque connecteur.

DansAWS IoT, vous pouvez rechercher de nouvelles versions pour les connecteurs de votre groupe Greengrass.

1. DansAWS IoTVolet de navigation de la console sousGérer, DéveloppezAppareils Greengrass, puisGroupes (V1).
2. UnderGroupes Greengrass, choisissez votre groupe.

3. Choisissez Connectors (Connecteurs) pour afficher les connecteurs du groupe.

   Si une nouvelle version est disponible pour un connecteur, un bouton Available (Disponible) apparaît dans la colonne Upgrade (Mise à niveau).

4. Pour mettre à niveau la version du connecteur :

   a. À partir de la page Connectors (Connecteurs), dans la colonne Upgrade (Mise à niveau), choisissez Available (Disponible). La page Upgrade connector (Mettre à niveau le connecteur) s'ouvre et affiche les paramètres actuels, le cas échéant.

      Choisissez la nouvelle version du connecteur, définissez les paramètres selon vos besoins, puis choisissez Upgrade (Mettre à niveau).

   b. Sur la page Subscriptions (Abonnements), ajoutez de nouveaux abonnements dans le groupe pour remplacer ceux qui utilisent le connecteur en tant que source ou cible. Ensuite, supprimez les anciens abonnements.

      Les abonnements font référence aux connecteurs par version, de sorte qu'ils deviennent non valides si vous modifiez la version du connecteur dans le groupe.

   c. Dans le menu Actions choisissez Deploy (Déployer) pour déployer vos modifications sur le noyau.

Pour mettre à niveau un connecteur à partir de l'API AWS IoT Greengrass, créez et déployez une version de groupe qui inclut le connecteur mis à jour et les abonnements. Utilisez le même processus que lorsque vous ajoutez un connecteur à un groupe. Pour obtenir des instructions détaillées qui vous montrent comment utiliser le documentAWS CLIpour configurer et déployer un exemple de connecteur Twilio Notifications, consultezthe section called “Démarrer avec les connecteurs (CLI)”.

Journalisation des connecteurs

Les connecteurs Greengrass contiennent des fonctions Lambda qui écrivent des événements et des erreurs dans les journaux Greengrass. En fonction des paramètres de votre groupe, les journaux sont écrits dans CloudWatch Les journaux, le système de fichiers local, ou les deux. Les journaux des connecteurs incluent l'ARN de la fonction correspondante. L'ARN dans l'exemple suivant constitue la réponse ARN du connecteur Kinesis Firehose :

```
arn:aws:lambda:aws-region:account-id:function:KinesisFirehoseClient:1
```

Par défaut, la configuration de la journalisation écrit des journaux de niveau information dans le système de fichiers à l'aide de la structure de répertoire suivante :

```
greengrass-root/ggc/var/log/user/region/aws/function-name.log
```

Pour plus d'informations sur la journalisation Greengrass, consultezthe section called “Surveillance avec les journaux AWS IoT Greengrass”.

AWS- connecteurs Greengrass fournis

AWS fournit les connecteurs suivants qui prennent en charge les AWS IoT Greengrass scénarios courants. Pour plus d'information sur le fonctionnement des connecteurs, consultez la documentation suivante :

- Intégrer à des services et protocoles à l'aide de connecteurs
- Démarrer avec les connecteurs (console) ou Démarrer avec les connecteurs (CLI)

Connecteur	Description	Runtimes Lambda pris en charge	Prend en charge le mode No container (Aucun conteneur)
CloudWatch Métriques	Publie des statistiques personnalisées sur Amazon CloudWatch.	• Python 3.8 * • Python 3.7 • Python 2.7	Oui
Défenseur de l'appareil	Envoie les métriques du système à AWS IoT Device Defender.	• Python 3.8 * • Python 3.7 • Python 2.7	Non

Connecteur	Description	Runtimes Lambda pris en charge	Prend en charge le mode No container (Aucun conteneur)
Déploiement de l'application Docker	Exécute un fichier Docker Compose pour démarrer une application Docker sur l'appareil principal (noyau).	• Python 3.8 • Python 3.7	Oui
IoT Analytics	Envoie les données des appareils et des capteurs à AWS IoT Analytics.	• Python 3.8 * • Python 3.7 • Python 2.7	Oui
Adaptateur de protocole IP Ethernet IoT	Collecte les données des appareils EtherNet/IP.	• Java 8	Oui
IoT SiteWise	Envoie les données d'appareils et de capteurs vers des propriétés de ressources dans AWS IoT SiteWise.	• Java 8	Oui
Kinesis Firehose	Envoie des données aux flux de livraison Amazon Data Firehose.	• Python 3.8 * • Python 3.7 • Python 2.7	Oui
Commentaires sur le ML	Publie l'entrée du modèle d'apprentissage automatique dans le cloud et la sortie dans une rubrique MQTT.	• Python 3.8 • Python 3.7	Non

Connecteur	Description	Runtimes Lambda pris en charge	Prend en charge le mode No container (Aucun conteneur)
Classification des images ML	Exécute un service d'inférence de classification de l'image locale. Ce connecteur fournit des versions pour plusieurs plateformes.	• Python 3.8 * • Python 3.7 • Python 2.7	Non
Détection d'objets ML	Exécute un service local d'inférence de détection d'objets. Ce connecteur fournit des versions pour plusieurs plateformes.	• Python 3.8 • Python 3.7	Non
Adaptateur de protocole Modbus-RTU	Envoie des requêtes aux appareils Modbus RTU.	• Python 3.8 * • Python 3.7 • Python 2.7	Non
Adaptateur de protocole Modbus-TCP	Collecte les données des appareils ModbusTCP.	• Java 8	Oui
Raspberry Pi GPIO	Contrôle les broches GPIO sur un appareil principal Raspberry Pi.	• Python 3.8 * • Python 3.7 • Python 2.7	Non
Stream en série	Lit et écrit dans un port série sur l'appareil principal.	• Python 3.8 * • Python 3.7 • Python 2.7	Non

Connecteur	Description	Runtimes Lambda pris en charge	Prend en charge le mode No container (Aucun conteneur)
ServiceNow MetricBase Intégration	Publie les statistiques des séries chronologiques sur ServiceNow MetricBase.	• Python 3.8 * • Python 3.7 • Python 2.7	Oui
SNS	Envoie des messages à une rubrique Amazon SNS.	• Python 3.8 * • Python 3.7 • Python 2.7	Oui
Intégration à Splunk	Publie des données dans Splunk HEC.	• Python 3.8 * • Python 3.7 • Python 2.7	Oui
Notifications Twilio	Lance un message texte ou vocal Twilio.	• Python 3.8 * • Python 3.7 • Python 2.7	Oui

* Pour utiliser les environnements d'exécution Python 3.8, vous devez créer un lien symbolique entre le dossier d'installation par défaut de Python 3.7 et les fichiers binaires Python 3.8 installés. Pour plus d'informations, consultez les exigences spécifiques au connecteur.

> **Note**
>
> Nous vous recommandons de mettre à jour les versions de connecteur de Python 2.7 vers Python 3.7. La prise en charge continue des connecteurs Python 2.7 dépend de la prise

en charge de l' AWS Lambda exécution. Pour plus d'informations, consultez la politique de support en matière d'exécution dans le guide du AWS Lambda développeur.

CloudWatch Connecteur Metrics

Le connecteur CloudWatch Metrics publie sur Amazon des métriques personnalisées provenant des appareils Greengrass. CloudWatch Le connecteur fournit une infrastructure centralisée pour publier CloudWatch des métriques, que vous pouvez utiliser pour surveiller et analyser l'environnement principal de Greengrass et agir en fonction des événements locaux. Pour plus d'informations, consultez la section Utilisation CloudWatch des métriques Amazon dans le guide de CloudWatch l'utilisateur Amazon.

Ce connecteur reçoit les données des métriques en tant que messages MQTT. Le connecteur regroupe les métriques qui se trouvent dans le même espace de noms et les publie à CloudWatch intervalles réguliers.

Ce connecteur est disponible dans les versions suivantes.

Version	ARN
5	`arn:aws:greengrass:region::/connectors/CloudWatchMetrics/versions/5`
4	`arn:aws:greengrass:region::/connectors/CloudWatchMetrics/versions/4`
3	`arn:aws:greengrass:region::/connectors/CloudWatchMetrics/versions/3`
2	`arn:aws:greengrass:region::/connectors/CloudWatchMetrics/versions/2`

Version	ARN
1	`arn:aws:greengrass:  region::/connectors/CloudWatchMetrics/versions/1`

Pour obtenir des informations sur les changements apportés aux versions, veuillez consulter le Journal des modifications.

Prérequis

Ce connecteur possède les critères suivants :

Version 3 - 5

- Logiciel AWS IoT Greengrass Core 1.9.3 ou version ultérieure.
- Python version 3.7 ou 3.8 installé sur le périphérique principal et ajouté à la variable d'environnement PATH.

  > **Note**
  >
  > Pour utiliser Python 3.8, exécutez la commande suivante pour créer un lien symbolique entre le dossier d'installation par défaut de Python 3.7 et les fichiers binaires Python 3.8 installés.
  >
  > ```
  > sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
  > ```
  >
  > Ceci configure votre appareil de sorte qu'il réponde aux exigence de Python pour AWS IoT Greengrass.

- Le rôle de groupe Greengrass est configuré pour autoriser l'`cloudwatch:PutMetricData`action, comme illustré dans l'exemple de politique AWS Identity and Access Management (IAM) suivant.

  ```
  {
      "Version": "2012-10-17",
      "Statement": [
          {
  ```

```
            "Sid": "Stmt1528133056761",
            "Action": [
                "cloudwatch:PutMetricData"
            ],
            "Effect": "Allow",
            "Resource": "*"
        }
    ]
}
```

Pour l'exigence de rôle de groupe, vous devez configurer le rôle de manière à accorder les autorisations requises et à vous assurer que le rôle a été ajouté au groupe. Pour plus d'informations, consultez the section called “Gérer le rôle de groupe (console)” ou the section called “Gérer le rôle de groupe (interface de ligne de commande)”.

Pour plus d'informations sur CloudWatch les autorisations, consultez la référence CloudWatch des autorisations Amazon dans le guide de l'utilisateur IAM.

Versions 1 - 2

- AWS IoT GreengrassLogiciel principal v1.7 ou version ultérieure.
- Python version 2.7 installé sur le périphérique principal et ajouté à la variable d'environnement PATH.
- Le rôle de groupe Greengrass est configuré pour autoriser l'`cloudwatch:PutMetricData`action, comme illustré dans l'exemple de politique AWS Identity and Access Management (IAM) suivant.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Stmt1528133056761",
            "Action": [
                "cloudwatch:PutMetricData"
            ],
            "Effect": "Allow",
            "Resource": "*"
        }
    ]
```

```
}
```

Pour l'exigence de rôle de groupe, vous devez configurer le rôle de manière à accorder les autorisations requises et à vous assurer que le rôle a été ajouté au groupe. Pour plus d'informations, consultez the section called “Gérer le rôle de groupe (console)” ou the section called “Gérer le rôle de groupe (interface de ligne de commande)”.

Pour plus d'informations sur CloudWatch les autorisations, consultez la référence CloudWatch des autorisations Amazon dans le guide de l'utilisateur IAM.

Paramètres du connecteur

Ce connecteur fournit les paramètres suivants :

Versions 4 - 5

`PublishInterval`

Le nombre maximum de secondes à attendre avant de publier les métriques par lot pour un espace de nom donné. La valeur maximale est 900. Pour configurer le connecteur afin qu'il publie les métriques au fur et à mesure qu'elles sont reçues (sans traitement par lot), spécifiez 0.

Le connecteur publie CloudWatch après avoir reçu 20 métriques dans le même espace de noms ou après l'intervalle spécifié.

> **Note**
>
> Le connecteur ne garantit pas l'ordre de publier des événements.

Nom d'affichage dans la AWS IoT console : Intervalle de publication

Nécessaire : `true`

Type : `string`

Valeurs valides : `0 - 900`

Modèle valide : `[0-9]|[1-9]\d|[1-9]\d\d|900`

`PublishRegion`

Le Région AWS vers lequel publier CloudWatch les statistiques. Cette valeur remplace la région de métriques Greengrass par défaut. Elle est obligatoire uniquement lors de la publication de métriques entre régions.

Nom d'affichage dans la AWS IoT console : Région de publication

Nécessaire : `false`

Type : `string`

Modèle valide : `^$|([a-z]{2}-[a-z]+-\d{1})`

`MemorySize`

La mémoire (en Ko) à allouer au connecteur.

Nom affiché dans la AWS IoT console : Taille de la mémoire

Nécessaire : `true`

Type : `string`

Modèle valide : `^[0-9]+$`

`MaxMetricsToRetain`

Nombre maximal de métriques sur tous les espaces de noms à enregistrer en mémoire avant qu'elles ne soient remplacées par de nouvelles métriques. La valeur minimale est de 2000.

Cette limite s'applique lorsqu'il n'y a aucune connexion à Internet et que le connecteur démarre pour mettre en tampon les métriques à publier ultérieurement. Lorsque le tampon est plein, les métriques les plus anciennes sont remplacées par de nouvelles métriques. Les métriques d'un espace de nom donné sont remplacées uniquement par des métriques du même espace de nom.

> **Note**
>
> Les métriques ne sont pas enregistrées si le processus hôte du connecteur est interrompu. Par exemple, cette interruption peut se produire pendant le déploiement du groupe ou lorsque le périphérique redémarre.

Nom affiché dans la AWS IoT console : nombre maximum de mesures à conserver

Nécessaire : `true`

Type : `string`

Modèle valide : `^([2-9]\d{3}|[1-9]\d{4,})$`

`IsolationMode`

Mode conteneurisation de ce connecteur. La valeur par défaut est `GreengrassContainer`, ce qui signifie que le connecteur s'exécute dans un environnement d'exécution isolé à l'intérieur du conteneur AWS IoT Greengrass.

> **Note**
>
> Le paramètre de conteneurisation par défaut pour le groupe ne s'applique pas aux connecteurs.

Nom affiché dans la AWS IoT console : mode d'isolation du conteneur

Nécessaire : `false`

Type : `string`

Valeurs valides : `GreengrassContainer` ou `NoContainer`

Modèle valide : `^NoContainer$|^GreengrassContainer$`

Versions 1 - 3

`PublishInterval`

Le nombre maximum de secondes à attendre avant de publier les métriques par lot pour un espace de nom donné. La valeur maximale est 900. Pour configurer le connecteur afin qu'il publie les métriques au fur et à mesure qu'elles sont reçues (sans traitement par lot), spécifiez 0.

Le connecteur publie CloudWatch après avoir reçu 20 métriques dans le même espace de noms ou après l'intervalle spécifié.

> **Note**
>
> Le connecteur ne garantit pas l'ordre de publier des événements.

Nom d'affichage dans la AWS IoT console : Intervalle de publication

Nécessaire : `true`

Type : `string`

Valeurs valides : `0 - 900`

Modèle valide : `[0-9]|[1-9]\d|[1-9]\d\d|900`

`PublishRegion`

Le Région AWS vers lequel publier CloudWatch les statistiques. Cette valeur remplace la région de métriques Greengrass par défaut. Elle est obligatoire uniquement lors de la publication de métriques entre régions.

Nom d'affichage dans la AWS IoT console : Région de publication

Nécessaire : `false`

Type : `string`

Modèle valide : `^$|([a-z]{2}-[a-z]+-\d{1})`

`MemorySize`

La mémoire (en Ko) à allouer au connecteur.

Nom affiché dans la AWS IoT console : Taille de la mémoire

Nécessaire : `true`

Type : `string`

Modèle valide : `^[0-9]+$`

`MaxMetricsToRetain`

Nombre maximal de métriques sur tous les espaces de noms à enregistrer en mémoire avant qu'elles ne soient remplacées par de nouvelles métriques. La valeur minimale est de 2000.

Cette limite s'applique lorsqu'il n'y a aucune connexion à Internet et que le connecteur démarre pour mettre en tampon les métriques à publier ultérieurement. Lorsque le tampon est plein, les métriques les plus anciennes sont remplacées par de nouvelles métriques. Les métriques d'un espace de nom donné sont remplacées uniquement par des métriques du même espace de nom.

> **Note**
>
> Les métriques ne sont pas enregistrées si le processus hôte du connecteur est interrompu. Par exemple, cette interruption peut se produire pendant le déploiement du groupe ou lorsque le périphérique redémarre.

Nom affiché dans la AWS IoT console : nombre maximum de mesures à conserver

Nécessaire : `true`

Type : `string`

Modèle valide : `^([2-9]\d{3}|[1-9]\d{4,})$`

Exemple de création de connecteur (AWS CLI)

La commande CLI suivante crée un `ConnectorDefinition` avec une version initiale contenant le connecteur 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"
            }
        }
```

```
    ]
}'
```

Dans la AWS IoT Greengrass console, vous pouvez ajouter un connecteur depuis la page Connecteurs du groupe. Pour plus d'informations, consultez the section called "Démarrer avec les connecteurs (console)".

Données d'entrée

Ce connecteur accepte les métriques relatives à un sujet MQTT et les publie sur. CloudWatch Les messages d'entrée doivent être au format JSON.

Filtre de rubrique dans l'abonnement

`cloudwatch/metric/put`

Propriétés des messages

`request`

Informations sur la métrique dans ce message.

L'objet de la demande contient les données de métrique à publier dans CloudWatch. Les valeurs métriques doivent répondre aux spécifications de l'`PutMetricData`API. Seules les propriétés `namespace`, `metricData.metricName` et `metricData.value` sont obligatoires.

Nécessaire : `true`

Type : `object` qui inclut les propriétés suivantes :

`namespace`

L'espace de noms défini par l'utilisateur pour les données métriques de cette demande. CloudWatch utilise des espaces de noms comme conteneurs pour les points de données métriques.

> **Note**
>
> Vous ne pouvez pas spécifier un espace de noms commençant par la chaîne `AWS/` réservée.

Nécessaire : `true`

Type : `string`

Modèle valide : `[^:].*`

`metricData`

Les données pour la métrique.

Nécessaire : `true`

Type : `object` qui inclut les propriétés suivantes :

`metricName`

Le nom de la métrique.

Nécessaire : `true`

Type : `string`

`dimensions`

Les dimensions qui sont associées à la métrique. Les dimensions fournissent des informations supplémentaires sur la métrique et ses données. Une métrique peut définir jusqu'à 10 dimensions.

Ce connecteur inclut automatiquement une dimension nomméecoreName, dont la valeur est le nom du noyau.

Nécessaire : `false`

Type : `array` d'objets de dimension qui incluent les propriétés suivantes :

`name`

Nom de la dimension.

Nécessaire : `false`

Type : `string`

`value`

Valeur de la dimension.

Nécessaire : `false`

Type : `string`

`timestamp`

Heure à laquelle les données métriques ont été reçues, exprimée en nombre de secondes écoulées depuis`Jan 1, 1970 00:00:00 UTC`. Si cette valeur n'est pas spécifiée, le connecteur utilise l'heure à laquelle il a reçu le message.

Nécessaire : `false`

Type : `timestamp`

> **Note**
>
> Si vous utilisez entre les versions 1 et 4 de ce connecteur, nous vous recommandons de récupérer l'horodatage séparément pour chaque métrique lorsque vous envoyez plusieurs métriques à partir d'une seule source. N'utilisez pas de variable pour enregistrer l'horodatage.

`value`

Valeur de la métrique.

> **Note**
>
> CloudWatch rejette les valeurs trop petites ou trop grandes. Ces valeurs doivent être dans la plage de `8.515920e-109` à `1.174271e+108` (Base 10) ou `2e-360` à `2e360` (Base 2). Les valeurs spéciales (par exemple, `NaN`, `+Infinity`, `-Infinity`) ne sont pas prises en charge.

Nécessaire : `true`

Type : `double`

`unit`

Unité de la métrique.

Nécessaire : `false`

Type : `string`

Valeurs valides : `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`

Limites

Toutes les limites imposées par l' CloudWatch `PutMetricData`API s'appliquent aux métriques lors de l'utilisation de ce connecteur. Les limites suivantes sont particulièrement importantes :

- Limite de 40 Ko pour la charge utile d'API
- 20 métriques par requête d'API
- 150 transactions par seconde (TPS) pour l'API `PutMetricData`

Pour plus d'informations, consultez CloudWatch les limites dans le guide de CloudWatch l'utilisateur Amazon.

Exemple d'entrée

```
{
   "request": {
       "namespace": "Greengrass",
       "metricData":
           {
               "metricName": "latency",
               "dimensions": [
                   {
                       "name": "hostname",
                       "value": "test_hostname"
                   }
               ],
               "timestamp": 1539027324,
               "value": 123.0,
               "unit": "Seconds"
            }
    }
```

```
}
```

Données de sortie

Ce connecteur publie des informations d'état sous forme de données de sortie dans une rubrique MQTT.

Filtre de rubrique dans l'abonnement

`cloudwatch/metric/put/status`

Exemple de sortie : réussite

La réponse inclut l'espace de noms des données métriques et le `RequestId` champ de la CloudWatch réponse.

```
{
   "response": {
        "cloudwatch_rid":"70573243-d723-11e8-b095-75ff2EXAMPLE",
        "namespace": "Greengrass",
        "status":"success"
    }
}
```

Exemple de sortie : échec

```
{
   "response" : {
        "namespace": "Greengrass",
        "error": "InvalidInputException",
        "error_message":"cw metric is invalid",
        "status":"fail"
   }
}
```

> **Note**
>
> Si le connecteur détecte une erreur réessayable (par exemple, des erreurs de connexion), il réessaie de publier dans le lot suivant.

Exemple d'utilisation

Suivez les étapes de haut niveau suivantes pour configurer un exemple de fonction Lambda en Python 3.7 que vous pouvez utiliser pour tester le connecteur.

> **Note**
>
> - Si vous utilisez d'autres environnements d'exécution Python, vous pouvez créer un lien symbolique entre Python3.x et Python 3.7.
> - Les rubriques Démarrer avec les connecteurs (console) et Démarrer avec les connecteurs (CLI) contiennent des étapes détaillées qui vous montrent comment configurer et déployer un exemple de connecteur Twilio Notifications.

1. Veillez à répondre aux conditions requises pour le connecteur.

   Pour l'exigence de rôle de groupe, vous devez configurer le rôle de manière à accorder les autorisations requises et à vous assurer que le rôle a été ajouté au groupe. Pour plus d'informations, consultez the section called “Gérer le rôle de groupe (console)” ou the section called “Gérer le rôle de groupe (interface de ligne de commande)”.

2. Créez et publiez une fonction Lambda qui envoie des données d'entrée au connecteur.

   Enregistrez l'exemple de code en tant que fichier PY. Téléchargez et décompressez le SDK AWS IoT Greengrass de base pour Python. Ensuite, créez un package zip contenant le fichier PY et le dossier `greengrasssdk` au niveau racine. Ce package zip est le package de déploiement vers lequel vous effectuez le téléchargementAWS Lambda.

   Après avoir créé la fonction Lambda de Python 3.7, publiez une version de la fonction et créez un alias.

3. Configurez votre groupe Greengrass.

   a. Ajoutez la fonction Lambda par son alias (recommandé). Configurez le cycle de vie Lambda comme étant de longue durée (ou dans `"Pinned": true` la CLI).

   b. Ajoutez le connecteur et configurez ses paramètres.

   c. Ajoutez des abonnements qui permettent au connecteur de recevoir des données d'entrée et d'envoyer des données de sortie sur des filtres de rubrique pris en charge.

- Définissez la fonction Lambda comme source, le connecteur comme cible et utilisez un filtre de rubrique d'entrée compatible.
- Définissez le connecteur en tant que source, AWS IoT Core en tant que cible et utilisez un filtre de rubrique de sortie pris en charge. Vous utilisez cet abonnement pour afficher les messages d'état dans la AWS IoT console.

4. Déployez le groupe.
5. Dans la AWS IoT console, sur la page Test, abonnez-vous à la rubrique des données de sortie pour consulter les messages d'état du connecteur. L'exemple de fonction Lambda a une longue durée de vie et commence à envoyer des messages immédiatement après le déploiement du groupe.

   Lorsque vous avez terminé les tests, vous pouvez définir le cycle de vie Lambda à la demande (ou `"Pinned": false` dans la CLI) et déployer le groupe. Cela empêche la fonction d'envoyer des messages.

Exemple

L'exemple suivant de fonction Lambda envoie un message d'entrée au connecteur.

```
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
```

Licences

Le connecteur CloudWatch Metrics inclut les logiciels/licences tiers suivants :

- AWS SDK for Python (Boto3)/Licence Apache 2.0
- botocore/Licence Apache 2.0
- dateutil/Licence PSF
- docutils/Licence BSD, licence GPL (General Public License) GNU, licence Python Software Foundation, domaine public
- jmespath/Licence MIT
- s3transfer/Licence Apache 2.0
- urllib3/Licence MIT

Ce connecteur est publié dans le cadre du contrat de licence logicielle Greengrass Core.

Journal des modifications

Le tableau suivant décrit les modifications apportées à chaque version du connecteur.

Version	Modifications
5	Correctif pour ajouter la prise en charge des horodatages dupliqués dans les données d'entrée.
4	Ajout du paramètre `IsolationMode` pour configurer le mode de conteneurisation du connecteur.
3	Mise à niveau de l'environnement d'exécution Lambda vers Python 3.7, ce qui modifie les exigences d'exécution.
2	Correctif pour réduire la journalisation excessive.
1	Première version.

Un groupe Greengrass ne peut contenir qu'une seule version du connecteur à la fois. Pour de plus amples informations sur la mise à niveau d'une version de connecteur, veuillez consulter the section called “Mise à niveau des versions du connecteur”.

Consultez aussi

- Intégrer à des services et protocoles à l'aide de connecteurs
- the section called “Démarrer avec les connecteurs (console)”
- the section called “Démarrer avec les connecteurs (CLI)”
- Utilisation des CloudWatch métriques Amazon dans le guide de CloudWatch l'utilisateur Amazon
- PutMetricDatadans le Amazon CloudWatch API Reference

Connecteur Device Defender

Le Device Defenderconnecteur Informe les administrateurs des modifications intervenues dans l'état d'un appareil Greengrass principal. Cela peut permettre d'identifier un comportement inhabituel qui pourrait indiquer un appareil compromis.

Ce connecteur lit les métriques du système à partir du répertoire `/proc` sur l'appareil principal, puis publie les métriques dans AWS IoT Device Defender. Pour plus d'informations sur les rapports de métriques, consultezSpécifications des métriques d'appareildans leAWS IoTManuel du développeur.

Ce connecteur possède les versions suivantes.

Version	ARN
3	`arn:aws:greengrass:  region::/connectors/DeviceDefender/versions/3`
2	`arn:aws:greengrass:  region::/connectors/DeviceDefender/versions/2`
1	`arn:aws:greengrass:  region::/connectors/DeviceDefender/versions/1`

Pour obtenir des informations sur les changements apportés aux versions, veuillez consulter le Journal des modifications.

Prérequis

Ce connecteur possède les critères suivants :

Version 3

- Logiciel AWS IoT Greengrass Core 1.9.3 ou version ultérieure.
- PythonVersion 3.7 ou 3.8 installée sur le périphérique principal et ajoutée à la variable d'environnement PATH.

> **Note**
>
> Pour utiliser Python 3.8, exécutez la commande suivante pour créer un lien symbolique à partir du dossier d'installation Python 3.7 par défaut vers les binaires Python 3.8 installés.

```
sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
```

Ceci configure votre appareil de sorte qu'il réponde aux exigence de Python pour AWS IoT Greengrass.

- AWS IoT Device Defender configuré pour utiliser la fonction Detect pour suivre les violations. Pour de plus amples informations, veuillez consulterDétecterdans leAWS IoTManuel du développeur.
- UNRessource de volume localedans le groupe Greengrass qui pointe vers le`/proc`répertoire. La ressource doit utiliser les propriétés suivantes :
  - Chemin source : `/proc`
  - Chemin de destination : `/host_proc` (ou une valeur qui correspond au modèle valide)
  - AutoAddGroupOwner : `true`
- La bibliothèque psutil installée sur le noyau Greengrass. La version 5.7.0 est la dernière version vérifiée pour fonctionner avec le connecteur.
- La bibliothèque cbor installée sur le noyau Greengrass. La version 1.0.0 est la dernière version vérifiée pour fonctionner avec le connecteur.

Versions 1 - 2

- AWS IoT GreengrassLogiciel Core v1.7 ou version ultérieure.
- PythonVersion 2.7 installée sur le périphérique principal et ajoutée à la variable d'environnement PATH.
- AWS IoT Device Defender configuré pour utiliser la fonction Detect pour suivre les violations. Pour de plus amples informations, veuillez consulterDétecterdans leAWS IoTManuel du développeur.
- UNRessource de volume localedans le groupe Greengrass qui pointe vers le`/proc`répertoire. La ressource doit utiliser les propriétés suivantes :
  - Chemin source : `/proc`
  - Chemin de destination : `/host_proc` (ou une valeur qui correspond au modèle valide)
  - AutoAddGroupOwner : `true`
- La bibliothèque psutil installée sur le noyau Greengrass.
- La bibliothèque cbor installée sur le noyau Greengrass.

Paramètres de connecteur

Ce connecteur fournit les paramètres suivants :

`SampleIntervalSeconds`

Le nombre de secondes entre chaque cycle de collecte et de présentation des métriques. La valeur minimale est de 300 secondes (5 minutes).

Nom d'affichage dans leAWS IoTConsole : Intervalle de présentation des métriques

Obligatoire`true`

Type: `string`

Modèle valide :`^[0-9]*(?:3[0-9][0-9]|[4-9][0-9]{2}|[1-9][0-9]{3,})$`

`ProcDestinationPath-ResourceId`

ID de la ressource de volume de `/proc`.

> **Note**
>
> Ce connecteur bénéficie d'un accès en lecture seule à la ressource.

Nom d'affichage dans leAWS IoTConsole : Ressource pour le répertoire /proc

Obligatoire`true`

Type: `string`

Modèle valide :`[a-zA-Z0-9_-]+`

`ProcDestinationPath`

Chemin de destination de la ressource de volume `/proc`.

Nom d'affichage dans leAWS IoTConsole : Chemin de destination de la ressource /proc

Obligatoire`true`

Type: `string`

Modèle valide :`\/[a-zA-Z0-9_-]+`

Exemple de création de connecteur (AWS CLI)

La commande suivante d'CLI crée un`ConnectorDefinition`avec une version initiale qui contient le connecteur 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 fonction Lambda de ce connecteur possède unlongue duréeCycle de vie.

DansAWS IoT Greengrass, vous pouvez ajouter un connecteur à partir duConnecteurs. Pour plus d'informations, consultez the section called “Démarrer avec les connecteurs (console)”.

Données d'entrée

Ce connecteur n'accepte pas les messages MQTT comme données d'entrée.

Données de sortie

Ce connecteur publie des métriques de sécurité dans AWS IoT Device Defender comme données de sortie.

Filtre de rubrique dans l'abonnement

```
$aws/things/+/defender/metrics/json
```

> **Note**
>
> Il s'agit de la syntaxe de rubrique que AWS IoT Device Defender attend. Le connecteur remplace le caractère générique + par le nom du périphérique (par exemple, `$aws/things/thing-name/defender/metrics/json`).

Exemple de sortie

Pour plus d'informations sur les rapports de métriques, consultezSpécifications des métriques d'appareildans leAWS IoTManuel du développeur.

```
{
    "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
            }
        }
    }
}
```

Licences

Ce connecteur est libéré sous leContrat de licence du logiciel Greengrass Core.

Journal des modifications

Le tableau suivante décrit les modifications intervenues dans chaque version du connecteur.

Version	Modifications
3	Mise à niveau de l'environnement d'exécution Lambda vers Python 3.7, ce qui modifie les exigences d'exécution.
2	Correctif pour réduire la journalisation excessive.
1	Première version.

Un groupe Greengrass ne peut contenir qu'une seule version du connecteur à la fois. Pour de plus amples informations sur la mise à niveau d'une version de connecteur, veuillez consulter the section called “Mise à niveau des versions du connecteur”.

Consulter aussi

- Intégrer à des services et protocoles à l'aide de connecteurs
- the section called “Démarrer avec les connecteurs (console)”
- the section called “Démarrer avec les connecteurs (CLI)”
- Device Defenderdans leAWS IoTManuel du développeur

Connecteur de déploiement d'applications Docker

Le connecteur de déploiement d'applications Greengrass Docker facilite l'exécution de vos images Docker sur un noyau. AWS IoT Greengrass Le connecteur utilise Docker Compose pour démarrer une application Docker multi-conteneur à partir d'un fichier `docker-compose.yml`. Plus précisément, le connecteur exécute des commandes `docker-compose` pour gérer les conteneurs Docker sur un appareil principal (noyau) unique. Pour de plus amples informations, veuillez consulter Overview of Docker Composer dans la documentation Docker. Le connecteur peut accéder aux images Docker stockées dans des registres de conteneurs Docker, tels qu'Amazon Elastic Container Registry (Amazon ECR), Docker Hub et des registres Docker sécurisés privés.

Après avoir déployé le groupe Greengrass, le connecteur extrait les dernières images et démarre les conteneurs Docker. Il exécute la `docker-compose up` commande `docker-compose pull` et. Le connecteur publie ensuite l'état de la commande dans une rubrique MQTT en sortie. Il enregistre

également des informations d'état concernant l'exécution des conteneurs Docker. Cela vous permet de surveiller les journaux de vos applications sur Amazon CloudWatch. Pour plus d'informations, consultez the section called "Surveillance avec les journaux AWS IoT Greengrass". Le connecteur démarre également les conteneurs Docker à chaque redémarrage du démon Greengrass. Le nombre de conteneurs Docker pouvant s'exécuter sur le noyau dépend de votre matériel.

Les conteneurs Docker s'exécutent en dehors du domaine Greengrass sur l'appareil principal (noyau), de sorte qu'ils ne peuvent pas accéder à la communication inter-processus (IPC) du noyau. Cependant, vous pouvez configurer certains canaux de communication avec des composants Greengrass, tels que les fonctions Lambda locales. Pour plus d'informations, consultez the section called "Communication avec les conteneurs Docker".

Vous pouvez utiliser le connecteur pour des scénarios tels que l'hébergement d'un serveur web ou d'un serveur MySQL sur votre appareil principal (noyau). Les services locaux de vos applications Docker peuvent communiquer entre eux, avec d'autres processus dans l'environnement local et avec des services cloud. Par exemple, vous pouvez exécuter un serveur Web sur le cœur qui envoie des demandes provenant de fonctions Lambda à un service Web dans le cloud.

Ce connecteur s'exécute en mode d'isolation No container (Aucun conteneur), de sorte que vous pouvez le déployer dans un groupe Greengrass qui s'exécute sans conteneurisation Greengrass.

Ce connecteur est disponible dans les versions suivantes.

Version	ARN
7	`arn:aws:greengrass:  region::/connectors/DockerApplicationDeployment/versions/7`
6	`arn:aws:greengrass:  region::/connectors/DockerApplicationDeployment/versions/6`
5	`arn:aws:greengrass:  region::/connectors/DockerApplicationDeployment/versions/5`

Version	ARN
4	`arn:aws:greengrass:region::/connectors/DockerApplicationDeployment/versions/4`
3	`arn:aws:greengrass:region::/connectors/DockerApplicationDeployment/versions/3`
2	`arn:aws:greengrass:region::/connectors/DockerApplicationDeployment/versions/2`
1	`arn:aws:greengrass:region::/connectors/DockerApplicationDeployment/versions/1`

Pour obtenir des informations sur les changements apportés aux versions, veuillez consulter le Journal des modifications.

Prérequis

Ce connecteur possède les critères suivants :

- AWS IoT GreengrassLogiciel de base v1.10 ou version ultérieure.

  > **Note**
  >
  > Ce connecteur n'est pas pris en charge sur OpenWrt les distributions.

- Python version 3.7 ou 3.8 installé sur le périphérique principal et ajouté à la variable d'environnement PATH.

  > **Note**
  >
  > Pour utiliser Python 3.8, exécutez la commande suivante pour créer un lien symbolique entre le dossier d'installation par défaut de Python 3.7 et les fichiers binaires Python 3.8 installés.

```
sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
```

Ceci configure votre appareil de sorte qu'il réponde aux exigence de Python pour AWS IoT Greengrass.

- Un minimum de 36 Mo de RAM sur le noyau Greengrass pour que le connecteur surveille les conteneurs Docker en cours d'exécution. La mémoire totale requise dépend du nombre de conteneurs Docker qui s'exécutent sur le noyau.
- Docker Engine 1.1.9.1 ou version ultérieure installé sur le noyau Greengrass. La version 19.0.3 est la dernière version vérifiée pour fonctionner avec le connecteur.

  L'exécutable `docker` doit se trouver dans le répertoire `/usr/local/bin` ou `/usr/bin`.

  Important

  Nous vous recommandons d'installer un magasin d'informations d'identification pour sécuriser les copies locales de vos informations d'identification Docker. Pour plus d'informations, consultez the section called “Remarque de sécurité”.

  Pour plus d'informations sur l'installation de Docker sur les distributions Amazon Linux, consultez les bases de Docker pour Amazon ECS dans le manuel Amazon Elastic Container Service Developer Guide.
- Docker Compose installé sur le noyau Greengrass. L'exécutable `docker-compose` doit se trouver dans le répertoire `/usr/local/bin` ou `/usr/bin`.

  Les versions suivantes de Docker Compose sont vérifiées pour fonctionner avec le connecteur.

Version du connecteur	Version vérifiée de Compose Docker
7	1.25.4
6	1.25.4
5	1.25.4
4	1.25.4

Version du connecteur	Version vérifiée de Compose Docker
3	1.25.4
2	1.25.1
1	1.24.1

- Un seul fichier Docker Compose (par exemple,`docker-compose.yml`), stocké dans Amazon Simple Storage Service (Amazon S3). Le format doit être compatible avec la version de Docker Compose installée sur le noyau. Vous devez tester le fichier avant de l'utiliser sur votre noyau. Si vous modifiez le fichier après avoir déployé le groupe Greengrass, vous devez redéployer le groupe pour mettre à jour votre copie locale sur le noyau.
- Un utilisateur Linux autorisé à appeler le démon Docker local et à écrire dans le répertoire qui stocke la copie locale de votre fichier Compose. Pour plus d'informations, consultez Configuration de l'utilisateur Docker sur le noyau.
- Rôle de groupe Greengrass configuré pour autoriser l'action `s3:GetObject` sur le compartiment S3 qui contient votre fichier Compose. Cette autorisation est illustrée dans l'exemple de politique IAM suivant.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowAccessToComposeFileS3Bucket",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:s3:::bucket-name/*"
        }
    ]
}
```

> **Note**
>
> Si votre compartiment S3 est activé pour la gestion des versions, le rôle doit être configuré pour autoriser également l'`s3:GetObjectVersion`action. Pour plus d'informations,

consultez la section Utilisation du versionnement dans le guide de l'utilisateur d'Amazon Simple Storage Service.

Pour l'exigence de rôle de groupe, vous devez configurer le rôle de manière à accorder les autorisations requises et à vous assurer que le rôle a été ajouté au groupe. Pour plus d'informations, consultez the section called "Gérer le rôle de groupe (console)" ou the section called "Gérer le rôle de groupe (interface de ligne de commande)".

- Si votre fichier Docker Compose fait référence à une image Docker stockée dans Amazon ECR, le rôle de groupe Greengrass est configuré pour autoriser ce qui suit :
  - `ecr:GetDownloadUrlForLayer`et `ecr:BatchGetImage` des actions sur vos référentiels Amazon ECR contenant les images Docker.
  - Action `ecr:GetAuthorizationToken` sur vos ressources.

  Les référentiels doivent se trouver dans le même Compte AWS emplacement Région AWS que le connecteur.

  Important

  Les autorisations associées au rôle de groupe peuvent être assumées par toutes les fonctions et connecteurs Lambda du groupe Greengrass. Pour plus d'informations, consultez the section called "Remarque de sécurité".

  Ces autorisations sont affichées dans l'exemple de stratégie suivant.

```
{
    "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": "*"
        }
    ]
}
```

Pour plus d'informations, consultez les exemples de politiques de dépôt Amazon ECR dans le guide de l'utilisateur Amazon ECR.

Pour l'exigence de rôle de groupe, vous devez configurer le rôle de manière à accorder les autorisations requises et à vous assurer que le rôle a été ajouté au groupe. Pour plus d'informations, consultez the section called “Gérer le rôle de groupe (console)” ou the section called “Gérer le rôle de groupe (interface de ligne de commande)”.

- Si votre fichier Docker Compose fait référence à une image Docker provenant d'AWS Marketplace, le connecteur requiert également le respect des conditions suivantes :
  - Vous devez être abonné aux produits de conteneur AWS Marketplace. Pour de plus amples informations, veuillez consulter Rechercher et s'abonner à des produits de conteneur dans le Guide des abonnés AWS Marketplace.
  - AWS IoT Greengrassdoit être configuré pour prendre en charge les secrets locaux, comme décrit dans la section Exigences relatives aux secrets. Le connecteur utilise cette fonctionnalité uniquement pour récupérer vos secretsAWS Secrets Manager, et non pour les stocker.
  - Vous devez créer un secret dans Secrets Manager pour chaque AWS Marketplace registre qui stocke une image Docker référencée dans votre fichier Compose. Pour plus d'informations, consultez the section called “Accès aux images Docker à partir de référentiels privés”.
- Si votre fichier Docker Compose fait référence à une image Docker provenant de référentiels privés dans des registres autres qu'Amazon ECR, tels que Docker Hub, le connecteur répond également aux exigences suivantes :
  - AWS IoT Greengrassdoit être configuré pour prendre en charge les secrets locaux, comme décrit dans la section Exigences relatives aux secrets. Le connecteur utilise cette fonctionnalité uniquement pour récupérer vos secretsAWS Secrets Manager, et non pour les stocker.
  - Vous devez créer un secret dans Secrets Manager pour chaque dépôt privé qui stocke une image Docker référencée dans votre fichier Compose. Pour plus d'informations, consultez the section called “Accès aux images Docker à partir de référentiels privés”.

- Le démon Docker doit être en cours d'exécution lorsque vous déployez un groupe Greengrass contenant ce connecteur.

Accès aux images Docker à partir de référentiels privés

Si vous utilisez des informations d'identification pour accéder à vos images Docker, vous devez autoriser le connecteur à y accéder. La façon dont vous procédez dépend de l'emplacement de l'image Docker.

Pour les images Docker stockées sur Amazon ECR, vous autorisez l'obtention de votre jeton d'autorisation dans le rôle du groupe Greengrass. Pour plus d'informations, consultez the section called “Prérequis”.

Pour les images Docker stockées dans d'autres référentiels privés ou registres, vous devez créer un secret dans AWS Secrets Manager pour stocker vos informations de connexion. Cela inclut les images Docker auxquelles vous vous êtes abonné dans AWS Marketplace. Créez un secret pour chaque référentiel. Si vous mettez à jour vos secrets dans Secrets Manager, les modifications se répercuteront sur le cœur la prochaine fois que vous déploierez le groupe.

> **Note**
>
> Secrets Manager est un service que vous pouvez utiliser pour stocker et gérer en toute sécurité vos informations d'identification, clés et autres secrets dans leAWS Cloud. Pour plus d'informations, consultez Présentation d'AWS Secrets Manager dans le Guide de l'utilisateur AWS Secrets Manager.

Chaque secret doit contenir les clés suivantes :

Clé	Valeur
`username`	Nom d'utilisateur utilisé pour accéder au référentiel ou au registre.
`password`	Mot de passe utilisé pour accéder au référentiel ou au registre.

Clé	Valeur
`registryUrl`	Point de terminaison du registre. Il doit correspondre à l'URL de registre correspon dante dans le fichier Compose.

> **Note**
>
> Pour permettre à AWS IoT Greengrass d'accéder à un secret par défaut, le nom du secret doit commencer par greengrass-. Sinon, c'est votre rôle de service Greengrass qui doit accorder l'accès. Pour plus d'informations, consultez the section called “Autoriser AWS IoT Greengrass à obtenir les valeurs secrètes”.

Pour obtenir les informations de connexion pour les images Docker à partir d'AWS Marketplace

1. Obtenez votre mot de passe pour les images Docker à AWS Marketplace l'aide de la `aws ecr get-login-password` commande. Pour plus d'informations, consultez la section get-login-password dans la référence des commandes AWS CLI.

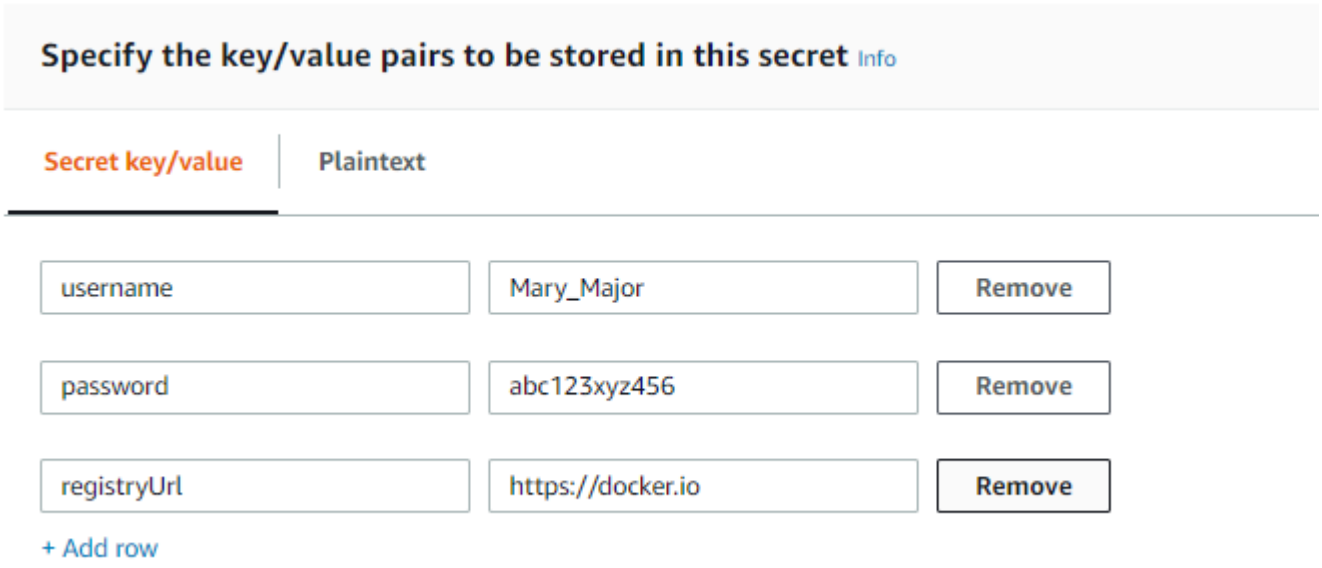
   ```
   aws ecr get-login-password
   ```

2. Récupérez l'URL de registre pour l'image Docker. Ouvrez le AWS Marketplace site Web et ouvrez la page de lancement du produit conteneur. Sous Images du conteneur, choisissez Afficher les détails de l'image du conteneur pour localiser le nom d'utilisateur et l'URL du registre.

Utilisez le nom d'utilisateur, le mot de passe et l'URL de registre récupérés pour créer un secret pour chaque AWS Marketplace registre qui stocke les images Docker référencées dans votre fichier Compose.

Pour créer des secrets (console)

Dans la console AWS Secrets Manager, choisissez Autre type de secrets. Sous Spécifiez les paires clé/valeur à stocker pour ce secret, ajoutez des lignes pour `username`, `password` et `registryUrl`. Pour plus d'informations, consultez la section Création d'un secret de base dans le guide de AWS Secrets Manager l'utilisateur.



Pour créer des secrets (interface de ligne de commande)

Dans leAWS CLI, utilisez la `create-secret` commande Secrets Manager, comme indiqué dans l'exemple suivant. Pour plus d'informations, voir create-secret dans la référence des AWS CLIcommandes.

```
aws secretsmanager create-secret --name greengrass-MySecret --secret-string
 [{"username":"Mary_Major"},{"password":"abc123xyz456"},{"registryUrl":"https://
docker.io"}]
```

> **Important**
>
> Il est de votre responsabilité de sécuriser le répertoire `DockerComposeFileDestinationPath` qui stocke votre fichier Docker Compose et les informations d'identification de vos images Docker provenant de référentiels privés. Pour plus d'informations, consultez the section called “Remarque de sécurité”.

Paramètres

Ce connecteur fournit les paramètres suivants :

Version 7

`DockerComposeFileS3Bucket`

Nom du compartiment S3 contenant votre fichier Docker Compose. Lorsque vous créez le bucket, assurez-vous de suivre les règles relatives aux noms de bucket décrites dans le guide de l'utilisateur d'Amazon Simple Storage Service.

Nom d'affichage dans la AWS IoT console : fichier Docker Compose dans S3

> **Note**
>
> Dans la console, la propriété Docker Compose file in S3 (Fichier Docker Compose dans S3) combine les paramètres `DockerComposeFileS3Version`, `DockerComposeFileS3Bucket` et `DockerComposeFileS3Key`.

Nécessaire : `true`

Type : `string`

Schéma valide `[a-zA-Z0-9\\-\\.]{3,63}`

`DockerComposeFileS3Key`

La clé d'objet de votre fichier Docker Compose dans Amazon S3. Pour plus d'informations, notamment les directives de dénomination des clés d'objet, consultez la section Clé d'objet et métadonnées dans le guide de l'utilisateur d'Amazon Simple Storage Service.

> **Note**
>
> Dans la console, la propriété Docker Compose file in S3 (Fichier Docker Compose dans S3) combine les paramètres `DockerComposeFileS3Version`, `DockerComposeFileS3Bucket` et `DockerComposeFileS3Key`.

Nécessaire : `true`

Type : `string`

Schéma valide `.+`

`DockerComposeFileS3Version`

Version de l'objet de votre fichier Docker Compose dans Amazon S3. Pour plus d'informations, notamment les directives de dénomination des clés d'objet, consultez la section Utilisation du versionnement dans le guide de l'utilisateur d'Amazon Simple Storage Service.

> **Note**
>
> Dans la console, la propriété Docker Compose file in S3 (Fichier Docker Compose dans S3) combine les paramètres `DockerComposeFileS3Version`, `DockerComposeFileS3Bucket` et `DockerComposeFileS3Key`.

Nécessaire : `false`

Type : `string`

Schéma valide `.+`

`DockerComposeFileDestinationPath`

Chemin absolu du répertoire local utilisé pour stocker une copie du fichier Docker Compose. Il doit s'agir d'un répertoire existant. L'utilisateur spécifié pour `DockerUserId` doit avoir l'autorisation de créer un fichier dans ce répertoire. Pour plus d'informations, consultez the section called “Configuration de l'utilisateur Docker sur le noyau”.

> **Important**
>
> Ce répertoire stocke votre fichier Docker Compose et les informations d'identification de vos images Docker provenant de référentiels privés. Il est de votre responsabilité de sécuriser ce répertoire. Pour plus d'informations, consultez the section called “Remarque de sécurité”.

Nom d'affichage dans la AWS IoT console : chemin du répertoire pour le fichier Compose local

Nécessaire : `true`

Type : `string`

Schéma valide `\/.*\/?`

Exemple : `/home/username/myCompose`

`DockerUserId`

UID de l'utilisateur Linux sous lequel le connecteur s'exécute. Cet utilisateur doit appartenir au groupe Linux `docker` sur l'appareil principal (noyau) et disposer d'autorisations d'écriture dans le répertoire `DockerComposeFileDestinationPath`. Pour plus d'informations, consultez Configuration de l'utilisateur Docker sur le noyau.

> **Note**
>
> Nous vous recommandons d'éviter l'exécution en tant que racine à moins que cela ne soit absolument nécessaire. Si vous spécifiez l'utilisateur root, vous devez autoriser les fonctions Lambda à s'exécuter en tant que root sur le AWS IoT Greengrass noyau. Pour plus d'informations, consultez the section called "Exécution d'une fonction Lambda en tant que root".

Nom affiché dans la AWS IoT console : ID utilisateur Docker

Nécessaire : `false`

Type : `string`

Modèle valide : `^[0-9]{1,5}$`

`AWSSecretsArnList`

ARN (Amazon Resource Names) des secrets dans AWS Secrets Manager qui contiennent les informations de connexion utilisées pour accéder à vos images Docker dans des référentiels privés. Pour plus d'informations, consultez the section called "Accès aux images Docker à partir de référentiels privés".

Nom affiché dans la AWS IoT console : Informations d'identification pour les référentiels privés

Obligatoire : `false`. Ce paramètre est requis pour accéder aux images Docker stockées dans des référentiels privés.

Type : `array` de `string`

Modèle valide : `[( ?,? ?"(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`

Fréquence (en secondes) à laquelle le connecteur enregistre les informations d'état sur les conteneurs Docker en cours d'exécution sur le noyau. La valeur par défaut est de 300 secondes (5 minutes).

Nom affiché dans la AWS IoT console : Fréquence de journalisation

Nécessaire : `false`

Type : `string`

Schéma valide : `^[1-9]{1}[0-9]{0,3}$`

`ForceDeploy`

Indique s'il faut forcer le déploiement de Docker en cas d'échec dû au nettoyage incorrect du dernier déploiement. La valeur par défaut est `False`.

Nom affiché dans la AWS IoT console : Forcer le déploiement

Nécessaire : `false`

Type : `string`

Schéma valide : `^(true|false)$`

`DockerPullBeforeUp`

Indique si le déployeur doit s'exécuter `docker-compose pull` avant de s'exécuter en raison `docker-compose up` d'un pull-down-up comportement. La valeur par défaut est `True`.

Nom affiché dans la AWS IoT console : Docker Pull Before Up

Nécessaire : `false`

Type : `string`

Schéma valide : `^(true|false)$`

`StopContainersOnNewDeployment`

Indique si le connecteur doit arrêter les conteneurs docker gérés par Docker Deployer lorsque GGC est arrêté (GGC s'arrête lorsqu'un nouveau groupe est déployé ou que le noyau est arrêté). La valeur par défaut est `True`.

Nom affiché dans la AWS IoT console : Docker s'arrête lors d'un nouveau déploiement

> **Note**
>
> Nous vous recommandons de conserver la `True` valeur par défaut de ce paramètre. Le paramètre to permet `False` à votre conteneur Docker de continuer à fonctionner même après avoir arrêté le AWS IoT Greengrass noyau ou lancé un nouveau déploiement. Si vous définissez ce paramètre sur`False`, vous devez vous assurer que vos conteneurs Docker sont maintenus si nécessaire en cas de modification ou d'ajout de nom de `docker-compose` service.
> Pour plus d'informations, consultez la documentation `docker-compose` du fichier de composition.

Nécessaire : `false`

Type : `string`

Schéma valide : `^(true|false)$`

`DockerOfflineMode`

Indique s'il faut utiliser le fichier Docker Compose existant lors du AWS IoT Greengrass démarrage hors ligne. La valeur par défaut est `False`.

Nécessaire : `false`

Type : `string`

Schéma valide : `^(true|false)$`

Version 6

`DockerComposeFileS3Bucket`

Nom du compartiment S3 contenant votre fichier Docker Compose. Lorsque vous créez le bucket, assurez-vous de suivre les règles relatives aux noms de bucket décrites dans le guide de l'utilisateur d'Amazon Simple Storage Service.

Nom d'affichage dans la AWS IoT console : fichier Docker Compose dans S3

> **Note**
>
> Dans la console, la propriété Docker Compose file in S3 (Fichier Docker Compose dans S3) combine les paramètres `DockerComposeFileS3Version`, `DockerComposeFileS3Bucket` et `DockerComposeFileS3Key`.

Nécessaire : `true`

Type : `string`

Schéma valide `[a-zA-Z0-9\\-\\.]{3,63}`

`DockerComposeFileS3Key`

La clé d'objet de votre fichier Docker Compose dans Amazon S3. Pour plus d'informations, notamment les directives de dénomination des clés d'objet, consultez la section Clé d'objet et métadonnées dans le guide de l'utilisateur d'Amazon Simple Storage Service.

> **Note**
>
> Dans la console, la propriété Docker Compose file in S3 (Fichier Docker Compose dans S3) combine les paramètres `DockerComposeFileS3Version`, `DockerComposeFileS3Bucket` et `DockerComposeFileS3Key`.

Nécessaire : `true`

Type : `string`

Schéma valide `.+`

`DockerComposeFileS3Version`

Version de l'objet de votre fichier Docker Compose dans Amazon S3. Pour plus d'informations, notamment les directives de dénomination des clés d'objet, consultez la section Utilisation du versionnement dans le guide de l'utilisateur d'Amazon Simple Storage Service.

> **Note**
>
> Dans la console, la propriété Docker Compose file in S3 (Fichier Docker Compose dans S3) combine les paramètres `DockerComposeFileS3Version`, `DockerComposeFileS3Bucket` et `DockerComposeFileS3Key`.

Nécessaire : `false`

Type : `string`

Schéma valide `.+`

`DockerComposeFileDestinationPath`

Chemin absolu du répertoire local utilisé pour stocker une copie du fichier Docker Compose. Il doit s'agir d'un répertoire existant. L'utilisateur spécifié pour `DockerUserId` doit avoir l'autorisation de créer un fichier dans ce répertoire. Pour plus d'informations, consultez the section called “Configuration de l'utilisateur Docker sur le noyau”.

> **Important**
>
> Ce répertoire stocke votre fichier Docker Compose et les informations d'identification de vos images Docker provenant de référentiels privés. Il est de votre responsabilité de sécuriser ce répertoire. Pour plus d'informations, consultez the section called “Remarque de sécurité”.

Nom d'affichage dans la AWS IoT console : chemin du répertoire pour le fichier Compose local

Nécessaire : `true`

Type : `string`

Schéma valide `\/.*\/?`

Exemple : `/home/username/myCompose`

`DockerUserId`

UID de l'utilisateur Linux sous lequel le connecteur s'exécute. Cet utilisateur doit appartenir au groupe Linux `docker` sur l'appareil principal (noyau) et disposer d'autorisations d'écriture dans

le répertoire `DockerComposeFileDestinationPath`. Pour plus d'informations, consultez Configuration de l'utilisateur Docker sur le noyau.

> **Note**
>
> Nous vous recommandons d'éviter l'exécution en tant que racine à moins que cela ne soit absolument nécessaire. Si vous spécifiez l'utilisateur root, vous devez autoriser les fonctions Lambda à s'exécuter en tant que root sur le AWS IoT Greengrass noyau. Pour plus d'informations, consultez the section called "Exécution d'une fonction Lambda en tant que root".

Nom affiché dans la AWS IoT console : ID utilisateur Docker

Nécessaire : `false`

Type : `string`

Schéma valide : `^[0-9]{1,5}$`

`AWSSecretsArnList`

ARN (Amazon Resource Names) des secrets dans AWS Secrets Manager qui contiennent les informations de connexion utilisées pour accéder à vos images Docker dans des référentiels privés. Pour plus d'informations, consultez the section called "Accès aux images Docker à partir de référentiels privés".

Nom affiché dans la AWS IoT console : Informations d'identification pour les référentiels privés

Obligatoire : `false`. Ce paramètre est requis pour accéder aux images Docker stockées dans des référentiels privés.

Type : `array` de `string`

Schéma valide : `[( ?,? ?"(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`

Fréquence (en secondes) à laquelle le connecteur enregistre les informations d'état sur les conteneurs Docker en cours d'exécution sur le noyau. La valeur par défaut est de 300 secondes (5 minutes).

Nom affiché dans la AWS IoT console : Fréquence de journalisation

Nécessaire : `false`

Type : `string`

Schéma valide : `^[1-9]{1}[0-9]{0,3}$`

`ForceDeploy`

Indique s'il faut forcer le déploiement de Docker en cas d'échec dû au nettoyage incorrect du dernier déploiement. La valeur par défaut est `False`.

Nom affiché dans la AWS IoT console : Forcer le déploiement

Nécessaire : `false`

Type : `string`

Schéma valide : `^(true|false)$`

`DockerPullBeforeUp`

Indique si le déployeur doit s'exécuter `docker-compose pull` avant de s'exécuter en raison `docker-compose up` d'un pull-down-up comportement. La valeur par défaut est `True`.

Nom affiché dans la AWS IoT console : Docker Pull Before Up

Nécessaire : `false`

Type : `string`

Schéma valide : `^(true|false)$`

`StopContainersOnNewDeployment`

Indique si le connecteur doit arrêter les conteneurs docker gérés par Docker Deployer lorsque GGC est arrêté (lorsqu'un nouveau déploiement de groupe est effectué ou que le noyau est arrêté). La valeur par défaut est `True`.

Nom affiché dans la AWS IoT console : Docker s'arrête lors d'un nouveau déploiement

> **Note**
>
> Nous vous recommandons de conserver la `True` valeur par défaut de ce paramètre. Le paramètre to permet `False` à votre conteneur Docker de continuer à fonctionner même après avoir arrêté le AWS IoT Greengrass noyau ou lancé un nouveau déploiement. Si vous définissez ce paramètre sur`False`, vous devez vous assurer que vos conteneurs Docker sont maintenus si nécessaire en cas de modification ou d'ajout de nom de `docker-compose` service.
> Pour plus d'informations, consultez la documentation `docker-compose` du fichier de composition.

Nécessaire : `false`

Type : `string`

Schéma valide : `^(true|false)$`

Version 5

`DockerComposeFileS3Bucket`

Nom du compartiment S3 contenant votre fichier Docker Compose. Lorsque vous créez le bucket, assurez-vous de suivre les règles relatives aux noms de bucket décrites dans le guide de l'utilisateur d'Amazon Simple Storage Service.

Nom d'affichage dans la AWS IoT console : fichier Docker Compose dans S3

> **Note**
>
> Dans la console, la propriété Docker Compose file in S3 (Fichier Docker Compose dans S3) combine les paramètres `DockerComposeFileS3Version`, `DockerComposeFileS3Bucket` et `DockerComposeFileS3Key`.

Nécessaire : `true`

Type : `string`

Schéma valide `[a-zA-Z0-9\\-\\.]{3,63}`

`DockerComposeFileS3Key`

La clé d'objet de votre fichier Docker Compose dans Amazon S3. Pour plus d'informations, notamment les directives de dénomination des clés d'objet, consultez la section Clé d'objet et métadonnées dans le guide de l'utilisateur d'Amazon Simple Storage Service.

> **Note**
>
> Dans la console, la propriété Docker Compose file in S3 (Fichier Docker Compose dans S3) combine les paramètres `DockerComposeFileS3Version`, `DockerComposeFileS3Bucket` et `DockerComposeFileS3Key`.

Nécessaire : `true`

Type : `string`

Schéma valide `.+`

`DockerComposeFileS3Version`

Version de l'objet de votre fichier Docker Compose dans Amazon S3. Pour plus d'informations, notamment les directives de dénomination des clés d'objet, consultez la section Utilisation du versionnement dans le guide de l'utilisateur d'Amazon Simple Storage Service.

> **Note**
>
> Dans la console, la propriété Docker Compose file in S3 (Fichier Docker Compose dans S3) combine les paramètres `DockerComposeFileS3Version`, `DockerComposeFileS3Bucket` et `DockerComposeFileS3Key`.

Nécessaire : `false`

Type : `string`

Schéma valide `.+`

`DockerComposeFileDestinationPath`

Chemin absolu du répertoire local utilisé pour stocker une copie du fichier Docker Compose. Il doit s'agir d'un répertoire existant. L'utilisateur spécifié pour `DockerUserId` doit avoir

l'autorisation de créer un fichier dans ce répertoire. Pour plus d'informations, consultez the section called "Configuration de l'utilisateur Docker sur le noyau".

> **Important**
>
> Ce répertoire stocke votre fichier Docker Compose et les informations d'identification de vos images Docker provenant de référentiels privés. Il est de votre responsabilité de sécuriser ce répertoire. Pour plus d'informations, consultez the section called "Remarque de sécurité".

Nom d'affichage dans la AWS IoT console : chemin du répertoire pour le fichier Compose local

Nécessaire : `true`

Type : `string`

Schéma valide `\/.*\/?`

Exemple : `/home/username/myCompose`

`DockerUserId`

UID de l'utilisateur Linux sous lequel le connecteur s'exécute. Cet utilisateur doit appartenir au groupe Linux `docker` sur l'appareil principal (noyau) et disposer d'autorisations d'écriture dans le répertoire `DockerComposeFileDestinationPath`. Pour plus d'informations, consultez Configuration de l'utilisateur Docker sur le noyau.

> **Note**
>
> Nous vous recommandons d'éviter l'exécution en tant que racine à moins que cela ne soit absolument nécessaire. Si vous spécifiez l'utilisateur root, vous devez autoriser les fonctions Lambda à s'exécuter en tant que root sur le AWS IoT Greengrass noyau. Pour plus d'informations, consultez the section called "Exécution d'une fonction Lambda en tant que root".

Nom affiché dans la AWS IoT console : ID utilisateur Docker

Nécessaire : `false`

Type : `string`

Schéma valide : `^[0-9]{1,5}$`

`AWSSecretsArnList`

ARN (Amazon Resource Names) des secrets dans AWS Secrets Manager qui contiennent les informations de connexion utilisées pour accéder à vos images Docker dans des référentiels privés. Pour plus d'informations, consultez the section called "Accès aux images Docker à partir de référentiels privés".

Nom affiché dans la AWS IoT console : Informations d'identification pour les référentiels privés

Obligatoire : `false`. Ce paramètre est requis pour accéder aux images Docker stockées dans des référentiels privés.

Type : `array` de `string`

Schéma valide : `[( ?,? ?"(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`

Fréquence (en secondes) à laquelle le connecteur enregistre les informations d'état sur les conteneurs Docker en cours d'exécution sur le noyau. La valeur par défaut est de 300 secondes (5 minutes).

Nom affiché dans la AWS IoT console : Fréquence de journalisation

Nécessaire : `false`

Type : `string`

Schéma valide : `^[1-9]{1}[0-9]{0,3}$`

`ForceDeploy`

Indique s'il faut forcer le déploiement de Docker en cas d'échec dû au nettoyage incorrect du dernier déploiement. La valeur par défaut est `False`.

Nom affiché dans la AWS IoT console : Forcer le déploiement

Nécessaire : `false`

Type : `string`

Schéma valide : `^(true|false)$`

`DockerPullBeforeUp`

Indique si le déployeur doit s'exécuter `docker-compose pull` avant de s'exécuter en raison `docker-compose up` d'un pull-down-up comportement. La valeur par défaut est `True`.

Nom affiché dans la AWS IoT console : Docker Pull Before Up

Nécessaire : `false`

Type : `string`

Schéma valide : `^(true|false)$`

Versions 2 - 4

`DockerComposeFileS3Bucket`

Nom du compartiment S3 contenant votre fichier Docker Compose. Lorsque vous créez le bucket, assurez-vous de suivre les règles relatives aux noms de bucket décrites dans le guide de l'utilisateur d'Amazon Simple Storage Service.

Nom d'affichage dans la AWS IoT console : fichier Docker Compose dans S3

> **Note**
>
> Dans la console, la propriété Docker Compose file in S3 (Fichier Docker Compose dans S3) combine les paramètres `DockerComposeFileS3Version`, `DockerComposeFileS3Bucket` et `DockerComposeFileS3Key`.

Nécessaire : `true`

Type : `string`

Schéma valide `[a-zA-Z0-9\\-\\.]{3,63}`

`DockerComposeFileS3Key`

La clé d'objet de votre fichier Docker Compose dans Amazon S3. Pour plus d'informations, notamment les directives de dénomination des clés d'objet, consultez la section Clé d'objet et métadonnées dans le guide de l'utilisateur d'Amazon Simple Storage Service.

> **Note**
>
> Dans la console, la propriété Docker Compose file in S3 (Fichier Docker Compose dans S3) combine les paramètres `DockerComposeFileS3Version`, `DockerComposeFileS3Bucket` et `DockerComposeFileS3Key`.

Nécessaire : `true`

Type : `string`

Schéma valide `.+`

`DockerComposeFileS3Version`

Version de l'objet de votre fichier Docker Compose dans Amazon S3. Pour plus d'informations, notamment les directives de dénomination des clés d'objet, consultez la section Utilisation du versionnement dans le guide de l'utilisateur d'Amazon Simple Storage Service.

> **Note**
>
> Dans la console, la propriété Docker Compose file in S3 (Fichier Docker Compose dans S3) combine les paramètres `DockerComposeFileS3Version`, `DockerComposeFileS3Bucket` et `DockerComposeFileS3Key`.

Nécessaire : `false`

Type : `string`

Schéma valide `.+`

`DockerComposeFileDestinationPath`

Chemin absolu du répertoire local utilisé pour stocker une copie du fichier Docker Compose. Il doit s'agir d'un répertoire existant. L'utilisateur spécifié pour `DockerUserId` doit avoir

l'autorisation de créer un fichier dans ce répertoire. Pour plus d'informations, consultez the section called "Configuration de l'utilisateur Docker sur le noyau".

> **Important**
>
> Ce répertoire stocke votre fichier Docker Compose et les informations d'identification de vos images Docker provenant de référentiels privés. Il est de votre responsabilité de sécuriser ce répertoire. Pour plus d'informations, consultez the section called "Remarque de sécurité".

Nom d'affichage dans la AWS IoT console : chemin du répertoire pour le fichier Compose local

Nécessaire : `true`

Type : `string`

Schéma valide `\/.*\/?`

Exemple : `/home/username/myCompose`

`DockerUserId`

UID de l'utilisateur Linux sous lequel le connecteur s'exécute. Cet utilisateur doit appartenir au groupe Linux `docker` sur l'appareil principal (noyau) et disposer d'autorisations d'écriture dans le répertoire `DockerComposeFileDestinationPath`. Pour plus d'informations, consultez Configuration de l'utilisateur Docker sur le noyau.

> **Note**
>
> Nous vous recommandons d'éviter l'exécution en tant que racine à moins que cela ne soit absolument nécessaire. Si vous spécifiez l'utilisateur root, vous devez autoriser les fonctions Lambda à s'exécuter en tant que root sur le AWS IoT Greengrass noyau. Pour plus d'informations, consultez the section called "Exécution d'une fonction Lambda en tant que root".

Nom affiché dans la AWS IoT console : ID utilisateur Docker

Nécessaire : `false`

Type : `string`

Schéma valide : `^[0-9]{1,5}$`

`AWSSecretsArnList`

ARN (Amazon Resource Names) des secrets dans AWS Secrets Manager qui contiennent les informations de connexion utilisées pour accéder à vos images Docker dans des référentiels privés. Pour plus d'informations, consultez the section called "Accès aux images Docker à partir de référentiels privés".

Nom affiché dans la AWS IoT console : Informations d'identification pour les référentiels privés

Obligatoire : `false`. Ce paramètre est requis pour accéder aux images Docker stockées dans des référentiels privés.

Type : `array` de `string`

Schéma valide : `[( ?,? ?"(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`

Fréquence (en secondes) à laquelle le connecteur enregistre les informations d'état sur les conteneurs Docker en cours d'exécution sur le noyau. La valeur par défaut est de 300 secondes (5 minutes).

Nom affiché dans la AWS IoT console : Fréquence de journalisation

Nécessaire : `false`

Type : `string`

Schéma valide : `^[1-9]{1}[0-9]{0,3}$`

`ForceDeploy`

Indique s'il faut forcer le déploiement de Docker en cas d'échec dû au nettoyage incorrect du dernier déploiement. La valeur par défaut est `False`.

Nom affiché dans la AWS IoT console : Forcer le déploiement

Nécessaire : `false`

Type : `string`

Schéma valide : `^(true|false)$`

Version 1

`DockerComposeFileS3Bucket`

Nom du compartiment S3 contenant votre fichier Docker Compose. Lorsque vous créez le bucket, assurez-vous de suivre les règles relatives aux noms de bucket décrites dans le guide de l'utilisateur d'Amazon Simple Storage Service.

Nom d'affichage dans la AWS IoT console : fichier Docker Compose dans S3

> **Note**
>
> Dans la console, la propriété Docker Compose file in S3 (Fichier Docker Compose dans S3) combine les paramètres `DockerComposeFileS3Version`, `DockerComposeFileS3Bucket` et `DockerComposeFileS3Key`.

Nécessaire : `true`

Type : `string`

Schéma valide `[a-zA-Z0-9\\-\\.]{3,63}`

`DockerComposeFileS3Key`

La clé d'objet de votre fichier Docker Compose dans Amazon S3. Pour plus d'informations, notamment les directives de dénomination des clés d'objet, consultez la section Clé d'objet et métadonnées dans le guide de l'utilisateur d'Amazon Simple Storage Service.

> **Note**
>
> Dans la console, la propriété Docker Compose file in S3 (Fichier Docker Compose dans S3) combine les paramètres `DockerComposeFileS3Version`, `DockerComposeFileS3Bucket` et `DockerComposeFileS3Key`.

Nécessaire : `true`

Type : `string`

Schéma valide `.+`

`DockerComposeFileS3Version`

Version de l'objet de votre fichier Docker Compose dans Amazon S3. Pour plus d'informations, notamment les directives de dénomination des clés d'objet, consultez la section Utilisation du versionnement dans le guide de l'utilisateur d'Amazon Simple Storage Service.

> **Note**
>
> Dans la console, la propriété Docker Compose file in S3 (Fichier Docker Compose dans S3) combine les paramètres `DockerComposeFileS3Version`, `DockerComposeFileS3Bucket` et `DockerComposeFileS3Key`.

Nécessaire : `false`

Type : `string`

Schéma valide `.+`

`DockerComposeFileDestinationPath`

Chemin absolu du répertoire local utilisé pour stocker une copie du fichier Docker Compose. Il doit s'agir d'un répertoire existant. L'utilisateur spécifié pour `DockerUserId` doit avoir l'autorisation de créer un fichier dans ce répertoire. Pour plus d'informations, consultez the section called “Configuration de l'utilisateur Docker sur le noyau”.

> **Important**
>
> Ce répertoire stocke votre fichier Docker Compose et les informations d'identification de vos images Docker provenant de référentiels privés. Il est de votre responsabilité de sécuriser ce répertoire. Pour plus d'informations, consultez the section called “Remarque de sécurité”.

Nom d'affichage dans la AWS IoT console : chemin du répertoire pour le fichier Compose local

Nécessaire : `true`

Type : `string`

Schéma valide `\/.*\/?`

Exemple : `/home/username/myCompose`

`DockerUserId`

UID de l'utilisateur Linux sous lequel le connecteur s'exécute. Cet utilisateur doit appartenir au groupe Linux `docker` sur l'appareil principal (noyau) et disposer d'autorisations d'écriture dans le répertoire `DockerComposeFileDestinationPath`. Pour plus d'informations, consultez Configuration de l'utilisateur Docker sur le noyau.

> **Note**
>
> Nous vous recommandons d'éviter l'exécution en tant que racine à moins que cela ne soit absolument nécessaire. Si vous spécifiez l'utilisateur root, vous devez autoriser les fonctions Lambda à s'exécuter en tant que root sur le AWS IoT Greengrass noyau. Pour plus d'informations, consultez the section called "Exécution d'une fonction Lambda en tant que root".

Nom affiché dans la AWS IoT console : ID utilisateur Docker

Nécessaire : `false`

Type : `string`

Schéma valide : `^[0-9]{1,5}$`

`AWSSecretsArnList`

ARN (Amazon Resource Names) des secrets dans AWS Secrets Manager qui contiennent les informations de connexion utilisées pour accéder à vos images Docker dans des référentiels privés. Pour plus d'informations, consultez the section called "Accès aux images Docker à partir de référentiels privés".

Nom affiché dans la AWS IoT console : Informations d'identification pour les référentiels privés

Obligatoire : `false`. Ce paramètre est requis pour accéder aux images Docker stockées dans des référentiels privés.

Type : `array` de `string`

Schéma valide : `[( ?,? ?"(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`

Fréquence (en secondes) à laquelle le connecteur enregistre les informations d'état sur les conteneurs Docker en cours d'exécution sur le noyau. La valeur par défaut est de 300 secondes (5 minutes).

Nom affiché dans la AWS IoT console : Fréquence de journalisation

Nécessaire : `false`

Type : `string`

Schéma valide : `^[1-9]{1}[0-9]{0,3}$`

Exemple de création de connecteur (AWS CLI)

La commande CLI suivante crée une `ConnectorDefinition` version initiale contenant le connecteur de déploiement d'applications 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": "myS3Bucket",
                "DockerComposeFileS3Key": "production-docker-compose.yml",
                "DockerComposeFileS3Version": "123",
                "DockerComposeFileDestinationPath": "/home/username/myCompose",
                "DockerUserId": "1000",
                "AWSSecretsArnList": "[\"arn:aws:secretsmanager:region:account-
id:secret:greengrass-secret1-hash\",\"arn:aws:secretsmanager:region:account-
id:secret:greengrass-secret2-hash\"]",
                "DockerContainerStatusLogFrequency": "30",
                "ForceDeploy": "True",
                "DockerPullBeforeUp": "True"
            }
        }
```

```
    ]
}'
```

> **Note**
>
> La fonction Lambda de ce connecteur a un cycle de vie prolongé.

Données d'entrée

Ce connecteur ne nécessite ni n'accepte les données d'entrée.

Données de sortie

Ce connecteur publie l'état de la commande `docker-compose up` sous forme de données de sortie.

Filtre de rubrique dans l'abonnement

`dockerapplicationdeploymentconnector/message/status`

Exemple de sortie : réussite

```
{
  "status":"success",
  "GreengrassDockerApplicationDeploymentStatus":"Successfully triggered docker-
compose up",
  "S3Bucket":"myS3Bucket",
  "ComposeFileName":"production-docker-compose.yml",
  "ComposeFileVersion":"123"
}
```

Exemple de sortie : échec

```
{
  "status":"fail",
  "error_message":"description of error",
  "error":"InvalidParameter"
}
```

Le type d'erreur peut être `InvalidParameter` ou `InternalError`.

Configuration de l'utilisateur Docker sur le noyau AWS IoT Greengrass

Le connecteur de déploiement de l'application Greengrass Docker s'exécute sous le nom d'utilisateur que vous spécifiez pour le paramètre. `DockerUserId` Si vous ne spécifiez pas de valeur, le connecteur s'exécute en tant que `ggc_user` (identité d'accès Greengrass par défaut).

Pour que le connecteur puisse interagir avec le démon Docker, l'utilisateur Docker doit appartenir au groupe Linux `docker` sur le noyau. L'utilisateur Docker doit également disposer d'autorisations d'écriture dans le répertoire `DockerComposeFileDestinationPath`. C'est à cet emplacement que le connecteur stocke votre fichier `docker-compose.yml` local et les informations d'identification Docker.

> **Note**
>
> - Nous vous recommandons de créer un utilisateur Linux au lieu d'utiliser la valeur par défaut `ggc_user`. Sinon, toute fonction Lambda du groupe Greengrass peut accéder au fichier Compose et aux informations d'identification Docker.
> - Nous vous recommandons d'éviter l'exécution en tant que racine à moins que cela ne soit absolument nécessaire. Si vous spécifiez l'utilisateur root, vous devez autoriser les fonctions Lambda à s'exécuter en tant que root sur le AWS IoT Greengrass noyau. Pour plus d'informations, consultez the section called “Exécution d'une fonction Lambda en tant que root”.

1. Créez l'utilisateur. Vous pouvez exécuter la commande `useradd` et inclure l'option facultative `-u` pour attribuer un UID. Par exemple :

   ```
   sudo useradd -u 1234 user-name
   ```

2. Ajoutez l'utilisateur au groupe `docker` sur le noyau. Par exemple :

   ```
   sudo usermod -aG docker user-name
   ```

   Pour de plus amples informations, notamment sur la création du groupe `docker`, veuillez consulter Manage Docker as a non-root user dans la documentation Docker.

3. Accordez à l'utilisateur les autorisations en écriture dans le répertoire spécifié pour le paramètre `DockerComposeFileDestinationPath`. Par exemple :

a. Pour définir l'utilisateur en tant que propriétaire du répertoire Cet exemple utilise l'UID de l'étape 1.

```
chown 1234 docker-compose-file-destination-path
```

b. Pour accorder les autorisations en lecture et en écriture au propriétaire

```
chmod 700 docker-compose-file-destination-path
```

Pour de plus amples informations, veuillez consulter How To Manage File And Folder Permissions In Linux dans la documentation Linux Foundation.

c. Si vous n'avez pas attribué d'UID lors de la création de l'utilisateur, ou si vous avez utilisé un utilisateur existant, exécutez la commande `id` pour rechercher l'UID.

```
id -u user-name
```

Vous utilisez l'UID pour configurer le paramètre `DockerUserId` pour le connecteur.

Informations d'utilisation

Lorsque vous utilisez le connecteur de déploiement d'applications Greengrass Docker, vous devez connaître les informations d'utilisation spécifiques à l'implémentation suivantes.

- Préfixe fixe pour les noms de projets. Le connecteur ajoute le préfixe `greengrassdockerapplicationdeployment` aux noms des conteneurs Docker qu'il démarre. Le connecteur utilise ce préfixe comme nom de projet dans les commandes `docker-compose` qu'il exécute.
- Comportement de journalisation. Le connecteur écrit des informations d'état et de dépannage dans un fichier journal. Vous pouvez configurer AWS IoT Greengrass pour envoyer des CloudWatch journaux à Logs et pour écrire des journaux localement. Pour plus d'informations, consultez the section called "Logging". Voici le chemin d'accès au journal local du connecteur :

  ```
  /greengrass-root/ggc/var/log/user/region/aws/DockerApplicationDeployment.log
  ```

  Vous devez disposer des autorisations root pour accéder aux journaux locaux.

- Mise à jour des images Docker. Docker met en cache les images sur l'appareil principal (noyau). Si vous mettez à jour une image Docker et que vous souhaitez propager la modification à l'appareil principal (noyau), assurez-vous de modifier la balise de l'image dans le fichier Compose. Les modifications prennent effet après le déploiement du groupe Greengrass.
- Délai d'attente de 10 minutes pour les opérations de nettoyage. Lorsque le daemon Greengrass s'arrête lors d'un redémarrage, la `docker-compose down` commande est lancée. Tous les conteneurs Docker disposent d'un maximum de 10 minutes après `docker-compose down` leur lancement pour effectuer toute opération de nettoyage. Si le nettoyage n'est pas terminé dans les 10 minutes, vous devez nettoyer les conteneurs restants manuellement. Pour de plus amples informations, veuillez consulter docker rm dans la documentation de l'interface de ligne de commande Docker.
- Exécution de commandes Docker. Pour résoudre les problèmes, vous pouvez exécuter des commandes Docker dans une fenêtre de terminal sur l'appareil principal (noyau). Par exemple, exécutez la commande suivante pour afficher les conteneurs Docker qui ont été démarrés par le connecteur :

  ```
  docker ps --filter name="greengrassdockerapplicationdeployment"
  ```

- ID de ressource réservée. Le connecteur utilise l'ID `DOCKER_DEPLOYER_SECRET_RESOURCE_RESERVED_ID_`*`index`* pour les ressources Greengrass qu'il crée dans le groupe Greengrass. Les ID de ressource doivent être uniques dans le groupe. Par conséquent, n'affectez pas un ID de ressource susceptible d'entrer en conflit avec cet ID de ressource réservé.
- Mode hors ligne. Lorsque vous définissez le paramètre de `DockerOfflineMode` configuration sur`True`, le connecteur Docker peut fonctionner en mode hors ligne. Cela peut se produire lorsqu'un déploiement de groupe Greengrass redémarre alors que l'appareil principal est hors ligne et que le connecteur ne peut pas établir de connexion avec Amazon S3 ou Amazon ECR pour récupérer le fichier Docker Compose.

  Lorsque le mode hors ligne est activé, le connecteur tente de télécharger votre fichier Compose et d'exécuter des `docker login` commandes comme il le ferait pour un redémarrage normal. Si ces tentatives échouent, le connecteur recherche un fichier Compose stocké localement dans le dossier spécifié à l'aide du `DockerComposeFileDestinationPath` paramètre. S'il existe un fichier Compose local, le connecteur suit la séquence normale de `docker-compose` commandes et extrait des images locales. Si le fichier Compose ou les images locales ne sont pas présents, le connecteur échoue. Le comportement des `StopContainersOnNewDeployment` paramètres `ForceDeploy` et reste le même en mode hors ligne.

Communication avec les conteneurs Docker

AWS IoT Greengrass prend en charge les canaux de communication suivants entre les composants Greengrass et les conteneurs Docker :

- Les fonctions Greengrass Lambda peuvent utiliser les API REST pour communiquer avec les processus dans les conteneurs Docker. Vous pouvez configurer un serveur dans un conteneur Docker qui ouvre un port. Les fonctions Lambda peuvent communiquer avec le conteneur sur ce port.
- Les processus dans les conteneurs Docker peuvent échanger des messages MQTT via le courtier de messages Greengrass local. Vous pouvez configurer le conteneur Docker en tant qu'appareil client dans le groupe Greengrass, puis créer des abonnements pour permettre au conteneur de communiquer avec les fonctions Lambda de Greengrass, les appareils clients et les autres connecteurs du groupe, AWS IoT ou avec le service parallèle local. Pour plus d'informations, consultez the section called “Configurer la communication MQTT avec les conteneurs Docker”.
- Les fonctions Greengrass Lambda peuvent mettre à jour un fichier partagé pour transmettre des informations aux conteneurs Docker. Vous pouvez utiliser le fichier Compose pour effectuer un montage lié du chemin d'accès du fichier partagé pour un conteneur Docker.

Configurer la communication MQTT avec les conteneurs Docker

Vous pouvez configurer un conteneur Docker en tant que périphérique client et l'ajouter à un groupe Greengrass. Ensuite, vous pouvez créer des abonnements permettant la communication MQTT entre le conteneur Docker et les composants Greengrass ou AWS IoT. Dans la procédure suivante, vous créez un abonnement qui permet à l'appareil du conteneur Docker de recevoir des messages de mise à jour shadow à partir du service shadow local. Vous pouvez suivre ce modèle pour créer d'autres abonnements.

> **Note**
>
> Cette procédure suppose que vous avez déjà créé un groupe Greengrass et un noyau Greengrass (v1.10 ou version ultérieure). Pour plus d'informations sur la création d'un groupe et d'un noyau Greengrass, consultez. Commencer avec AWS IoT Greengrass

Pour configurer un conteneur Docker en tant qu'appareil client et l'ajouter à un groupe Greengrass

1. Créez un dossier sur l'appareil principal pour stocker les certificats et les clés utilisés pour authentifier l'appareil Greengrass.

   Le chemin d'accès du fichier doit être monté sur le conteneur Docker que vous souhaitez démarrer. L'extrait suivant montre comment monter un chemin d'accès de fichier dans votre fichier Compose. Dans cet exemple, *path-to-device-certs*représente le dossier que vous avez créé à cette étape.

   ```
   version: '3.3'
   services:
     myService:
       image: user-name/repo:image-tag
       volumes:
         -  /path-to-device-certs/:/path-accessible-in-container
   ```

2. Dans le volet de navigation de la AWS IoT console, sous Gérer, développez les appareils Greengrass, puis choisissez Groups (V1).
3. Choisissez le groupe cible.
4. Sur la page de configuration du groupe, choisissez Appareils clients, puis sélectionnez Associer.
5. Dans le mode Associer un appareil client à ce groupe, choisissez Create new AWS IoT thing.

   La page Créer des objets s'ouvre dans un nouvel onglet.
6. Sur la page Créer des objets, choisissez Créer un objet unique, puis cliquez sur Suivant.
7. Sur la page Spécifier les propriétés de l'objet, entrez le nom de l'appareil, puis choisissez Suivant.
8. Sur la page Configurer le certificat de l'appareil, choisissez Next.
9. Sur la page Attacher des politiques au certificat, effectuez l'une des opérations suivantes :
   - Sélectionnez une politique existante qui accorde les autorisations requises par les appareils clients, puis choisissez Create thing.

     Un modal s'ouvre dans lequel vous pouvez télécharger les certificats et les clés que l'appareil utilise pour se connecter au AWS Cloud et au noyau.
   - Créez et joignez une nouvelle politique qui accorde des autorisations aux appareils clients. Procédez comme suit :

     a. Choisissez Créer une politique.

La page Créer une stratégie s'ouvre dans un nouvel onglet.

b. Sur la page Create policy (Créer une stratégie), procédez comme suit :

i. Dans Nom de la stratégie, entrez un nom qui décrit la stratégie, tel que**`GreengrassV1ClientDevicePolicy`**.

ii. Dans l'onglet Déclarations de politique, sous Document de stratégie, sélectionnez JSON.

iii. Entrez le document de politique suivant. Cette politique permet à l'appareil client de découvrir les cœurs de Greengrass et de communiquer sur tous les sujets MQTT. Pour plus d'informations sur la façon de restreindre l'accès à cette politique, consultezAuthentification et autorisation d'appareil pour 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. Choisissez Create (Créer) pour créer la politique.

c. Retournez à l'onglet du navigateur avec la page Joindre les politiques au certificat ouverte. Procédez comme suit :

   i. Dans la liste des politiques, sélectionnez la politique que vous avez créée, telle que GreengrassV1ClientDevicePolicy.

      Si vous ne voyez pas la politique, cliquez sur le bouton d'actualisation.

   ii. Choisissez Créer objet.

      Un modal s'ouvre dans lequel vous pouvez télécharger les certificats et les clés que l'appareil utilise pour se connecter au AWS Cloud et au noyau.

10. Dans le modal Télécharger les certificats et les clés, téléchargez les certificats de l'appareil.

> **Important**
>
> Avant de choisir OK, téléchargez les ressources de sécurité.

Procédez comme suit :

a. Pour le certificat de l'appareil, choisissez Télécharger pour télécharger le certificat de l'appareil.

b. Pour le fichier de clé publique, choisissez Télécharger pour télécharger la clé publique du certificat.

c. Pour Fichier de clé privée, choisissez Télécharger pour télécharger le fichier de clé privée pour le certificat.

d. Passez en revue l'authentification du serveur dans le guide du AWS IoT développeur et choisissez le certificat CA racine approprié. Nous vous recommandons d'utiliser les points de terminaison Amazon Trust Services (ATS) et les certificats CA racine ATS. Sous Certificats de l'autorité de certification racine, choisissez Télécharger pour un certificat de l'autorité de certification racine.

e. Sélectionnez Exécuté.

Notez l'ID de certificat commun aux noms de fichiers du certificat et des clés de l'appareil. Vous en aurez besoin ultérieurement.

11. Copiez les certificats et les clés dans le dossier que vous avez créé à l'étape 1.

Ensuite, créez un abonnement dans le groupe. Pour cet exemple, vous créez un abonnement qui permet à l'appareil du conteneur Docker de recevoir des messages MQTT à partir du service shadow local.

> **Note**
>
> La taille maximale d'un document shadow est de 8 Ko. Pour plus d'informations, consultez la section sur AWS IoTles quotas dans le guide du AWS IoT développeur.

Pour créer un abonnement permettant à l'appareil de conteneur Docker de recevoir des messages MQTT du service shadow local

1. Sur la page de configuration du groupe, choisissez l'onglet Abonnements, puis choisissez Ajouter un abonnement.
2. Sur la page Sélectionnez la source et la cible, configurez la source et la cible comme suit :
   a. Dans Sélectionner une source, choisissez Services, puis Service shadow local.
   b. Pour Sélectionnez une cible, choisissez Appareils, puis choisissez votre appareil.
   c. Choisissez Suivant.
   d. Sur la page Filtrer vos données à l'aide d'une rubrique, dans Filtrer par rubrique**$aws/things/*MyDockerDevice*/shadow/update/accepted**, sélectionnez, puis cliquez sur Suivant. *MyDockerDevice*Remplacez-le par le nom de l'appareil que vous avez créé précédemment.
   e. Choisissez Finish (Terminer).

Incluez l'extrait de code suivant dans l'image Docker que vous référencez dans votre fichier Compose. C'est le code de l'appareil Greengrass. Ajoutez également du code dans votre conteneur Docker pour démarrer l'appareil Greengrass à l'intérieur du conteneur. Il peut s'exécuter comme un processus distinct dans l'image ou dans un thread distinct.

```
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)
```

Remarque de sécurité

Lorsque vous utilisez le connecteur de déploiement d'applications Greengrass Docker, tenez compte des considérations de sécurité suivantes.

Stockage local du fichier Docker Compose

Le connecteur stocke une copie de votre fichier Compose dans le répertoire spécifié pour le paramètre `DockerComposeFileDestinationPath`.

Il est de votre responsabilité de sécuriser ce répertoire. Vous devez utiliser les autorisations du système de fichiers pour restreindre l'accès au répertoire.

Stockage local des informations d'identification Docker

Si vos images Docker sont stockées dans des référentiels privés, le connecteur stocke vos informations d'identification Docker dans le répertoire spécifié pour le paramètre `DockerComposeFileDestinationPath`.

Il est de votre responsabilité de sécuriser ces informations d'identification. Par exemple, vous devez utiliser l'assistant des informations d'identification (credential-helper) sur l'appareil principal (noyau) lorsque vous installez Docker Engine.

Installation de Docker Engine à partir d'une source fiable

Il est de votre responsabilité d'installer Docker Engine à partir d'une source fiable. Ce connecteur utilise le démon Docker sur l'appareil principal (noyau) pour accéder à vos ressources Docker et gérer les conteneurs Docker.

Portée des autorisations de rôle de groupe Greengrass

Les autorisations que vous ajoutez dans le rôle de groupe Greengrass peuvent être assumées par toutes les fonctions et connecteurs Lambda du groupe Greengrass. Ce connecteur nécessite l'accès à votre fichier Docker Compose stocké dans un compartiment S3. Cela nécessite également l'accès à votre jeton d'autorisation Amazon ECR si vos images Docker sont stockées dans un référentiel privé sur Amazon ECR.

Licences

Le connecteur de déploiement d'applications Greengrass Docker inclut les logiciels/licences tiers suivants :

- AWS SDK for Python (Boto3)/Licence Apache 2.0
- botocore/Licence Apache 2.0
- dateutil/Licence PSF
- docutils/Licence BSD, licence GPL (General Public License) GNU, licence Python Software Foundation, domaine public
- jmespath/Licence MIT
- s3transfer/Licence Apache 2.0
- urllib3/Licence MIT

Ce connecteur est publié dans le cadre du contrat de licence logicielle Greengrass Core.

Journal des modifications

Le tableau suivant décrit les modifications apportées à chaque version du connecteur.

Version	Modifications
7	Ajouté `DockerOfflineMode` pour utiliser un fichier Docker Compose existant lors d'un AWS IoT Greengrass démarrage hors ligne. Rétentatives implémentées pour la `docker login` commande. Support pour les UID 32 bits.
6	Ajouté `StopContainersOnNewDeployment` pour annuler le nettoyage des conteneurs lorsqu'un nouveau déploiement est effectué ou que GGC s'arrête. Mécanismes d'arrêt et de démarrage plus sûrs. Correction d'un bogue de validation YAML.

Version	Modifications
5	Les images sont extraites avant de courir`docker-compose down` .
4	Ajout d' pull-before-up un comportement pour mettre à jour les images Docker.
3	Correction d'un problème lié à la recherche des variables d'environnement.
2	Ajout du paramètre `ForceDeploy` .
1	Première version.

Un groupe Greengrass ne peut contenir qu'une seule version du connecteur à la fois. Pour de plus amples informations sur la mise à niveau d'une version de connecteur, veuillez consulter the section called “Mise à niveau des versions du connecteur”.

Consultez aussi

- Intégrer à des services et protocoles à l'aide de connecteurs
- the section called “Démarrer avec les connecteurs (console)”
- the section called “Démarrer avec les connecteurs (CLI)”

IoT Analytics

> **Warning**
>
> Ce connecteur est entré dans sa phase de durée de vie prolongée etAWS IoT Greengrass ne publiera pas de mises à jour fournissant des fonctionnalités, des améliorations aux fonctionnalités existantes, des correctifs de sécurité ou des corrections de bogues. Pour plus d'informations, veuillez consulter AWS IoT Greengrass Version 1politique de maintenance.

Le connecteur IoT Analytics envoie les données des appareils locaux àAWS IoT Analytics. Vous pouvez utiliser ce connecteur comme hub central pour collecter des données à partir des capteurs du périphérique principal Greengrass et des appareils clients connectés. Le connecteur envoie les données auxAWS IoT Analytics canaux de la zone actuelleCompte AWS et de la région.

Il peut envoyer des données vers un canal de destination par défaut et des canaux spécifiés dynamiquement.

> **Note**
>
> AWS IoT Analytics est un service entièrement géré qui vous permet de collecter, de stocker, de traiter et d'interroger des données IoT. Dans AWS IoT Analytics, les données peuvent être analysées et traitées. Par exemple, elles peuvent être utilisées pour former des modèles ML pour surveiller l'état des machines ou pour tester de nouvelles stratégies de modélisation. Pour plus d'informations, consultez Présentation d'AWS IoT Analytics dans le Guide de l'utilisateur AWS IoT Analytics.

Le connecteur accepte les données formatées et non formatées sur les rubriques d'entrée MQTT. Il prend en charge deux rubriques prédéfinies où le canal de destination est spécifié en ligne. Il peut également recevoir des messages sur des rubriques définies par le client qui sont configurées dans les abonnements. Cela peut être utilisé pour acheminer des messages provenant d'appareils clients qui publient sur des sujets fixes ou qui gèrent des données non structurées ou dépendantes d'une pile provenant de périphériques aux ressources limitées.

Ce connecteur utilise l'`BatchPutMessage`API pour envoyer des données (sous forme de chaîne codée en JSON ou en base64) au canal de destination. Le connecteur peut traiter les données brutes dans un format conforme aux exigences de l'API. Le connecteur met en tampon les messages d'entrée dans des files d'attente par canal et traite les lots de façon asynchrone. Il fournit des paramètres qui vous permettent de contrôler le comportement de traitement par lots et de mise en file d'attente, et de limiter la consommation de mémoire. Par exemple, vous pouvez configurer la taille maximale de la file d'attente, l'intervalle des lots, la taille de la mémoire et le nombre de canaux actifs.

Ce connecteur possède les versions suivantes.

Version	ARN
4	`arn:aws:greengrass:  region::/connectors/IoTAnalytics/versions/4`

Version	ARN
3	`arn:aws:greengrass:  region::/connectors/IoTAnalytics/versions/3`
2	`arn:aws:greengrass:  region::/connectors/IoTAnalytics/versions/2`
1	`arn:aws:greengrass:  region::/connectors/IoTAnalytics/versions/1`

Pour obtenir des informations sur les changements apportés aux versions, veuillez consulter le Journal des modifications.

Prérequis

Ce connecteur possède les critères suivants :

Version 3 - 4

- Logiciel AWS IoT Greengrass Core 1.9.3 ou version ultérieure.
- Python version 3.7 ou 3.8 installée sur le périphérique principal et ajoutée à la variable d'environnement PATH.

> **Note**
>
> Pour utiliser Python 3.8, exécutez la commande suivante pour créer un lien symbolique depuis le dossier d'installation par défaut de Python 3.7 vers les fichiers binaires Python 3.8 installés.
>
> ```
> sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
> ```
>
> Ceci configure votre appareil de sorte qu'il réponde aux exigence de Python pour AWS IoT Greengrass.

- Ce connecteur ne peut être utilisé que dans les régions Amazon Web Services où les deux AWS IoT Greengrasset AWS IoT Analyticssont pris en charge.
- Toutes les entités et flux de travail AWS IoT Analytics associés sont créés et configurés. Les entités comprennent les canaux, le pipeline, les banques de données et les ensembles de données. Pour plus d'informations, consultez les procédures AWS CLI ou console dans le Guide de l'utilisateur AWS IoT Analytics.

  > **Note**
  >
  > LesAWS IoT Analytics chaînes de destination doivent utiliser le même compte et se trouver dansRégion AWS le même connecteur.

- Le rôle de groupe Greengrass est configuré pour autoriser l'`iotanalytics:BatchPutMessage`action sur les canaux de destination, comme indiqué dans l'exemple de politique IAM suivant. Les chaînes doivent se trouver dans la actuelleCompte AWS et la région

  ```
  {
      "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"
              ]
          }
      ]
  }
  ```

  Pour l'exigence de rôle de groupe, vous devez configurer le rôle de manière à accorder les autorisations requises et à vous assurer que le rôle a été ajouté au groupe. Pour plus d'informations, consultez the section called “Gérer le rôle de groupe (console)” ou the section called “Gérer le rôle de groupe (interface de ligne de commande)”.

Versions 1 - 2

- AWS IoT GreengrassLogiciel de base v1.7 ou version ultérieure.
- Python version 2.7 installé sur le périphérique principal et ajouté à la variable d'environnement PATH.
- Ce connecteur ne peut être utilisé que dans les régions Amazon Web Services où les deux AWS IoT Greengrasset AWS IoT Analyticssont pris en charge.
- Toutes les entités et flux de travail AWS IoT Analytics associés sont créés et configurés. Les entités comprennent les canaux, le pipeline, les banques de données et les ensembles de données. Pour plus d'informations, consultez les procédures AWS CLI ou console dans le Guide de l'utilisateur AWS IoT Analytics.

  > **Note**
  >
  > LesAWS IoT Analytics chaînes de destination doivent utiliser le même compte et se trouver dansRégion AWS le même connecteur.

- Le rôle de groupe Greengrass est configuré pour autoriser l'`iotanalytics:BatchPutMessage`action sur les canaux de destination, comme indiqué dans l'exemple de politique IAM suivant. Les chaînes doivent se trouver dans la actuelleCompte AWS et la région

  ```
  {
      "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"
              ]
          }
      ]
  }
  ```

Pour l'exigence de rôle de groupe, vous devez configurer le rôle de manière à accorder les autorisations requises et à vous assurer que le rôle a été ajouté au groupe. Pour plus d'informations, consultez the section called “Gérer le rôle de groupe (console)” ou the section called “Gérer le rôle de groupe (interface de ligne de commande)”.

Paramètres

`MemorySize`

Quantité de mémoire (en Ko) allouée à ce connecteur.

Nom d'affichage dans laAWS IoT console : Taille de la mémoire

Obligatoire :`true`

Type: `string`

Schéma valide :`^[0-9]+$`

`PublishRegion`

Celui dansRégion AWS lequel vosAWS IoT Analytics chaînes sont créées. Utilisez la même région que le connecteur.

> **Note**
>
> Cela doit également correspondre à la région pour les canaux qui sont spécifiés dans le rôle de groupe.

Nom d'affichage dans laAWS IoT console : Région de publication

Obligatoire :`false`

Type: `string`

Schéma valide :`^$|([a-z]{2}-[a-z]+-\\d{1})`

`PublishInterval`

Intervalle (en secondes) pour publier un lot de données reçues dans AWS IoT Analytics.

Nom d'affichage dans laAWS IoT console : Intervalle de publication

Obligatoire :`false`

Type: `string`

Valeur par défaut : `1`

Schéma valide :`$|^[0-9]+$`

`IotAnalyticsMaxActiveChannels`

Nombre maximal de canaux AWS IoT Analytics surveillés activement par le connecteur. Ce nombre doit être supérieur à 0 et au moins égal au nombre de canaux que le connecteur publiera à un moment donné.

Vous pouvez utiliser ce paramètre pour restreindre la consommation de mémoire en limitant le nombre total de files d'attente que le connecteur peut gérer à un moment donné. Une file d'attente est supprimée lorsque tous les messages en file d'attente sont envoyés.

Nom d'affichage dans laAWS IoT console : Nombre maximum de chaînes actives

Obligatoire :`false`

Type: `string`

Valeur par défaut : `50`

Schéma valide :`^$|^[1-9][0-9]*$`

`IotAnalyticsQueueDropBehavior`

Comportement pour supprimer des messages d'une file d'attente de canal lorsque la file d'attente est pleine.

Nom d'affichage dans laAWS IoT console : comportement de suppression de la file d'attente

Obligatoire :`false`

Type: `string`

Valeurs valides : `DROP_NEWEST` ou `DROP_OLDEST`

Valeur par défaut : `DROP_NEWEST`

Schéma valide :`^DROP_NEWEST$|^DROP_OLDEST$`

`IotAnalyticsQueueSizePerChannel`

Nombre maximal de messages à conserver en mémoire (par canal) avant que les messages soient soumis ou abandonnés. Ce nombre doit être supérieur à 0.

Nom d'affichage dans laAWS IoT console : Taille de file d'attente maximale par canal

Obligatoire :`false`

Type: `string`

Valeur par défaut : `2048`

Schéma valide :`^$|^[1-9][0-9]*$`

`IotAnalyticsBatchSizePerChannel`

Nombre maximal de messages à envoyer à un canal AWS IoT Analytics dans une demande par lot. Ce nombre doit être supérieur à 0.

Nom affiché dans laAWS IoT console : nombre maximum de messages à regrouper par canal

Obligatoire :`false`

Type: `string`

Valeur par défaut : `5`

Schéma valide :`^$|^[1-9][0-9]*$`

`IotAnalyticsDefaultChannelName`

Nom du canal AWS IoT Analytics que ce connecteur utilise pour les messages qui sont envoyés à une rubrique d'entrée définie par le client.

Nom d'affichage dans laAWS IoT console : nom de la chaîne par défaut

Obligatoire :`false`

Type: `string`

Schéma valide :`^[a-zA-Z0-9_]$`

`IsolationMode`

Mode conteneurisation de ce connecteur. La valeur par défaut est `GreengrassContainer`, ce qui signifie que le connecteur s'exécute dans un environnement d'exécution isolé à l'intérieur du conteneur AWS IoT Greengrass.

> **Note**
>
> Le paramètre de conteneurisation par défaut pour le groupe ne s'applique pas aux connecteurs.

Nom d'affichage dans laAWS IoT console : mode d'isolation du conteneur

Obligatoire :`false`

Type: `string`

Valeurs valides : `GreengrassContainer` ou `NoContainer`

Schéma valide :`^NoContainer$|^GreengrassContainer$`

Exemple de création de connecteur (AWS CLI)

La commande CLI suivante crée un`ConnectorDefinition` avec une version initiale contenant le connecteur 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 fonction Lambda de ce connecteur a une longue durée de vie.

Dans laAWS IoT Greengrass console, vous pouvez ajouter un connecteur depuis la page Connecteurs du groupe. Pour plus d'informations, veuillez consulter the section called “Démarrer avec les connecteurs (console)”.

Données d'entrée

Ce connecteur accepte les données sur des rubriques MQTT prédéfinies et définies par le client. Les éditeurs peuvent être des appareils clients, des fonctions Lambda ou d'autres connecteurs.

Rubriques prédéfinies

Le connecteur prend en charge deux rubriques MQTT structurées qui permettent aux éditeurs de spécifier le nom du canal en ligne.

- Un message formaté sur la rubrique `iotanalytics/channels/+/messages/put`. Les données IoT dans ces messages d'entrée doivent se présenter sous la forme d'un fichier JSON ou d'une chaîne encodée en base64.
- Un message non formaté sur la rubrique `iotanalytics/channels/+/messages/binary/put`. Les messages d'entrée reçus sur cette rubrique sont traités comme des données binaires et peuvent contenir n'importe quel type de données.

  Pour publier dans des rubriques prédéfinies, remplacez le caractère générique + par le nom du canal. Par exemple :

  ```
  iotanalytics/channels/my_channel/messages/put
  ```

Rubriques définies par le client

Le connecteur prend en charge la syntaxe de rubrique #, ce qui lui permet d'accepter les messages d'entrée sur n'importe quelle rubrique MQTT que vous configurez dans un

abonnement. Nous vous recommandons de spécifier un chemin de rubrique au lieu d'utiliser uniquement le# caractère générique dans vos abonnements. Ces messages sont envoyés au canal par défaut que vous spécifiez pour le connecteur.

Les messages d'entrée sur les rubriques définies par le client sont traités comme des données binaires. Ils peuvent utiliser n'importe quel format de message et peuvent contenir n'importe quel type de données. Vous pouvez utiliser les rubriques définies par le client pour acheminer les messages à partir des appareils qui publient dans des rubriques fixes. Vous pouvez également les utiliser pour accepter des données d'entrée provenant de périphériques clients qui ne peuvent pas les transformer en un message formaté à envoyer au connecteur.

Pour plus d'informations sur les abonnements et les rubriques MQTT, consultez the section called “Entrées et sorties”.

Le rôle de groupe doit autoriser l'action `iotanalytics:BatchPutMessage` sur tous les canaux de destination. Pour plus d'informations, veuillez consulter the section called “Prérequis”.

Filtre de rubriques : `iotanalytics/channels/+/messages/put`

Utilisez cette rubrique pour envoyer des messages formatés au connecteur et spécifier dynamiquement un canal de destination. Cette rubrique vous permet également de spécifier un ID qui est renvoyé dans la réponse. Le connecteur vérifie que les ID sont uniques pour chaque message dans la demande `BatchPutMessage` de sortie qu'il envoie à AWS IoT Analytics. Un message qui a un ID dupliqué est abandonné.

Les données d'entrée envoyées à cette rubrique doivent utiliser le format de message suivant.

Propriétés des messages

`request`

Données à envoyer au canal spécifié.

Obligatoire :`true`

Le type suivant.`object`

`message`

Données de l'appareil ou du capteur au format JSON ou sous forme de chaîne encodée en base64.

Obligatoire :`true`

Type: `string`

`id`

ID arbitraire de la demande. Cette propriété est utilisée pour mapper une demande d'entrée à une réponse de sortie. Lorsque spécifiée, la propriété `id` dans l'objet de réponse est définie sur cette valeur. Si vous omettez cette propriété, le connecteur génère un ID.

Obligatoire :`false`

Type: `string`

Schéma valide :`.*`

Exemple d'entrée

```
{
    "request": {
        "message" : "{\"temp\":23.33}"
    },
    "id" : "req123"
}
```

Filtre de rubriques : `iotanalytics/channels/+/messages/binary/put`

Utilisez cette rubrique pour envoyer des messages non formatés au connecteur et spécifier dynamiquement un canal de destination.

Les données du connecteur n'analysent pas les messages d'entrée reçus dans cette rubrique. Elles sont traitées comme des données binaires. Avant d'envoyer les messages à AWS IoT Analytics, le connecteur les code et les formate pour respecter les exigences de l'API `BatchPutMessage` :

- Le connecteur en base64 code les données brutes et inclut la charge utile codée dans une demande `BatchPutMessage` sortante.
- Le connecteur génère et attribue un ID à chaque message d'entrée.

> **Note**
>
> La réponse du connecteur n'inclut pas de corrélation d'ID pour ces messages d'entrée.

Propriétés des messages

Aucun.

Filtre de rubriques : #

Utilisez cette rubrique pour envoyer n'importe quel format de message au canal par défaut. Cela est particulièrement utile lorsque vos appareils clients publient sur des sujets fixes ou lorsque vous souhaitez envoyer des données vers le canal par défaut depuis des appareils clients qui ne peuvent pas traiter les données dans le format de message pris en charge par le connecteur.

Vous définissez la syntaxe du sujet dans l'abonnement que vous créez pour connecter ce connecteur à la source de données. Nous vous recommandons de spécifier un chemin de rubrique au lieu d'utiliser uniquement le# caractère générique dans vos abonnements.

Les données du connecteur n'analysent pas les messages qui sont publiés dans cette rubrique d'entrée. Tous les messages d'entrée sont traités comme des données binaires. Avant d'envoyer les messages à AWS IoT Analytics, le connecteur les code et les formate pour respecter les exigences de l'API `BatchPutMessage` :

- Le connecteur en base64 code les données brutes et inclut la charge utile codée dans une demande `BatchPutMessage` sortante.
- Le connecteur génère et attribue un ID à chaque message d'entrée.

> **Note**
>
> La réponse du connecteur n'inclut pas de corrélation d'ID pour ces messages d'entrée.

Propriétés des messages

Aucun.

Données de sortie

Ce connecteur publie des informations d'état sous forme de données de sortie dans une rubrique MQTT. Ces informations contiennent la réponse renvoyée parAWS IoT Analytics pour chaque message d'entrée qu'elle reçoit et auquel elle est envoyéeAWS IoT Analytics.

Filtre de rubrique dans l'abonnement

```
iotanalytics/messages/put/status
```

Exemple de sortie : réussite

```
{
```

```
    "response" : {
        "status" : "success"
    },
    "id" : "req123"
}
```

Exemple de sortie : échec

```
{
    "response" : {
        "status" : "fail",
        "error" : "ResourceNotFoundException",
        "error_message" : "A resource with the specified name could not be found."
    },
    "id" : "req123"
}
```

> **Note**
>
> Si le connecteur détecte une erreur pouvant être réessayée (par exemple, des erreurs de connexion), il réessaie de publier dans le lot suivant. Le retard exponentiel est géré par leAWS SDK. Les demandes qui échouent avec des erreurs pouvant être retentées sont ajoutées à la file d'attente des canaux pour la publication en fonction du paramètre `IotAnalyticsQueueDropBehavior`.

Exemple d'utilisation

Suivez les étapes de haut niveau suivantes pour configurer un exemple de fonction Lambda Python 3.7 que vous pouvez utiliser pour tester le connecteur.

> **Note**
>
> - Si vous utilisez d'autres environnements d'exécution Python, vous pouvez créer un lien symbolique entre Python3.x et Python 3.7.
> - Les rubriques Démarrer avec les connecteurs (console) et Démarrer avec les connecteurs (CLI) contiennent des étapes détaillées qui vous montrent comment configurer et déployer un exemple de connecteur Twilio Notifications.

1. Veillez à répondre aux conditions requises pour le connecteur.

   Pour l'exigence de rôle de groupe, vous devez configurer le rôle de manière à accorder les autorisations requises et à vous assurer que le rôle a été ajouté au groupe. Pour plus d'informations, consultez the section called “Gérer le rôle de groupe (console)” ou the section called “Gérer le rôle de groupe (interface de ligne de commande)”.

2. Créez et publiez une fonction Lambda qui envoie des données d'entrée au connecteur.

   Enregistrez l'exemple de code en tant que fichier PY. Téléchargez et décompressez le SDKAWS IoT Greengrass Core pour Python. Ensuite, créez un package zip contenant le fichier PY et le dossier `greengrasssdk` au niveau racine. Ce package zip est le package de déploiement vers lequel vous importezAWS Lambda.

   Après avoir créé la fonction Lambda Python 3.7, publiez une version de la fonction et créez un alias.

3. Configurez votre groupe Greengrass.

   a. Ajoutez la fonction Lambda par son alias (recommandé). Configurez le cycle de vie Lambda de manière à ce qu'il dure longtemps (ou`"Pinned": true` dans la CLI).

   b. Ajoutez le connecteur et configurez ses paramètres.

   c. Ajoutez des abonnements qui permettent au connecteur de recevoir des données d'entrée et d'envoyer des données de sortie sur des filtres de rubrique pris en charge.

      - Définissez la fonction Lambda comme source, le connecteur comme cible et utilisez un filtre de rubrique d'entrée compatible.
      - Définissez le connecteur en tant que source, AWS IoT Core en tant que cible et utilisez un filtre de rubrique de sortie pris en charge. Cet abonnement vous permet de consulter les messages d'état dans laAWS IoT console.

4. Déployez le groupe.

5. Dans laAWS IoT console, sur la page Test, abonnez-vous à la rubrique sur les données de sortie pour consulter les messages d'état du connecteur. L'exemple de fonction Lambda a une longue durée de vie et commence à envoyer des messages immédiatement après le déploiement du groupe.

   Lorsque vous avez terminé les tests, vous pouvez définir le cycle de vie Lambda à la demande (ou`"Pinned": false` dans la CLI) et déployer le groupe. Cela empêche la fonction d'envoyer des messages.

Exemple

L'exemple de fonction Lambda suivant envoie un message d'entrée au connecteur.

```
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
```

Limites

Ce connecteur est soumis aux limites suivantes.

- Toutes les limites imposées par leAWS SDK for Python (Boto3) pour l'AWS IoT Analytics`batch_put_message`action.
- Tous les quotas imposés par l'AWS IoT Analytics BatchPutMessageAPI. Pour plus d'informations, consultez la section Service Quotas pourAWS IoT Analytics le Références générales AWS.
  - 100 000 messages par seconde par canal.
  - 100 messages par lot.
  - 128 Ko par message.

Cette API utilise les noms des canaux (et non leurs ARN), ce qui signifie que l'envoi des données vers des canaux couvrant plusieurs régions ou comptes n'est pas pris en charge.

- Tous les quotas imposés par le AWS IoT Greengrass Core. Pour plus d'informations, consultez la section Service Quotas pour leAWS IoT Greengrass noyau dans le Références générales AWS.

  Les quotas suivants peuvent être plus particulièrement applicables :

  - La taille maximale des messages envoyés par un appareil est de 128 Ko.
  - La taille maximale de la file d'attente des messages du routeur principal Greengrass est de 2,5 Mo.
  - La longueur maximale d'une chaîne de rubrique est de 256 octets de caractères codés UTF-8.

Licences

Le connecteur IoT Analytics inclut les logiciels/licences tiers suivants :

- AWS SDK for Python (Boto3)/Licence Apache 2.0
- botocore/Licence Apache 2.0
- dateutil/Licence PSF
- docutils/Licence BSD, licence GPL (General Public License) GNU, licence Python Software Foundation, domaine public
- jmespath/Licence MIT
- s3transfer/Licence Apache 2.0
- urllib3/Licence MIT

Ce connecteur est publié dans le cadre du contrat de licence du logiciel Greengrass Core.

Journal des modifications

Le tableau suivant décrit les modifications dans chaque édition du connecteur

Version	Modifications
4	Ajoute le`IsolationMode` paramètre permettant de configurer le mode de conteneur isation pour le connecteur.

Version	Modifications
3	Mise à niveau du runtime Lambda vers Python 3.7, ce qui modifie les exigences en matière d'exécution.
2	Correctif pour réduire la journalisation excessive.
1	Première version.

Un groupe Greengrass ne peut contenir qu'une seule version du connecteur à la fois. Pour de plus amples informations sur la mise à niveau d'une version de connecteur, veuillez consulter the section called “Mise à niveau des versions du connecteur”.

Consulter aussi

- Intégrer à des services et protocoles à l'aide de connecteurs
- the section called “Démarrer avec les connecteurs (console)”
- the section called “Démarrer avec les connecteurs (CLI)”
- Qu'est-ce que AWS IoT Analytics ? dans le Guide de l'utilisateur AWS IoT Analytics

Connecteur adaptateur de protocole IP IoT Ethernet

L'adaptateur de protocole IP Ethernet IoTconnecteurcollecte des données à partir d'appareils locaux à l'aide du protocole EtherNet/IP. Vous pouvez utiliser ce connecteur pour collecter des données à partir de plusieurs appareils et les publier sur un`StreamManager`flux de messages.

Vous pouvez également utiliser ce connecteur avec l'IoT SiteWise connecteur et votre passerelle IoT SiteWise. Votre passerelle doit fournir la configuration du connecteur. Pour de plus amples informations, veuillez consulterConfiguration d'une source EtherNet/IP (EIP)dans l'IoT SiteWise guide de l'utilisateur.

> **Note**
>
> Ce connecteur fonctionne dansAucun conteneurmode d'isolation, pour que vous puissiez le déployer sur unAWS IoT Greengrassgroupe exécuté dans un conteneur Docker.

Ce connecteur a les versions suivantes.

Version	ARN
2 (recommandé)	`arn:aws:greengrass:  region::/connectors/IoTEIPProtocolAdaptor/versions/2`
1	`arn:aws:greengrass:  region::/connectors/IoTEIPProtocolAdaptor/versions/1`

Pour obtenir des informations sur les changements apportés aux versions, veuillez consulter le Journal des modifications.

Prérequis

Ce connecteur possède les critères suivants :

Version 1 and 2

- AWS IoT GreengrassLogiciel principal v1.10.2 ou version ultérieure.
- Gestionnaire de flux activé sur l'ongletAWS IoT Greengrass.
- Java 8 installé sur l'appareil principal et ajouté au modulePATHvariable d'environnement.
- Un minimum de 256 Mo de RAM supplémentaire. Cette exigence s'ajoute àAWS IoT GreengrassExigences en mémoire de base.

> **Note**
>
> Ce connecteur est disponible dans les régions suivantes :

- cn-north-1
- ap-southeast-1
- ap-southeast-2
- eu-central-1
- eu-west-1
- us-east-1
- us-west-2

Paramètres du connecteur

Ce connecteur prend en charge les paramètres suivants :

`LocalStoragePath`

Le répertoire sur leAWS IoT Greengrasshôte que l'IoT SiteWise peut écrire des données persistantes dans. Le répertoire par défaut est `/var/sitewise`.

Nom d'affichage dans l'ongletAWS IoTConsole : Stockage local chemin d'accès

Obligatoire`false`

Type: `string`

Modèle valide :`^\s*$|\/.`

`ProtocolAdapterConfiguration`

Ensemble de configurations de collecteurs EtherNet/IP à partir desquels le connecteur collecte des données ou se connecte. Cela peut être une liste vide.

Nom d'affichage dans l'ongletAWS IoTConsole : Configuration de l'adaptateur de protocole

Obligatoire`true`

Type : Chaîne JSON bien formée qui définit l'ensemble des configurations de commentaires prises en charge.

Voici un exemple de`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"
                        }
                    ]
                }
            ]
        }
    ]
}
```

Exemple de création de connecteur (AWS CLI)

La commande d'interface de ligne de commande suivante crée un élément`ConnectorDefinition`avec une version initiale qui contient le connecteur 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 fonction Lambda de ce connecteur possède unlongue duréeCycle de vie.

Données d'entrée

Ce connecteur n'accepte pas les messages MQTT comme données d'entrée.

Données de sortie

Ce connecteur publie des données dans`StreamManager`. Vous devez configurer le flux de messages de destination. Les messages de sortie ont la structure suivante :

```
{
    "alias": "string",
    "messages": [
        {
            "name": "string",
            "value": boolean|double|integer|string,
            "timestamp": number,
            "quality": "string"
        }
```

```
    ]
}
```

Licences

Le connecteur IoT Ethernet IP Protocol Adapter inclut les logiciels et licences tiers suivants :

- Client EtherNet/IP
- MapDB
- Elsa

Ce connecteur est libéré sous leContrat de licence du logiciel Greengrass Core.

Journal des modifications

Le tableau suivant décrit les modifications apportées à chaque version du connecteur.

Version	Modifications	Date
2	Cette version contient des correctifs de bogues.	23 décembre 2021
1	Première version.	15 décembre 2020

Un groupe Greengrass peut contenir une seule version du connecteur à la fois. Pour de plus amples informations sur la mise à niveau d'une version de connecteur, veuillez consulter the section called “Mise à niveau des versions du connecteur”.

Consulter aussi

- Intégrer à des services et protocoles à l'aide de connecteurs
- the section called “Démarrer avec les connecteurs (console)”
- the section called “Démarrer avec les connecteurs (CLI)”

SiteWise Connecteur IoT

Le SiteWise connecteur IoT envoie les données des appareils et équipements locaux aux propriétés des actifs dansAWS IoT SiteWise. Vous pouvez utiliser ce connecteur pour collecter des données provenant de plusieurs serveurs OPC-UA et les publier sur l'IoT. SiteWise Le connecteur envoie les données aux propriétés des actifs dans la région actuelle Compte AWS et dans la région.

> **Note**
>
> L'IoT SiteWise est un service entièrement géré qui collecte, traite et visualise les données provenant d'appareils et d'équipements industriels. Vous pouvez configurer les propriétés des ressources afin qu'elles effectuent un traitement des données brutes envoyées par ce connecteur aux propriétés de mesure de vos ressources. Par exemple, vous pouvez définir une propriété de transformation qui convertit en Fahrenheit les points de données de température d'un appareil exprimés en Celsius, ou vous pouvez définir une propriété de mesure qui calcule la température horaire moyenne. Pour plus d'informations, consultez Présentation d'AWS IoT SiteWise dans le Guide de l'utilisateur AWS IoT SiteWise.

Le connecteur envoie des données à l'IoT SiteWise avec les chemins de flux de données OPC-UA envoyés par les serveurs OPC-UA. Par exemple, le chemin de flux de données `/company/windfarm/3/turbine/7/temperature` peut représenter le capteur de température de la turbine n°7 au parc éolien n°3. Si le AWS IoT Greengrass cœur perd la connexion à Internet, le connecteur met en cache les données jusqu'à ce qu'il puisse se connecter correctement auAWS Cloud. Vous pouvez configurer la taille maximale du tampon disque utilisée pour la mise en cache des données. Si la taille du cache dépasse la taille maximale du tampon disque, le connecteur supprime les données les plus anciennes de la file d'attente.

Après avoir configuré et déployé le SiteWise connecteur IoT, vous pouvez ajouter une passerelle et des sources OPC-UA dans la console IoT SiteWise . Lorsque vous configurez une source dans la console, vous pouvez filtrer ou préfixer les chemins de flux de données OPC-UA envoyés par le connecteur IoT. SiteWise Pour obtenir des instructions sur la manière de terminer la configuration de votre passerelle et de vos sources, veuillez consulter Ajout de la passerelle dans le Guide de l'utilisateur AWS IoT SiteWise.

L'IoT SiteWise reçoit des données uniquement à partir de flux de données que vous avez mappés aux propriétés de mesure des SiteWise actifs IoT. Pour mapper des flux de données aux propriétés des ressources, vous pouvez définir l'alias d'une propriété comme équivalent à un chemin de flux de

données OPC-UA. Pour de plus amples informations sur la définition de modèles de ressource et la création de ressources, veuillez consulter Modélisation de ressources industrielles dans le Guide de l'utilisateur AWS IoT SiteWise.

> **Remarques**
>
> Vous pouvez utiliser le gestionnaire de flux pour télécharger des données vers l'IoT à SiteWise partir de sources autres que les serveurs OPC-UA. Le gestionnaire de flux fournit également un support personnalisable pour la persistance et la gestion de la bande passante. Pour plus d'informations, consultez Gestion des flux de données.
> Ce connecteur fonctionne en mode sans isolation de conteneur. Vous pouvez donc le déployer sur un groupe Greengrass exécuté dans un conteneur Docker.

Ce connecteur est disponible dans les versions suivantes.

Version	ARN
12 (recommandé)	`arn:aws:greengrass:region::/connectors/IoTSiteWise/versions/12`
11	`arn:aws:greengrass:region::/connectors/IoTSiteWise/versions/11`
10	`arn:aws:greengrass:region::/connectors/IoTSiteWise/versions/10`
9	`arn:aws:greengrass:region::/connectors/IoTSiteWise/versions/9`
8	`arn:aws:greengrass:region::/connectors/IoTSiteWise/versions/8`

Version	ARN
7	`arn:aws:greengrass:region::/connectors/IoTSiteWise/versions/7`
6	`arn:aws:greengrass:region::/connectors/IoTSiteWise/versions/6`
5	`arn:aws:greengrass:region::/connectors/IoTSiteWise/versions/5`
4	`arn:aws:greengrass:region::/connectors/IoTSiteWise/versions/4`
3	`arn:aws:greengrass:region::/connectors/IoTSiteWise/versions/3`
2	`arn:aws:greengrass:region::/connectors/IoTSiteWise/versions/2`
1	`arn:aws:greengrass:region::/connectors/IoTSiteWise/versions/1`

Pour obtenir des informations sur les changements apportés aux versions, veuillez consulter le Journal des modifications.

Prérequis

Ce connecteur possède les critères suivants :

Version 9, 10, 11, and 12

> **Important**
>
> Cette version introduit de nouvelles exigences : logiciel de AWS IoT Greengrass base v1.10.2 et gestionnaire de flux.

- AWS IoT GreengrassLogiciel de base v1.10.2.
- Gestionnaire de flux activé sur le groupe Greengrass.
- Java 8 installé sur l'appareil principal et ajouté à la variable d'environnement PATH.
- Ce connecteur ne peut être utilisé que dans les régions Amazon Web Services où les deux technologies AWS IoT Greengrasset l'IoT SiteWise sont pris en charge.
- Une politique IAM a été ajoutée au rôle de groupe Greengrass. Ce rôle permet au groupe AWS IoT Greengrass d'accéder à l'action `iotsitewise:BatchPutAssetPropertyValue` sur l'actif racine cible et ses enfants, comme illustré dans l'exemple suivant. Vous pouvez le supprimer `Condition` de la politique pour permettre au connecteur d'accéder à tous vos SiteWise actifs IoT.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
             "Effect": "Allow",
             "Action": "iotsitewise:BatchPutAssetPropertyValue",
             "Resource": "*",
             "Condition": {
                 "StringLike": {
                     "iotsitewise:assetHierarchyPath": [
                         "/root node asset ID",
                         "/root node asset ID/*"
                     ]
                 }
             }
        }
    ]
}
```

Pour plus d'informations, consultez la section Ajout et suppression de politiques IAM dans le Guide de l'utilisateur IAM.

Versions 6, 7, and 8

> **Important**
>
> Cette version introduit de nouvelles exigences : logiciel AWS IoT Greengrass Core version 1.10.0 et gestionnaire de flux.

- AWS IoT GreengrassLogiciel de base v1.10.0.
- Gestionnaire de flux activé sur le groupe Greengrass.
- Java 8 installé sur l'appareil principal et ajouté à la variable d'environnement PATH.
- Ce connecteur ne peut être utilisé que dans les régions Amazon Web Services où les deux technologies AWS IoT Greengrasset l'IoT SiteWise sont pris en charge.
- Une politique IAM a été ajoutée au rôle de groupe Greengrass. Ce rôle permet au groupe AWS IoT Greengrass d'accéder à l'action `iotsitewise:BatchPutAssetPropertyValue` sur l'actif racine cible et ses enfants, comme illustré dans l'exemple suivant. Vous pouvez le supprimer `Condition` de la politique pour permettre au connecteur d'accéder à tous vos SiteWise actifs IoT.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
             "Effect": "Allow",
             "Action": "iotsitewise:BatchPutAssetPropertyValue",
             "Resource": "*",
             "Condition": {
                 "StringLike": {
                     "iotsitewise:assetHierarchyPath": [
                         "/root node asset ID",
                         "/root node asset ID/*"
                     ]
                 }
             }
        }
```

```
    ]
}
```

Pour plus d'informations, consultez la section Ajout et suppression de politiques IAM dans le Guide de l'utilisateur IAM.

Version 5

- AWS IoT GreengrassLogiciel de base v1.9.4.
- Java 8 installé sur l'appareil principal et ajouté à la variable d'environnement PATH.
- Ce connecteur ne peut être utilisé que dans les régions Amazon Web Services où les deux technologies AWS IoT Greengrasset l'IoT SiteWise sont pris en charge.
- Une politique IAM a été ajoutée au rôle de groupe Greengrass. Ce rôle permet au groupe AWS IoT Greengrass d'accéder à l'action `iotsitewise:BatchPutAssetPropertyValue` sur l'actif racine cible et ses enfants, comme illustré dans l'exemple suivant. Vous pouvez le supprimer `Condition` de la politique pour permettre au connecteur d'accéder à tous vos SiteWise actifs IoT.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
             "Effect": "Allow",
             "Action": "iotsitewise:BatchPutAssetPropertyValue",
             "Resource": "*",
             "Condition": {
                 "StringLike": {
                     "iotsitewise:assetHierarchyPath": [
                         "/root node asset ID",
                         "/root node asset ID/*"
                     ]
                 }
             }
        }
    ]
}
```

Pour plus d'informations, consultez la section Ajout et suppression de politiques IAM dans le Guide de l'utilisateur IAM.

Version 4

- AWS IoT GreengrassLogiciel de base v1.10.0.
- Java 8 installé sur l'appareil principal et ajouté à la variable d'environnement PATH.
- Ce connecteur ne peut être utilisé que dans les régions Amazon Web Services où les deux technologies AWS IoT Greengrasset l'IoT SiteWise sont pris en charge.
- Une politique IAM a été ajoutée au rôle de groupe Greengrass. Ce rôle permet au groupe AWS IoT Greengrass d'accéder à l'action `iotsitewise:BatchPutAssetPropertyValue` sur l'actif racine cible et ses enfants, comme illustré dans l'exemple suivant. Vous pouvez le supprimer `Condition` de la politique pour permettre au connecteur d'accéder à tous vos SiteWise actifs IoT.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
             "Effect": "Allow",
             "Action": "iotsitewise:BatchPutAssetPropertyValue",
             "Resource": "*",
             "Condition": {
                 "StringLike": {
                     "iotsitewise:assetHierarchyPath": [
                         "/root node asset ID",
                         "/root node asset ID/*"
                     ]
                 }
             }
        }
    ]
}
```

Pour plus d'informations, consultez la section Ajout et suppression de politiques IAM dans le Guide de l'utilisateur IAM.

Version 3

- AWS IoT GreengrassLogiciel de base v1.9.4.
- Java 8 installé sur l'appareil principal et ajouté à la variable d'environnement PATH.

- Ce connecteur ne peut être utilisé que dans les régions Amazon Web Services où les deux technologies AWS IoT Greengrasset l'IoT SiteWise sont pris en charge.
- Une politique IAM a été ajoutée au rôle de groupe Greengrass. Ce rôle permet au groupe AWS IoT Greengrass d'accéder à l'action `iotsitewise:BatchPutAssetPropertyValue` sur l'actif racine cible et ses enfants, comme illustré dans l'exemple suivant. Vous pouvez le supprimer `Condition` de la politique pour permettre au connecteur d'accéder à tous vos SiteWise actifs IoT.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
             "Effect": "Allow",
             "Action": "iotsitewise:BatchPutAssetPropertyValue",
             "Resource": "*",
             "Condition": {
                 "StringLike": {
                     "iotsitewise:assetHierarchyPath": [
                         "/root node asset ID",
                         "/root node asset ID/*"
                     ]
                 }
             }
        }
    ]
}
```

Pour plus d'informations, consultez la section Ajout et suppression de politiques IAM dans le Guide de l'utilisateur IAM.

Versions 1 and 2

- AWS IoT GreengrassLogiciel de base v1.9.4.
- Java 8 installé sur l'appareil principal et ajouté à la variable d'environnement PATH.
- Ce connecteur ne peut être utilisé que dans les régions Amazon Web Services où les deux technologies AWS IoT Greengrasset l'IoT SiteWise sont pris en charge.
- Une politique IAM ajoutée au rôle de groupe Greengrass permet d'accéder AWS IoT Core à `iotsitewise:BatchPutAssetPropertyValue` l'actif racine cible et à ses enfants et d'agir

sur ceux-ci, comme illustré dans l'exemple suivant. Vous pouvez le supprimer `Condition` de la politique pour permettre au connecteur d'accéder à tous vos SiteWise actifs 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": "*"
        }
    ]
}
```

Pour plus d’informations, consultez la rubrique Ajout et suppression d’autorisations basées sur l’identité IAM du Guide de l’utilisateur IAM.

Paramètres

Versions 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, and 12

`SiteWiseLocalStoragePath`

Le répertoire de l'AWS IoT Greengrasshôte dans lequel le SiteWise connecteur IoT peut écrire des données persistantes. La valeur par défaut est `/var/sitewise`.

Nom affiché dans la AWS IoT console : chemin de stockage local

Nécessaire : `false`

Type : `string`

Modèle valide : `^\s*$|\/.`

`AWSSecretsArnList`

Liste de secrets dans AWS Secrets Manager contenant chacun le nom d'utilisateur OPC-UA et la paire clé-valeur du mot de passe. Chaque secret doit être un secret de type paire clé-valeur.

Nom d'affichage dans la AWS IoT console : liste des ARN pour les secrets de nom d'utilisateur/mot de passe OPC-UA

Nécessaire : `false`

Type : `JsonArrayOfStrings`

Modèle valide : `\[( ?,? ?\"(arn:(aws(-[a-z]+)*):secretsmanager:[a-z0-9\\-]+:[0-9]{12}:secret:([a-zA-Z0-9\\\\]+\/)*[a-zA-Z0-9\/_+=,.@\\-]+-[a-zA-Z0-9]+)*\")*\]`

`MaximumBufferSize`

Taille maximale en Go pour l'utilisation des SiteWise disques IoT. La valeur par défaut est 10 Go.

Nom affiché dans la AWS IoT console : taille maximale de la mémoire tampon du disque

Nécessaire : `false`

Type : `string`

Modèle valide : `^\s*$|[0-9]+`

Version 1

`SiteWiseLocalStoragePath`

Le répertoire de l'AWS IoT Greengrasshôte dans lequel le SiteWise connecteur IoT peut écrire des données persistantes. La valeur par défaut est `/var/sitewise`.

Nom affiché dans la AWS IoT console : chemin de stockage local

Nécessaire : `false`

Type : `string`

Modèle valide : `^\s*$|\/.`

`SiteWiseOpcuaUserIdentityTokenSecretArn`

Secret dans AWS Secrets Manager qui contient le nom d'utilisateur OPC-UA et la paire clé-valeur du mot de passe. Ce secret doit être un secret de type paire clé-valeur.

Nom affiché dans la AWS IoT console : ARN du code secret du nom d'utilisateur/mot de passe OPC-UA

Nécessaire : `false`

Type : `string`

Modèle valide : `^$|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`

Ressource secrète du groupe AWS IoT Greengrass qui fait référence à un secret de nom d'utilisateur et mot de passe OPC-UA.

Nom affiché dans la AWS IoT console : ressource secrète de nom d'utilisateur/mot de passe OPC-UA

Nécessaire : `false`

Type : `string`

Modèle valide : `^$|.+`

`MaximumBufferSize`

Taille maximale en Go pour l'utilisation des SiteWise disques IoT. La valeur par défaut est 10 Go.

Nom affiché dans la AWS IoT console : taille maximale de la mémoire tampon du disque

Nécessaire : `false`

Type : `string`

Modèle valide : `^\s*$|[0-9]+`

Exemple de création de connecteur (AWS CLI)

La AWS CLI commande suivante crée un `ConnectorDefinition` avec une version initiale contenant le SiteWise connecteur IoT.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
    "Connectors": [
        {
            "Id": "MyIoTSiteWiseConnector",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/IoTSiteWise/
versions/11"
        }
    ]
}'
```

> **Note**
>
> Les fonctions Lambda de ce connecteur ont un cycle de vie prolongé.

Dans la AWS IoT Greengrass console, vous pouvez ajouter un connecteur depuis la page Connecteurs du groupe. Pour plus d'informations, consultez the section called "Démarrer avec les connecteurs (console)".

Données d'entrée

Ce connecteur n'accepte pas les messages MQTT comme données d'entrée.

Données de sortie

Ce connecteur ne publie pas les messages MQTT en tant que données de sortie.

Limites

Ce connecteur est soumis à toutes les limites imposées par l'IoT SiteWise, y compris les suivantes. Pour plus d'informations, consultez la section AWS IoT SiteWisePoints de terminaison et quotas dans le. Références générales AWS

- Nombre maximum de passerelles par. Compte AWS
- Nombre maximal de sources OPC-UA par passerelle.
- Débit maximal de points de données timestamp-quality-value (TQV) stockés par. Compte AWS
- Taux maximal de points de données TQV stockés par propriété de ressource.

Licences

Version 9, 10, 11, and 12

Le SiteWise connecteur IoT inclut les logiciels/licences tiers suivants :

- Base de données cartographique
- Elsa
- Éclipse Milo

Ce connecteur est publié dans le cadre du contrat de licence logicielle Greengrass Core.

Versions 6, 7, and 8

Le SiteWise connecteur IoT inclut les logiciels/licences tiers suivants :

- Milo / EDL 1.0

Ce connecteur est publié dans le cadre du contrat de licence logicielle Greengrass Core.

Versions 1, 2, 3, 4, and 5

Le SiteWise connecteur IoT inclut les logiciels/licences tiers suivants :

- Milo / EDL 1.0

- Chronicle-Queue / Licence Apache 2.0

Ce connecteur est publié dans le cadre du contrat de licence logicielle Greengrass Core.

Journal des modifications

Le tableau suivant décrit les modifications apportées à chaque version du connecteur.

Version	Modifications	Date
12	• Cette version contient des corrections de bogues.	22 décembre 2021
11	• Support pour les chaînes contenant des caractères masqués ou non imprimabl es. Les caractères masqués et non imprimables sont automatiquement supprimés avant que les chaînes ne soient envoyées au. AWS Cloud • Correction d'un problème en raison duquel la SiteWise passerelle IoT réessayait indéfiniment les demandes non valides. • Correction d'un problème qui provoquait un point de contrôle endommagé lorsque la SiteWise passerelle IoT était connectée à une source de données haute fréquence. • Messages d'erreur améliorés pour aider à	24 mars 2021

Version	Modifications	Date
	résoudre les problèmes de configuration de la passerelle.	
10	Configuré `StreamManager` pour améliorer la gestion lorsque la connexion source est perdue puis rétablie. Cette version accepte également les valeurs OPC-UA avec un `ServerTimestamp` lorsque non `SourceTimestamp` est disponible.	22 janvier 2021
9	Support lancé pour les destinations de `StreamManager` streaming Greengrass personnalisées, le deadbanding OPC-UA, le mode de scan personnalisé et le taux de scan personnalisé. Inclut également des performances améliorées lors des mises à jour de configuration effectuées à partir de la SiteWise passerelle IoT.	15 décembre 2020
8	Stabilité améliorée lorsque le connecteur est confronté à une connectivité réseau intermittente.	19 novembre 2020
7	Correction d'un problème lié aux métriques de passerelle.	14 août 2020

Version	Modifications	Date
6	Ajout de la prise en charge des CloudWatch métriques et de la découverte automatiq ue de nouvelles balises OPC-UA. Cette version nécessite le gestionnaire de flux et le logiciel AWS IoT Greengras s Core version 1.10.0 ou ultérieure.	29 avril 2020
5	Correction d'un problème de compatibilité avec le logiciel AWS IoT Greengrass Core v1.9.4.	12 février 2020
4	Correction d'un problème lié à la reconnexion du serveur OPC-UA.	7 février 2020
3	Suppression de l'obligation des autorisations `iot:*`.	17 décembre 2019
2	Ajout de la prise en charge de plusieurs ressources de secret OPC-UA.	10 décembre 2019
1	Première version.	2 décembre 2019

Un groupe Greengrass ne peut contenir qu'une seule version du connecteur à la fois. Pour de plus amples informations sur la mise à niveau d'une version de connecteur, veuillez consulter the section called “Mise à niveau des versions du connecteur”.

Consultez aussi

- Intégrer à des services et protocoles à l'aide de connecteurs
- the section called “Démarrer avec les connecteurs (console)”

- the section called “Démarrer avec les connecteurs (CLI)”
- Consultez les rubriques suivantes dans le Guide de l'utilisateur AWS IoT SiteWise :
  - Présentation de AWS IoT SiteWise
  - Utilisation d'une passerelle
  - CloudWatch Métriques de la passerelle
  - Résolution des problèmes liés à une SiteWise passerelle IoT

Kinesis Firehose

Le connecteur Kinesis Firehose publie des données via un flux de diffusion Amazon Data Firehose vers des destinations telles qu'Amazon S3, Amazon Redshift ou Amazon Service. OpenSearch

Ce connecteur est un producteur de données pour un flux de diffusion Kinesis. Il reçoit les données d'entrée dans une rubrique MQTT et envoie les données à un flux de diffusion spécifié. Le flux de diffusion envoie ensuite l'enregistrement des données à la destination configurée (par exemple, un compartiment S3).

Ce connecteur est disponible dans les versions suivantes.

Version	ARN
5	`arn:aws:greengrass:  region::/connectors/KinesisFirehose/versions/5`
4	`arn:aws:greengrass:  region::/connectors/KinesisFirehose/versions/4`
3	`arn:aws:greengrass:  region::/connectors/KinesisFirehose/versions/3`
2	`arn:aws:greengrass:  region::/connectors/KinesisFirehose/versions/2`

Version	ARN
1	`arn:aws:greengrass:  region::/connectors/KinesisFirehose/versions/1`

Pour obtenir des informations sur les changements apportés aux versions, veuillez consulter le Journal des modifications.

Prérequis

Ce connecteur possède les critères suivants :

Version 4 - 5

- AWS IoT Greengrass Logiciel de base v1.9.3 ou version ultérieure.
- Python version 3.7 ou 3.8 installé sur le périphérique principal et ajouté à la variable d'environnement PATH.

  > **Note**
  >
  > Pour utiliser Python 3.8, exécutez la commande suivante pour créer un lien symbolique entre le dossier d'installation par défaut de Python 3.7 et les fichiers binaires Python 3.8 installés.
  >
  > ```
  > sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
  > ```
  >
  > Ceci configure votre appareil de sorte qu'il réponde aux exigence de Python pour AWS IoT Greengrass.

- Un flux de diffusion Kinesis configuré. Pour plus d'informations, consultez la section Création d'un flux de diffusion Amazon Data Firehose dans le manuel du développeur Amazon Kinesis Firehose.
- Le rôle de groupe Greengrass est configuré pour autoriser les `firehose:PutRecordBatch` actions `firehose:PutRecord` et sur le flux de diffusion cible, comme illustré dans l'exemple de politique IAM suivant.

```
{
    "Version":"2012-10-17",
    "Statement":[
        {
            "Sid":"Stmt1528133056761",
            "Action":[
                "firehose:PutRecord",
                "firehose:PutRecordBatch"
            ],
            "Effect":"Allow",
            "Resource":[
                "arn:aws:firehose:region:account-id:deliverystream/stream-name"
            ]
        }
    ]
}
```

Ce connecteur vous permet de remplacer le flux de diffusion par défaut de façon dynamique dans la charge utile des messages d'entrée. Si votre implémentation utilise cette fonctionnalité, la politique IAM doit inclure tous les flux cibles en tant que ressources. Vous pouvez octroyer un accès précis ou conditionnel aux ressources (par exemple, en utilisant un schéma d'attribution de nom avec caractère générique *).

Pour l'exigence de rôle de groupe, vous devez configurer le rôle de manière à accorder les autorisations requises et à vous assurer que le rôle a été ajouté au groupe. Pour plus d'informations, consultez the section called “Gérer le rôle de groupe (console)” ou the section called “Gérer le rôle de groupe (interface de ligne de commande)”.

Versions 2 - 3

- AWS IoT Greengrass Logiciel de base v1.7 ou version ultérieure.
- Python version 2.7 installé sur le périphérique principal et ajouté à la variable d'environnement PATH.
- Un flux de diffusion Kinesis configuré. Pour plus d'informations, consultez la section Création d'un flux de diffusion Amazon Data Firehose dans le manuel du développeur Amazon Kinesis Firehose.

- Le rôle de groupe Greengrass est configuré pour autoriser les `firehose:PutRecordBatch` actions `firehose:PutRecord` et sur le flux de diffusion cible, comme illustré dans l'exemple de politique IAM suivant.

```
{
    "Version":"2012-10-17",
    "Statement":[
        {
            "Sid":"Stmt1528133056761",
            "Action":[
                "firehose:PutRecord",
                "firehose:PutRecordBatch"
            ],
            "Effect":"Allow",
            "Resource":[
                "arn:aws:firehose:region:account-id:deliverystream/stream-name"
            ]
        }
    ]
 }
```

  Ce connecteur vous permet de remplacer le flux de diffusion par défaut de façon dynamique dans la charge utile des messages d'entrée. Si votre implémentation utilise cette fonctionnalité, la politique IAM doit inclure tous les flux cibles en tant que ressources. Vous pouvez octroyer un accès précis ou conditionnel aux ressources (par exemple, en utilisant un schéma d'attribution de nom avec caractère générique *).

  Pour l'exigence de rôle de groupe, vous devez configurer le rôle de manière à accorder les autorisations requises et à vous assurer que le rôle a été ajouté au groupe. Pour plus d'informations, consultez the section called “Gérer le rôle de groupe (console)” ou the section called “Gérer le rôle de groupe (interface de ligne de commande)”.

Version 1

- AWS IoT Greengrass Logiciel de base v1.7 ou version ultérieure.
- Python version 2.7 installé sur le périphérique principal et ajouté à la variable d'environnement PATH.

- Un flux de diffusion Kinesis configuré. Pour plus d'informations, consultez la section Création d'un flux de diffusion Amazon Data Firehose dans le manuel du développeur Amazon Kinesis Firehose.
- Le rôle de groupe Greengrass est configuré pour autoriser l'`firehose:PutRecord`action sur le flux de diffusion cible, comme illustré dans l'exemple de politique IAM suivant.

```
{
    "Version":"2012-10-17",
    "Statement":[
        {
            "Sid":"Stmt1528133056761",
            "Action":[
                "firehose:PutRecord"
            ],
            "Effect":"Allow",
            "Resource":[
                "arn:aws:firehose:region:account-id:deliverystream/stream-name"
            ]
        }
    ]
 }
```

Ce connecteur vous permet de remplacer le flux de diffusion par défaut de façon dynamique dans la charge utile des messages d'entrée. Si votre implémentation utilise cette fonctionnalité, la politique IAM doit inclure tous les flux cibles en tant que ressources. Vous pouvez octroyer un accès précis ou conditionnel aux ressources (par exemple, en utilisant un schéma d'attribution de nom avec caractère générique *).

Pour l'exigence de rôle de groupe, vous devez configurer le rôle de manière à accorder les autorisations requises et à vous assurer que le rôle a été ajouté au groupe. Pour plus d’informations, consultez the section called “Gérer le rôle de groupe (console)” ou the section called “Gérer le rôle de groupe (interface de ligne de commande)”.

Paramètres du connecteur

Ce connecteur fournit les paramètres suivants :

Versions 5

`DefaultDeliveryStreamArn`

L'ARN du flux de diffusion Firehose par défaut auquel envoyer les données. Le flux de destination peut être remplacé par la propriété `delivery_stream_arn` dans la charge utile du message d'entrée.

> **Note**
>
> Le rôle de groupe doit autoriser les actions appropriées sur toutes les flux cibles de la diffusion. Pour de plus amples informations, veuillez consulter the section called “Prérequis”.

Nom d'affichage dans la AWS IoT console : ARN du flux de diffusion par défaut

Nécessaire : `true`

Type : `string`

Schéma valide : `arn:aws:firehose:([a-z]{2}-[a-z]+-\d{1}):(\d{12}):deliverystream/([a-zA-Z0-9_\-.]+)$`

`DeliveryStreamQueueSize`

Le nombre maximal d'enregistrements à conserver en mémoire avant que de nouveaux enregistrements pour le même flux de diffusion soient rejetés. La valeur minimale est de 2000.

Nom affiché dans la AWS IoT console : nombre maximum d'enregistrements à mettre en mémoire tampon (par flux)

Nécessaire : `true`

Type : `string`

Schéma valide : `^([2-9]\\d{3}|[1-9]\\d{4,})$`

`MemorySize`

Quantité de mémoire (en Ko) allouée à ce connecteur.

Nom affiché dans la AWS IoT console : Taille de la mémoire

Nécessaire : `true`

Type : `string`

Schéma valide : `^[0-9]+$`

`PublishInterval`

Intervalle (en secondes) pendant lequel les enregistrements sont publiés dans Firehose. Pour désactiver la mise en lots, définissez cette valeur sur 0.

Nom d'affichage dans la AWS IoT console : Intervalle de publication

Nécessaire : `true`

Type : `string`

Valeurs valides : `0 - 900`

Schéma valide : `[0-9]|[1-9]\\d|[1-9]\\d\\d|900`

`IsolationMode`

Mode conteneurisation de ce connecteur. La valeur par défaut est`GreengrassContainer`, ce qui signifie que le connecteur s'exécute dans un environnement d'exécution isolé à l'intérieur du AWS IoT Greengrass conteneur.

> **Note**
>
> Le paramètre de conteneurisation par défaut pour le groupe ne s'applique pas aux connecteurs.

Nom affiché dans la AWS IoT console : mode d'isolation du conteneur

Nécessaire : `false`

Type : `string`

Valeurs valides : `GreengrassContainer` ou `NoContainer`

Schéma valide : `^NoContainer$|^GreengrassContainer$`

Versions 2 - 4

`DefaultDeliveryStreamArn`

L'ARN du flux de diffusion Firehose par défaut auquel envoyer les données. Le flux de destination peut être remplacé par la propriété `delivery_stream_arn` dans la charge utile du message d'entrée.

> **Note**
>
> Le rôle de groupe doit autoriser les actions appropriées sur toutes les flux cibles de la diffusion. Pour de plus amples informations, veuillez consulter the section called “Prérequis”.

Nom d'affichage dans la AWS IoT console : ARN du flux de diffusion par défaut

Nécessaire : `true`

Type : `string`

Schéma valide : `arn:aws:firehose:([a-z]{2}-[a-z]+-\d{1}):(\d{12}):deliverystream/([a-zA-Z0-9_\-.]+)$`

`DeliveryStreamQueueSize`

Le nombre maximal d'enregistrements à conserver en mémoire avant que de nouveaux enregistrements pour le même flux de diffusion soient rejetés. La valeur minimale est de 2000.

Nom affiché dans la AWS IoT console : nombre maximum d'enregistrements à mettre en mémoire tampon (par flux)

Nécessaire : `true`

Type : `string`

Schéma valide : `^([2-9]\\d{3}|[1-9]\\d{4,})$`

`MemorySize`

Quantité de mémoire (en Ko) allouée à ce connecteur.

Nom affiché dans la AWS IoT console : Taille de la mémoire

Nécessaire : `true`

Type : `string`

Schéma valide : `^[0-9]+$`

`PublishInterval`

Intervalle (en secondes) pendant lequel les enregistrements sont publiés dans Firehose. Pour désactiver la mise en lots, définissez cette valeur sur 0.

Nom d'affichage dans la AWS IoT console : Intervalle de publication

Nécessaire : `true`

Type : `string`

Valeurs valides : `0 - 900`

Schéma valide : `[0-9]|[1-9]\\d|[1-9]\\d\\d|900`

Version 1

`DefaultDeliveryStreamArn`

L'ARN du flux de diffusion Firehose par défaut auquel envoyer les données. Le flux de destination peut être remplacé par la propriété `delivery_stream_arn` dans la charge utile du message d'entrée.

> **Note**
>
> Le rôle de groupe doit autoriser les actions appropriées sur toutes les flux cibles de la diffusion. Pour de plus amples informations, veuillez consulter the section called “Prérequis”.

Nom d'affichage dans la AWS IoT console : ARN du flux de diffusion par défaut

Nécessaire : `true`

Type : `string`

Schéma valide : `arn:aws:firehose:([a-z]{2}-[a-z]+-\d{1}):(\d{12}):deliverystream/([a-zA-Z0-9_\-.]+)$`

Example

Exemple de création d'un connecteur (AWS CLI)

La commande CLI suivante crée un `ConnectorDefinition` avec une version initiale contenant le connecteur.

```
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"
            }
        }
    ]
}'
```

Dans la AWS IoT Greengrass console, vous pouvez ajouter un connecteur depuis la page Connecteurs du groupe. Pour de plus amples informations, veuillez consulter the section called “Démarrer avec les connecteurs (console)”.

Données d'entrée

Ce connecteur accepte de diffuser le contenu sur les rubriques MQTT, puis renvoie le contenu vers le flux de diffusion cible. Il accepte deux types de données d'entrée :

- les données JSON dans la rubrique `kinesisfirehose/message`.
- les données binaires dans la rubrique `kinesisfirehose/message/binary/#`.

Versions 2 - 5

Filtre de rubriques : `kinesisfirehose/message`

Utilisez cette rubrique pour envoyer un message qui contient des données JSON.

Propriétés des messages

`request`

Données à envoyer au flux de diffusion et au flux de diffusion cible, s'il est différent du flux par défaut.

Nécessaire : `true`

Type : `object` qui inclut les propriétés suivantes :

`data`

Données à envoyer au flux de diffusion.

Nécessaire : `true`

Type : `string`

`delivery_stream_arn`

L'ARN du flux de diffusion Kinesis cible. Incluez cette propriété pour remplacer le flux de diffusion par défaut.

Nécessaire : `false`

Type : `string`

Schéma valide : `arn:aws:firehose:([a-z]{2}-[a-z]+-\d{1}):(\d{12}):deliverystream/([a-zA-Z0-9_\-.]+)$`

`id`

ID arbitraire de la demande. Cette propriété est utilisée pour mapper une demande d'entrée à une réponse de sortie. Lorsque spécifiée, la propriété `id` dans l'objet de réponse est définie sur cette valeur. Si vous n'utilisez pas cette fonctionnalité, vous pouvez omettre cette propriété ou spécifier une chaîne vide.

Nécessaire : `false`

Type : `string`

Schéma valide : `.*`

Exemple d'entrée

```
{
     "request": {
        "delivery_stream_arn": "arn:aws:firehose:region:account-
id:deliverystream/stream2-name",
        "data": "Data to send to the delivery stream."
     },
     "id": "request123"
}
```

Filtre de rubriques : `kinesisfirehose/message/binary/#`

Utilisez cette rubrique pour envoyer un message qui contient des données binaires. Le connecteur n'analyse pas les données binaires. Les données sont diffusées en l'état.

Pour mapper la demande d'entrée à une réponse de sortie, remplacez le caractère générique # dans la rubrique du message par un ID de demande arbitraire. Par exemple, si vous publiez un message dans `kinesisfirehose/message/binary/request123`, la propriété `id` dans l'objet de réponse est définie sur `request123`.

Si vous ne souhaitez pas mapper une demande à une réponse, vous pouvez publier vos messages dans `kinesisfirehose/message/binary/`. Veillez à inclure la barre oblique de fin (/).

Version 1

Filtre de rubriques : `kinesisfirehose/message`

Utilisez cette rubrique pour envoyer un message qui contient des données JSON.

Propriétés des messages

`request`

Données à envoyer au flux de diffusion et au flux de diffusion cible, s'il est différent du flux par défaut.

Nécessaire : `true`

Type : `object` qui inclut les propriétés suivantes :

`data`

Données à envoyer au flux de diffusion.

Nécessaire : `true`

Type : `string`

`delivery_stream_arn`

L'ARN du flux de diffusion Kinesis cible. Incluez cette propriété pour remplacer le flux de diffusion par défaut.

Nécessaire : `false`

Type : `string`

Schéma valide : `arn:aws:firehose:([a-z]{2}-[a-z]+-\d{1}):(\d{12}):deliverystream/([a-zA-Z0-9_\-.]+)$`

`id`

ID arbitraire de la demande. Cette propriété est utilisée pour mapper une demande d'entrée à une réponse de sortie. Lorsque spécifiée, la propriété `id` dans l'objet de réponse est définie sur cette valeur. Si vous n'utilisez pas cette fonctionnalité, vous pouvez omettre cette propriété ou spécifier une chaîne vide.

Nécessaire : `false`

Type : `string`

Schéma valide : `.*`

Exemple d'entrée

```
{
     "request": {
        "delivery_stream_arn": "arn:aws:firehose:region:account-
id:deliverystream/stream2-name",
        "data": "Data to send to the delivery stream."
     },
```

```
    "id": "request123"
}
```

Filtre de rubriques : `kinesisfirehose/message/binary/#`

Utilisez cette rubrique pour envoyer un message qui contient des données binaires. Le connecteur n'analyse pas les données binaires. Les données sont diffusées en l'état.

Pour mapper la demande d'entrée à une réponse de sortie, remplacez le caractère générique # dans la rubrique du message par un ID de demande arbitraire. Par exemple, si vous publiez un message dans `kinesisfirehose/message/binary/request123`, la propriété `id` dans l'objet de réponse est définie sur `request123`.

Si vous ne souhaitez pas mapper une demande à une réponse, vous pouvez publier vos messages dans `kinesisfirehose/message/binary/`. Veillez à inclure la barre oblique de fin (/).

Données de sortie

Ce connecteur publie des informations d'état sous forme de données de sortie dans une rubrique MQTT.

Versions 2 - 5

Filtre de rubrique dans l'abonnement

`kinesisfirehose/message/status`

Exemple de sortie

La réponse contient le statut de chaque enregistrement de données envoyé dans le lot.

```
{
    "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**
>
> Si le connecteur détecte une erreur réessayable (par exemple, des erreurs de connexion), il réessaie de publier dans le lot suivant. Le recul exponentiel est géré par le AWS SDK. Les demandes qui échouent avec des erreurs pouvant être retentées sont ajoutées à la fin de la file d'attente pour la publication.

Version 1

Filtre de rubrique dans l'abonnement

```
kinesisfirehose/message/status
```

Exemple de sortie : réussite

```
{
   "response": {
       "firehose_record_id": "1lxfuuuFomkpJYzt/34ZU/r8JYPf8Wyf7AXqlXm",
       "status": "success"
    },
    "id": "request123"
}
```

Exemple de sortie : échec

```
{
   "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"
}
```

Exemple d'utilisation

Suivez les étapes de haut niveau suivantes pour configurer un exemple de fonction Lambda en Python 3.7 que vous pouvez utiliser pour tester le connecteur.

> **Note**
>
> - Si vous utilisez d'autres environnements d'exécution Python, vous pouvez créer un lien symbolique entre Python3.x et Python 3.7.
> - Les rubriques Démarrer avec les connecteurs (console) et Démarrer avec les connecteurs (CLI) contiennent des étapes détaillées qui vous montrent comment configurer et déployer un exemple de connecteur Twilio Notifications.

1. Veillez à répondre aux conditions requises pour le connecteur.

   Pour l'exigence de rôle de groupe, vous devez configurer le rôle de manière à accorder les autorisations requises et à vous assurer que le rôle a été ajouté au groupe. Pour plus d'informations, consultez the section called “Gérer le rôle de groupe (console)” ou the section called “Gérer le rôle de groupe (interface de ligne de commande)”.
2. Créez et publiez une fonction Lambda qui envoie des données d'entrée au connecteur.

   Enregistrez l'exemple de code en tant que fichier PY. Téléchargez et décompressez le SDK AWS IoT Greengrass de base pour Python. Ensuite, créez un package zip contenant le fichier PY et le dossier `greengrasssdk` au niveau racine. Ce package zip est le package de déploiement vers lequel vous effectuez le téléchargement AWS Lambda.

   Après avoir créé la fonction Lambda de Python 3.7, publiez une version de la fonction et créez un alias.
3. Configurez votre groupe Greengrass.

a. Ajoutez la fonction Lambda par son alias (recommandé). Configurez le cycle de vie Lambda comme étant de longue durée (ou dans `"Pinned": true` la CLI).

b. Ajoutez le connecteur et configurez ses paramètres.

c. Ajoutez des abonnements qui permettent au connecteur de recevoir des données d'entrée JSON et d'envoyer des données de sortie sur des filtres de rubrique pris en charge.

   - Définissez la fonction Lambda comme source, le connecteur comme cible et utilisez un filtre de rubrique d'entrée compatible.
   - Définissez le connecteur en tant que source, AWS IoT Core en tant que cible et utilisez un filtre de rubrique de sortie pris en charge. Vous utilisez cet abonnement pour afficher les messages d'état dans la AWS IoT console.

4. Déployez le groupe.

5. Dans la AWS IoT console, sur la page Test, abonnez-vous à la rubrique relative aux données de sortie pour consulter les messages d'état provenant du connecteur. L'exemple de fonction Lambda a une longue durée de vie et commence à envoyer des messages immédiatement après le déploiement du groupe.

   Lorsque vous avez terminé les tests, vous pouvez définir le cycle de vie Lambda à la demande (ou `"Pinned": false` dans la CLI) et déployer le groupe. Cela empêche la fonction d'envoyer des messages.

Exemple

L'exemple suivant de fonction Lambda envoie un message d'entrée au connecteur. Ce message contient des données 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
```

Licences

Le connecteur Kinesis Firehose inclut les logiciels/licences tiers suivants :

- AWS SDK for Python (Boto3)/Licence Apache 2.0
- botocore/Licence Apache 2.0
- dateutil/Licence PSF
- docutils/Licence BSD, licence GPL (General Public License) GNU, licence Python Software Foundation, domaine public
- jmespath/Licence MIT
- s3transfer/Licence Apache 2.0
- urllib3/Licence MIT

Ce connecteur est publié dans le cadre du contrat de licence logicielle Greengrass Core.

Journal des modifications

Le tableau suivant décrit les modifications apportées à chaque version du connecteur.

Version	Modifications
5	Ajout du paramètre `IsolationMode` pour configurer le mode de conteneurisation du connecteur.

Version	Modifications
4	Mise à niveau de l'environnement d'exécution Lambda vers Python 3.7, ce qui modifie les exigences d'exécution.
3	Correctif permettant de réduire la journalisation excessive et d'autres correctifs de bogues mineurs.
2	Ajout du support pour l'envoi d'enregistrements de données par lots à Firehose à un intervalle spécifié. • Requiert également l'action `firehose:PutRecordBatch` dans le rôle de groupe. • Nouveaux paramètres `MemorySize`, `DeliveryStreamQueueSize` et `PublishInterval`. • Le message de sortie contient un tableau des réponses d'état pour les enregistrements de données publiés.
1	Première version.

Un groupe Greengrass ne peut contenir qu'une seule version du connecteur à la fois. Pour de plus amples informations sur la mise à niveau d'une version de connecteur, veuillez consulter the section called “Mise à niveau des versions du connecteur”.

Consultez aussi

- Intégrer à des services et protocoles à l'aide de connecteurs
- the section called “Démarrer avec les connecteurs (console)”
- the section called “Démarrer avec les connecteurs (CLI)”
- Qu'est-ce qu'Amazon Kinesis Data Firehose ? dans le manuel Amazon Kinesis Developer Guide

Connecteur ML Feedback

> **Warning**
>
> Ce connecteur a été déplacé dans lePhase de vie prolongée, etAWS IoT Greengrassne publiera pas de mises à jour fournissant des fonctionnalités, des améliorations aux fonctionnalités existantes, des correctifs de sécurité ou des corrections de bugs. Pour plus d'informations, consultez AWS IoT Greengrass Version 1politique de maintenance.

Le connecteur ML Feedback facilite l'accès aux données de votre modèle d'apprentissage-machine pour la reformation et l'analyse du modèle. Le connecteur :

- Charge les données d'entrée (exemples) utilisées par votre modèle d'apprentissage-machine dans Amazon S3. L'entrée du modèle peut être dans n'importe quel format, tel que des images, JSON ou audio. Une fois les échantillons chargés dans le cloud, vous pouvez les utiliser pour reformer le modèle afin d'améliorer l'exactitude et la précision de ses prédictions. Par exemple, vous pouvez utiliserSageMaker Ground Truthpour étiqueter vos échantillons etSageMakerpour recycler le modèle.
- Publie les résultats des prédictions à partir du modèle sous forme de messages MQTT. Cela vous permet de surveiller et d'analyser la qualité d'inférence de votre modèle en temps réel. Vous pouvez également stocker les résultats des prédictions et les utiliser pour analyser les tendances au fil du temps.
- Publie des métriques sur des exemples de chargements et des exemples de données vers Amazon CloudWatch.

Pour configurer ce connecteur, vous décrivez vos configurations de commentaires prises en charge au format JSON. Une configuration de commentaire définit des propriétés telles que le compartiment Amazon S3 de destination, le type de contenu etstratégie d'échantillonnage. (Une stratégie d'échantillonnage est utilisée pour déterminer les échantillons à charger.)

Vous pouvez utiliser le connecteur ML Feedback dans les scénarios suivants :

- Avec les fonctions Lambda définies par l'utilisateur Les fonctions Lambda d'inférence locales utilisent leAWS IoT GreengrassMachine Learning SDK pour appeler ce connecteur et transmettre la configuration des commentaires cibles, l'entrée du modèle et la sortie du modèle (résultats de prédiction). Pour voir un exemple, consultez the section called “Exemple d'utilisation”.

- Avec l'Connecteur ML Image Classification(v2). Pour utiliser ce connecteur avec le connecteur ML Image Classification, configurez le`MLFeedbackConnectorConfigId`pour le connecteur ML Image Classification.
- Avec l'Connecteur de détection d'objets ML. Pour utiliser ce connecteur avec le connecteur ML Object Detection, configurez le`MLFeedbackConnectorConfigId`pour le connecteur ML Object Detection.

ARN : `arn:aws:greengrass:region::/connectors/MLFeedback/versions/1`

Prérequis

Ce connecteur possède les critères suivants :

- AWS IoT GreengrassCore Software v1.9.3 ou version ultérieure.
- PythonLa version 3.7 ou 3.8 installée sur l'appareil principal et ajoutée à la variable d'environnement PATH.

> **Note**
>
> Pour utiliser Python 3.8, exécutez la commande suivante pour créer un lien symbolique à partir du dossier d'installation Python 3.7 par défaut vers les binaires Python 3.8 installés.
>
> ```
> sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
> ```
>
> Ceci configure votre appareil de sorte qu'il réponde aux exigence de Python pour AWS IoT Greengrass.

- Un ou plusieurs compartiments Amazon S3. Le nombre de compartiments que vous utilisez dépend de votre stratégie d'échantillonnage.
- LeRôle de groupe Greengrassconfiguré pour autoriser les`s3:PutObject`action sur des objets dans le compartiment Amazon S3 de destination, comme indiqué dans l'exemple de stratégie IAM suivant.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
```

```
            "Action": "s3:PutObject",
            "Resource": [
                "arn:aws:s3:::bucket-name/*"
            ]
        }
    ]
}
```

La stratégie doit inclure tous les compartiments de destination en tant que ressources. Vous pouvez octroyer un accès précis ou conditionnel aux ressources (par exemple, en utilisant un schéma d'attribution de nom avec caractère générique *).

Pour l'exigence de rôle de groupe, vous devez configurer le rôle de manière à accorder les autorisations requises et à vous assurer que le rôle a été ajouté au groupe. Pour plus d'informations, consultez the section called “Gérer le rôle de groupe (console)” ou the section called “Gérer le rôle de groupe (interface de ligne de commande)”.

- LeConnecteur CloudWatch Metricsajouté au groupe Greengrass et configuré. Cette opération est obligatoire uniquement si vous souhaitez utiliser la fonction de création de rapports de métriques.
- AWS IoT GreengrassKit SDK de Machine Learningv1.1.0 est nécessaire pour interagir avec ce connecteur.

Paramètres

`FeedbackConfigurationMap`

Ensemble d'une ou plusieurs configurations de commentaire que le connecteur peut utiliser pour charger des exemples dans Amazon S3. Une configuration de commentaire définit des paramètres tels que le compartiment de destination, le type de contenu et la stratégie d'échantillonnage. Lorsque ce connecteur est appelé, la fonction ou le connecteur Lambda appelante spécifie une configuration de commentaire cible.

Nom d'affichage dans leAWS IoTConsole : Mappage de configuration des commentaires

Obligatoire`true`

Type : Une chaîne JSON bien formée qui définit l'ensemble des configurations de commentaire prises en charge. Pour voir un exemple, consultez the section called “Exemple FeedbackConfigurationMap”.

L'ID d'un objet de configuration de commentaire présente les exigences suivantes.

ID :

- Doit être unique parmi les objets de configuration.
- Doit commencer par une lettre ou un chiffre. Les noms de compartiments peuvent contenir des lettres minuscules, des chiffres et des traits d'union.
- Nombre maximal de caractères : 2 à 63.

Obligatoire`true`

Type: `string`

Modèle valide :`^[a-zA-Z0-9][a-zA-Z0-9-]{1,62}$`

Exemples : `MyConfig0`, `config-a`, `12id`

Le corps d'un objet de configuration de commentaire contient les propriétés suivantes.

`s3-bucket-name`

Nom du compartiment Amazon S3 de destination.

> **Note**
>
> Le rôle de groupe doit autoriser l'action `s3:PutObject` sur tous les compartiments de destination. Pour plus d'informations, consultez the section called “Prérequis”.

Obligatoire`true`

Type: `string`

Modèle valide :`^[a-z0-9\.\-]{3,63}$`

`content-type`

Type de contenu des exemples à charger. Tout le contenu d'une configuration de commentaire individuelle doit être du même type.

Obligatoire`true`

Type: `string`

Exemples : `image/jpeg`, `application/json`, `audio/ogg`

`s3-prefix`

Préfixe de clé à utiliser pour les exemples chargés. Un préfixe est similaire à un nom de répertoire. Il vous permet de stocker des données similaires dans le même répertoire d'un compartiment. Pour de plus amples informations, veuillez consulterClé d'objet et métadonnéesdans leGuide de l'utilisateur Amazon Simple Storage Service.

Obligatoire`false`

Type: `string`

`file-ext`

Extension de fichier à utiliser pour les exemples chargés. Doit être une extension de fichier valide pour le type de contenu.

Obligatoire`false`

Type: `string`

Exemples : `jpg`, `json`, `ogg`

`sampling-strategy`

Stratégie d'échantillonnage à utiliser pour filtrer les exemples à charger. S'il n'est pas spécifié, le connecteur essaie de charger tous les exemples qu'il reçoit.

Obligatoire`false`

Type : Une chaîne JSON bien formée qui contient les propriétés suivantes.

`strategy-name`

Nom de la stratégie d'échantillonnage.

Obligatoire`true`

Type: `string`

Valeurs valides : `RANDOM_SAMPLING`, `LEAST_CONFIDENCE`, `MARGIN` ou `ENTROPY`

`rate`

Taux de la stratégie d'échantillonnage aléatoire .

Obligatoiretruesistrategy-nameestRANDOM_SAMPLING.

Type: number

Valeurs valides : 0.0 - 1.0

threshold

Seuil de la stratégie d'échantillonnage Least Confidence, Margin ou Entropy .

Obligatoiretruesistrategy-nameestLEAST_CONFIDENCE,MARGIN, ouENTROPY.

Type: number

Valeurs valides :

- 0.0 - 1.0 pour la stratégie LEAST_CONFIDENCE ou MARGIN.
- 0.0 - no limit pour la stratégie ENTROPY.

RequestLimit

Nombre maximal de demandes que le connecteur peut traiter à la fois.

Vous pouvez utiliser ce paramètre pour limiter la consommation de mémoire en limitant le nombre de demandes traitées par le connecteur en même temps. Les demandes qui dépassent cette limite sont ignorées.

Nom d'affichage dans leAWS IoTConsole : Limite de demande

Obligatoirefalse

Type: string

Valeurs valides : 0 - 999

Modèle valide :^$|^[0-9]{1,3}$

Exemple de création de connecteur (AWS CLI)

La commande CLI suivante crée uneConnectorDefinitionavec une version initiale qui contient le connecteur 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"
            }
        }
    ]
}'
```

Exemple FeedbackConfigurationMap

Voici un exemple de valeur développé pour le paramètre `FeedbackConfigurationMap`. Cet exemple inclut plusieurs configurations de commentaires qui utilisent différentes stratégies d'échantillonnage.

```
{
    "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"
    }
}
```

Stratégies d'échantillonnage

Le connecteur prend en charge quatre stratégies d'échantillonnage qui déterminent s'il convient de charger les échantillons transmis au connecteur. Les exemples sont des instances de données discrètes qu'un modèle utilise pour une prédiction. Vous pouvez utiliser des stratégies d'échantillonnage pour filtrer les échantillons les plus susceptibles d'améliorer la précision du modèle.

`RANDOM_SAMPLING`

Charge de façon aléatoire les exemples en fonction du débit fourni. Il charge un exemple si une valeur générée de façon aléatoire est inférieure à la fréquence. Plus le taux est élevé, plus le nombre d'échantillons chargés est élevé.

> **Note**
>
> Cette stratégie ignore toute prédiction de modèle fournie.

`LEAST_CONFIDENCE`

Charge des exemples dont la probabilité de fiabilité maximale est inférieure au seuil fourni.

Exemple de scénario

Seuil : `.6`

Prédiction du modèle : `[.2, .2, .4, .2]`

Probabilité de fiabilité maximale : `.4`

Résultat:

Utilisez l'exemple, car la probabilité de fiabilité maximale (`.4`) **<=** seuil (`.6`).

`MARGIN`

Charge des exemples si la marge entre les deux principales probabilités de confiance se situe dans le seuil fourni. La marge correspond à la différence entre les deux principales probabilités.

Exemple de scénario

Seuil : `.02`

Prédiction du modèle : `[.3, .35, .34, .01]`

Les deux principales probabilités de confiance : `[.35, .34]`

Marge : `.01` (`.35 - .34`)

Résultat:

Utilisez l'exemple car marge (`.01`) **<=** seuil (`.02`).

`ENTROPY`

Charge des exemples dont l'entropie est supérieure au seuil fourni. Utilise l'entropie normalisée de la prédiction du modèle.

Exemple de scénario

Seuil : `0.75`

Prédiction du modèle : `[.5, .25, .25]`

Entropie pour la prédiction : `1.03972`

Résultat:

Utilisez sample car entropy (`1.03972`) > threshold (`0.75`).

Données d'entrée

Les fonctions Lambda définies par l'utilisateur utilisent le`publish`fonction de la`feedback`client dans leAWS IoT GreengrassSDK Machine Learning pour appeler le connecteur. Pour voir un exemple, consultez the section called “Exemple d'utilisation”.

> **Note**
>
> Ce connecteur n'accepte pas les messages MQTT comme données d'entrée.

La fonction `publish` accepte les arguments suivants :

ConfigId

ID de la configuration des commentaires cibles. Cela doit correspondre à l'ID d'une configuration de commentaire définie dans leFeedbackConfigurationMappour le connecteur ML Feedback.

Obligatoire

Type : chaîne

ModelInput

Données d'entrée qui ont été transmises à un modèle pour l'inférence. Ces données d'entrée sont chargées à l'aide de la configuration cible, sauf si elles sont filtrées en fonction de la stratégie d'échantillonnage.

Obligatoire

Type : octets

ModelPrediction

La prédiction est le résultat du modèle. Le type de résultat peut être un dictionnaire ou une liste. Par exemple, les résultats de prédiction du connecteur ML Image Classification sont une liste de probabilités (par exemple,`[0.25, 0.60, 0.15]`). Ces données sont publiées dans la rubrique `/feedback/message/prediction`.

Obligatoire

Type : dictionnaire ou liste de`float`valeurs

Metadonnées

Métadonnées spécifiques à l'application définies par le client qui sont attachées à l'exemple chargé et publiées dans la rubrique `/feedback/message/prediction`. Le connecteur insère également une clé `publish-ts` avec une valeur d'horodatage dans les métadonnées.

Obligatoire : false

Type : dictionnaire

Exemple : `{"some-key": "some value"}`

Données de sortie

Ce connecteur publie les données dans trois rubriques MQTT :

- Informations sur le statut à partir du connecteur dans la rubrique `feedback/message/status`.
- Résultats des prédictions sur la rubrique `feedback/message/prediction`.
- Mesures destinées à CloudWatch sur le`cloudwatch/metric/put`sujet.

Vous devez configurer les abonnements pour autoriser le connecteur à communiquer sur les rubriques MQTT. Pour plus d'informations, consultez the section called “Entrées et sorties”.

Filtre de rubriques : `feedback/message/status`

Utilisez cette rubrique pour surveiller le statut des exemples de chargement et d'exemples supprimés. Le connecteur publie dans cette rubrique chaque fois qu'il reçoit une demande.

Exemple de sortie : Exemple de chargement réussi

```
{
  "response": {
    "status": "success",
    "s3_response": {
      "ResponseMetadata": {
        "HostId": "IOWQ4fDEXAMPLEQM+ey7N9WgVhSnQ6JEXAMPLEZb7hSQDASK
+Jd1vEXAMPLEa3Km",
        "RetryAttempts": 1,
```

```
        "HTTPStatusCode": 200,
        "RequestId": "79104EXAMPLEB723",
        "HTTPHeaders": {
          "content-length": "0",
          "x-amz-id-2":
 "lbbqaDVFOhMlyU3gRvAX1ZIdg8P0WkGkCSSFsYFvSwLZk3j7QZhG5EXAMPLEdd4/pEXAMPLEUqU=",
          "server": "AmazonS3",
          "x-amz-expiration": "expiry-date=\"Wed, 17 Jul 2019 00:00:00 GMT\",
 rule-id=\"OGZjYWY3OTgtYWI2Zi00ZDllLWE4YmQtNzMyYzEXAMPLEoUw\"",
          "x-amz-request-id": "79104EXAMPLEB723",
          "etag": "\"b9c4f172e64458a5fd674EXAMPLE5628\"",
          "date": "Thu, 11 Jul 2019 00:12:50 GMT",
          "x-amz-server-side-encryption": "AES256"
        }
      },
      "bucket": "greengrass-feedback-connector-data-us-west-2",
      "ETag": "\"b9c4f172e64458a5fd674EXAMPLE5628\"",
      "Expiration": "expiry-date=\"Wed, 17 Jul 2019 00:00:00 GMT\", rule-id=
\"OGZjYWY3OTgtYWI2Zi00ZDllLWE4YmQtNzMyYzEXAMPLEoUw\"",
      "key": "s3-key-prefix/UUID.file_ext",
      "ServerSideEncryption": "AES256"
    }
  },
  "id": "5aaa913f-97a3-48ac-5907-18cd96b89eeb"
}
```

Le connecteur ajoute le`bucket`et`key`champs de réponse d'Amazon S3. Pour plus d'informations sur la réponse Amazon S3, consultezPUT Objectdans leRéférence des API Amazon Simple Storage Service.

Exemple de sortie : Exemple supprimé en raison de la stratégie d'échantillonnage

```
{
  "response": {
    "status": "sample_dropped_by_strategy"
  },
  "id": "4bf5aeb0-d1e4-4362-5bb4-87c05de78ba3"
}
```

Exemple de sortie : Échec du chargement d'exemple

Un statut d'échec inclut le message d'erreur comme valeur `error_message` et la classe d'exception comme valeur `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"
}
```

Exemple de sortie : Demande limitée en raison de la limite de demande

```
{
  "response": {
    "status": "fail",
    "error_message": "Request limit has been reached (max request: 10 ). Dropping
 request.",
    "error": "Queue.Full"
  },
  "id": "4bf5aeb0-d1e4-4362-5bb4-87c05de78ba3"
}
```

Filtre de rubriques : `feedback/message/prediction`

Utilisez cette rubrique pour écouter les prédictions basées sur des exemples de données chargés. Cela vous permet d'analyser les performances de votre modèle en temps réel. Les prédictions de modèle sont publiées dans cette rubrique uniquement si les données sont correctement chargées vers Amazon S3. Les messages publiés dans cette rubrique sont au format JSON. Ils contiennent le lien vers l'objet de données chargé, la prédiction du modèle et les métadonnées incluses dans la demande.

Vous pouvez également stocker les résultats des prédictions et les utiliser pour signaler et analyser les tendances au fil du temps. Les tendances peuvent fournir des informations précieuses. Par exemple, une tendance de précision décroissante au fil du temps peut vous aider à décider si le modèle doit être reformé.

Exemple de sortie

```
{
```

```
  "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**
>
> Vous pouvez configurer leConnecteur IoT Analyticspour vous abonner à ce sujet et envoyer les informations àAWS IoT Analyticspour une analyse plus poussée ou historique.

Filtre de rubriques : `cloudwatch/metric/put`

Il s'agit de la rubrique de sortie utilisée pour publier des métriques dans CloudWatch. Cette fonction nécessite que vous installiez et configuriez leConnecteur CloudWatch Metrics.

Les métriques incluent :

- Nombre d'échantillons chargés.
- Taille des échantillons chargés.
- Nombre d'erreurs depuis les chargements vers Amazon S3.
- Nombre d'échantillons abandonnés en fonction de la stratégie d'échantillonnage.
- Nombre de demandes limitées.

Exemple de sortie : Taille de l'échantillon de données (publié avant le chargement réel)

```
{
  "request": {
    "namespace": "GreengrassFeedbackConnector",
    "metricData": {
      "value": 47592,
```

```
      "unit": "Bytes",
      "metricName": "SampleSize"
    }
  }
}
```

Exemple de sortie : Exemple de chargement réussi

```
{
  "request": {
    "namespace": "GreengrassFeedbackConnector",
    "metricData": {
      "value": 1,
      "unit": "Count",
      "metricName": "SampleUploadSuccess"
    }
  }
}
```

Exemple de sortie : L'exemple de chargement réussi et résultat de prédiction publié

```
{
  "request": {
    "namespace": "GreengrassFeedbackConnector",
    "metricData": {
      "value": 1,
      "unit": "Count",
      "metricName": "SampleAndPredictionPublished"
    }
  }
}
```

Exemple de sortie : Échec du chargement d'exemple

```
{
  "request": {
    "namespace": "GreengrassFeedbackConnector",
    "metricData": {
      "value": 1,
      "unit": "Count",
      "metricName": "SampleUploadFailure"
    }
  }
```

```
}
```

Exemple de sortie : Exemple supprimé en raison de la stratégie d'échantillonnage

```
{
  "request": {
    "namespace": "GreengrassFeedbackConnector",
    "metricData": {
      "value": 1,
      "unit": "Count",
      "metricName": "SampleNotUsed"
    }
  }
}
```

Exemple de sortie : Demande limitée en raison de la limite de demande

```
{
  "request": {
    "namespace": "GreengrassFeedbackConnector",
    "metricData": {
      "value": 1,
      "unit": "Count",
      "metricName": "ErrorRequestThrottled"
    }
  }
}
```

Exemple d'utilisation

L'exemple suivant est une fonction Lambda définie par l'utilisateur qui utilise le kitAWS IoT GreengrassKit SDK de Machine Learningpour envoyer des données au connecteur ML Feedback.

> **Note**
>
> Vous pouvez télécharger leAWS IoT GreengrassSDK Machine Learning à partir duAWS IoT Greengrass page des téléchargements.

```
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
```

Licences

Le connecteur inclut les logiciels et licences tiers suivants :

- AWS SDK for Python (Boto3)/Licence Apache 2.0
- botocore/Licence Apache 2.0
- dateutil/Licence PSF
- docutils/Licence BSD, licence GPL (General Public License) GNU, licence Python Software Foundation, domaine public
- jmespath/Licence MIT
- s3transfer/Licence Apache 2.0
- urllib3/Licence MIT

- six/MIT

Ce connecteur est libéré sous leContrat de licence du logiciel Greengrass Core.

Consulter aussi

- Intégrer à des services et protocoles à l'aide de connecteurs
- the section called “Démarrer avec les connecteurs (console)”
- the section called “Démarrer avec les connecteurs (CLI)”

Connecteur de classification d'images ML

> **Warning**
>
> Ce connecteur est passé à la phase de durée de vie prolongée et AWS IoT Greengrass ne publiera pas de mises à jour fournissant des fonctionnalités, des améliorations aux fonctionnalités existantes, des correctifs de sécurité ou des corrections de bogues. Pour plus d'informations, veuillez consulter AWS IoT Greengrass Version 1politique de maintenance.

Les connecteurs ML Image Classification fournissent un service d'inférence d'apprentissage automatique (ML) qui s'exécute sur le AWS IoT Greengrass cœur. Ce service d'inférence local

effectue la classification des images à l'aide d'un modèle entraîné par l'algorithme de classification des SageMaker images.

Les fonctions Lambda définies par l'utilisateur utilisent le SDK Machine AWS IoT Greengrass Learning pour envoyer des demandes d'inférence au service d'inférence local. Le service exécute l'inférence localement et renvoie les probabilités que l'image d'entrée appartient à certaines catégories.

AWS IoT Greengrass fournit les versions suivantes de ce connecteur, qui est disponible pour plusieurs plateformes.

Version 2

Connecteur	Description et ARN
Classification d'images XML Aarch64 JTX2	Service d'inférence de classification d'images pour NVIDIA Jetson TX2. Prend en charge l'accélération GPU. ARN : `arn:aws:greengrass:` *`region`*`::/connectors/ImageClassificationAarch64JTX2/versions/2`
Classification d'images ML x86_64	Service d'inférence de classification d'images pour les plateformes x86_64. ARN : `arn:aws:greengrass:` *`region`*`::/connectors/ImageClassificationx86-64/versions/2`
Classification d'images ML ARMv7	Service d'inférence de classification d'images pour les plateformes ARMv7. ARN : `arn:aws:greengrass:` *`region`*`::/connectors/ImageClassificationARMv7/versions/2`

Version 1

Connecteur	Description et ARN
Classification d'images XML Aarch64 JTX2	Service d'inférence de classification d'images pour NVIDIA Jetson TX2. Prend en charge l'accélération GPU. ARN : `arn:aws:greengrass:region::/connectors/ImageClassificationAarch64JTX2/versions/1`
Classification d'images ML x86_64	Service d'inférence de classification d'images pour les plateformes x86_64. ARN : `arn:aws:greengrass:region::/connectors/ImageClassificationx86-64/versions/1`
Classification d'images ML Armv7	Service d'inférence de classification d'images pour les plateformes Armv7. ARN : `arn:aws:greengrass:region::/connectors/ImageClassificationARMv7/versions/1`

Pour obtenir des informations sur les changements apportés aux versions, veuillez consulter le Journal des modifications.

Prérequis

Ces connecteurs possèdent les exigences suivantes :

Version 2

- AWS IoT GreengrassCore Software v1.9.3 ou version ultérieure.

- Python version 3.7 ou 3.8 installé sur le périphérique principal et ajouté à la variable d'environnement PATH.

  > **Note**
  >
  > Pour utiliser Python 3.8, exécutez la commande suivante pour créer un lien symbolique entre le dossier d'installation par défaut de Python 3.7 et les fichiers binaires Python 3.8 installés.
  >
  > ```
  > sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
  > ```
  >
  > Ceci configure votre appareil de sorte qu'il réponde aux exigence de Python pour AWS IoT Greengrass.

- Dépendances pour l'infrastructure Apache MXNet installée sur l'appareil principal. Pour plus d'informations, veuillez consulter the section called “Installation des dépendances MXNet”.
- Une ressource ML du groupe Greengrass qui fait référence à une source de SageMaker modèle. Ce modèle doit être entraîné par l'algorithme de classification des SageMaker images. Pour plus d'informations, consultez la section Algorithme de classification des images dans le manuel Amazon SageMaker Developer Guide.
- Le connecteur ML Feedback a été ajouté au groupe Greengrass et configuré. Cette opération est obligatoire uniquement si vous souhaitez utiliser le connecteur pour charger les données d'entrée de modèle et publier les prédictions dans une rubrique MQTT.
- Le rôle de groupe Greengrass est configuré pour autoriser l'`sagemaker:DescribeTrainingJob`action sur la tâche de formation cible, comme illustré dans l'exemple de politique IAM suivant.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sagemaker:DescribeTrainingJob"
            ],
            "Resource": "arn:aws:sagemaker:region:account-id:training-
job:training-job-name"
        }
```

```
    ]
}
```

Pour l'exigence de rôle de groupe, vous devez configurer le rôle de manière à accorder les autorisations requises et à vous assurer que le rôle a été ajouté au groupe. Pour plus d'informations, consultez the section called “Gérer le rôle de groupe (console)” ou the section called “Gérer le rôle de groupe (interface de ligne de commande)”.

Vous pouvez octroyer un accès précis ou conditionnel aux ressources (par exemple, en utilisant un schéma d'attribution de nom avec caractère générique *). Si vous modifiez le poste de formation cible à l'avenir, veillez à mettre à jour le rôle du groupe.

- AWS IoT GreengrassLe SDK Machine Learning v1.1.0 est requis pour interagir avec ce connecteur.

Version 1

- AWS IoT GreengrassCore Software v1.7 ou version ultérieure.
- Python version 2.7 installé sur le périphérique principal et ajouté à la variable d'environnement PATH.
- Dépendances pour l'infrastructure Apache MXNet installée sur l'appareil principal. Pour plus d'informations, veuillez consulter the section called “Installation des dépendances MXNet”.
- Une ressource ML du groupe Greengrass qui fait référence à une source de SageMaker modèle. Ce modèle doit être entraîné par l'algorithme de classification des SageMaker images. Pour plus d'informations, consultez la section Algorithme de classification des images dans le manuel Amazon SageMaker Developer Guide.
- Le rôle de groupe Greengrass est configuré pour autoriser l'`sagemaker:DescribeTrainingJob`action sur la tâche de formation cible, comme illustré dans l'exemple de politique IAM suivant.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sagemaker:DescribeTrainingJob"
            ],
```

```
            "Resource": "arn:aws:sagemaker:region:account-id:training-
job:training-job-name"
        }
    ]
}
```

Pour l'exigence de rôle de groupe, vous devez configurer le rôle de manière à accorder les autorisations requises et à vous assurer que le rôle a été ajouté au groupe. Pour plus d'informations, consultez the section called “Gérer le rôle de groupe (console)” ou the section called “Gérer le rôle de groupe (interface de ligne de commande)”.

Vous pouvez octroyer un accès précis ou conditionnel aux ressources (par exemple, en utilisant un schéma d'attribution de nom avec caractère générique *). Si vous modifiez le poste de formation cible à l'avenir, veillez à mettre à jour le rôle du groupe.

- AWS IoT GreengrassLe SDK Machine Learning v1.0.0 ou version ultérieure est requis pour interagir avec ce connecteur.

Paramètres du connecteur

Ces connecteurs fournissent les paramètres suivants.

Version 2

`MLModelDestinationPath`

Le chemin local absolu de la ressource ML dans l'environnement Lambda. Il s'agit du chemin de destination spécifié pour la ressource de ML.

> **Note**
>
> Si vous avez créé la ressource de ML dans la console, il s'agit du chemin d'accès local.

Nom d'affichage dans la AWS IoT console : chemin de destination du modèle

Nécessaire : `true`

Type: `string`

Schéma valide : `.+`

`MLModelResourceId`

ID de la ressource de ML qui référence le modèle de source.

Nom affiché dans la AWS IoT console : ressource ARN du SageMaker job

Nécessaire : `true`

Type: `string`

Schéma valide : `[a-zA-Z0-9:_-]+`

`MLModelSageMakerJobArn`

L'ARN de la tâche de SageMaker formation qui représente la source du SageMaker modèle. Le modèle doit être entraîné par l'algorithme de classification des SageMaker images.

Nom affiché dans la AWS IoT console : SageMaker job ARN

Nécessaire : `true`

Type: `string`

Schéma valide : `^arn:aws:sagemaker:[a-zA-Z0-9-]+:[0-9]+:training-job/[a-zA-Z0-9][a-zA-Z0-9-]+$`

`LocalInferenceServiceName`

Nom du service d'inférence local. Les fonctions Lambda définies par l'utilisateur appellent le service en transmettant le nom à `invoke_inference_service` la fonction du SDK Machine AWS IoT Greengrass Learning. Pour voir un exemple, consultez the section called “Exemple d'utilisation”.

Nom affiché dans la AWS IoT console : nom du service d'inférence local

Nécessaire : `true`

Type: `string`

Schéma valide : `[a-zA-Z0-9][a-zA-Z0-9-]{1,62}`

`LocalInferenceServiceTimeoutSeconds`

Durée (en secondes) avant laquelle la demande d'inférence est interrompue. La valeur minimale est de 1.

Nom affiché dans la AWS IoT console : Timeout (seconde)

Nécessaire : `true`

Type: `string`

Schéma valide : `[1-9][0-9]*`

`LocalInferenceServiceMemoryLimitKB`

Quantité de mémoire (en Ko) à laquelle le service a accès. La valeur minimale est de 1.

Nom affiché dans la AWS IoT console : Limite de mémoire (Ko)

Nécessaire : `true`

Type: `string`

Schéma valide : `[1-9][0-9]*`

`GPUAcceleration`

Contexte de calcul de l'UC ou du GPU (accéléré). Cette propriété s'applique uniquement au connecteur ML Image Classification Aarch64 JTX2.

Nom affiché dans la AWS IoT console : accélération du processeur graphique

Nécessaire : `true`

Type: `string`

Valeurs valides : `CPU` ou `GPU`

`MLFeedbackConnectorConfigId`

ID de la configuration de commentaire à utiliser pour charger les données d'entrée du modèle. Cela doit correspondre à l'ID d'une configuration de commentaire définie pour le connecteur ML Feedback.

Ce paramètre est obligatoire uniquement si vous souhaitez utiliser le connecteur ML Feedback pour charger les données d'entrée du modèle et publier des prédictions dans une rubrique MQTT.

Nom affiché dans la AWS IoT console : ID de configuration du connecteur ML Feedback

Nécessaire : `false`

Type: `string`

Schéma valide : `^$|^[a-zA-Z0-9][a-zA-Z0-9-]{1,62}$`

Version 1

`MLModelDestinationPath`

Le chemin local absolu de la ressource ML dans l'environnement Lambda. Il s'agit du chemin de destination spécifié pour la ressource de ML.

> **Note**
>
> Si vous avez créé la ressource de ML dans la console, il s'agit du chemin d'accès local.

Nom d'affichage dans la AWS IoT console : chemin de destination du modèle

Nécessaire : `true`

Type: `string`

Schéma valide : `.+`

`MLModelResourceId`

ID de la ressource de ML qui référence le modèle de source.

Nom affiché dans la AWS IoT console : ressource ARN du SageMaker job

Nécessaire : `true`

Type: `string`

Schéma valide : `[a-zA-Z0-9:_-]+`

`MLModelSageMakerJobArn`

L'ARN de la tâche de SageMaker formation qui représente la source du SageMaker modèle. Le modèle doit être entraîné par l'algorithme de classification des SageMaker images.

Nom affiché dans la AWS IoT console : SageMaker job ARN

Nécessaire : `true`

Type: `string`

Schéma valide : `^arn:aws:sagemaker:[a-zA-Z0-9-]+:[0-9]+:training-job/[a-zA-Z0-9][a-zA-Z0-9-]+$`

`LocalInferenceServiceName`

Nom du service d'inférence local. Les fonctions Lambda définies par l'utilisateur appellent le service en transmettant le nom à `invoke_inference_service` la fonction du SDK Machine AWS IoT Greengrass Learning. Pour voir un exemple, consultez the section called “Exemple d'utilisation”.

Nom affiché dans la AWS IoT console : nom du service d'inférence local

Nécessaire : `true`

Type: `string`

Schéma valide : `[a-zA-Z0-9][a-zA-Z0-9-]{1,62}`

`LocalInferenceServiceTimeoutSeconds`

Durée (en secondes) avant laquelle la demande d'inférence est interrompue. La valeur minimale est de 1.

Nom affiché dans la AWS IoT console : Timeout (seconde)

Nécessaire : `true`

Type: `string`

Schéma valide : `[1-9][0-9]*`

`LocalInferenceServiceMemoryLimitKB`

Quantité de mémoire (en Ko) à laquelle le service a accès. La valeur minimale est de 1.

Nom affiché dans la AWS IoT console : Limite de mémoire (Ko)

Nécessaire : `true`

Type: `string`

Schéma valide : `[1-9][0-9]*`

`GPUAcceleration`

Contexte de calcul de l'UC ou du GPU (accéléré). Cette propriété s'applique uniquement au connecteur ML Image Classification Aarch64 JTX2.

Nom affiché dans la AWS IoT console : accélération du processeur graphique

Nécessaire : `true`

Type: `string`

Valeurs valides : `CPU` ou `GPU`

Exemple de création de connecteur (AWS CLI)

Les commandes CLI suivantes créent un `ConnectorDefinition` avec une version initiale contenant un connecteur de classification d'images ML.

Exemple : instance d'UC

Cet exemple crée une instance du connecteur ARMv7L ML Image Classification.

```
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"
            }
        }
    ]
}'
```

Exemple : instance de GPU

Cet exemple crée une instance du connecteur ML Image Classification Aarch64 JTX2, qui prend en charge l'accélération du GPU sur une carte 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 fonction Lambda de ces connecteurs a une longue durée de vie.

Dans la AWS IoT Greengrass console, vous pouvez ajouter un connecteur depuis la page Connecteurs du groupe. Pour plus d'informations, veuillez consulter the section called “Démarrer avec les connecteurs (console)”.

Données d'entrée

Ces connecteurs acceptent un fichier image comme entrée. Les fichiers image d'entrée doivent être au format `jpeg` ou `png`. Pour plus d'informations, veuillez consulter the section called “Exemple d'utilisation”.

Ces connecteurs n'acceptent pas les messages MQTT comme données d'entrée.

Données de sortie

Ces connecteurs renvoient une prédiction formatée pour l'objet identifié dans l'image d'entrée :

```
[0.3,0.1,0.04,...]
```

La prédiction contient une liste de valeurs qui correspondent aux catégories utilisées dans l'ensemble de données de formation pendant la formation du modèle. Chaque valeur représente la probabilité que l'image relève de la catégorie correspondante. La catégorie avec la plus haute probabilité est la prédiction dominante.

Ces connecteurs ne publient pas les messages MQTT sous forme de données de sortie.

Exemple d'utilisation

L'exemple de fonction Lambda suivant utilise le SDK AWS IoT Greengrass Machine Learning pour interagir avec un connecteur ML Image Classification.

> **Note**
>
> Vous pouvez télécharger le SDK depuis la page de téléchargement du SDK AWS IoT Greengrass Machine Learning.

L'exemple initialise un client de kit de développement logiciel et appelle de façon synchrone la fonction `invoke_inference_service` du SDK pour appeler le service d'inférence local. Il transmet le type d'algorithme, le nom de service, le type d'image et le contenu de l'image. Ensuite, l'exemple analyse la réponse du service pour obtenir les résultats probables (prédictions).

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` fonction du SDK AWS IoT Greengrass Machine Learning accepte les arguments suivants.

Argument	Description
`AlgoType`	Nom du type d'algorithme à utiliser pour l'inférence. Actuellement, seul `image-classification` est pris en charge. Nécessaire : `true` Type: `string`

Argument	Description
	Valeurs valides : `image-classification`
`ServiceName`	Nom du service d'inférence local. Utilisez le nom que vous avez spécifié pour le paramètre `LocalInferenceServiceName` lorsque vous avez configuré le connecteur. Nécessaire : `true` Type: `string`
`ContentType`	Nom du type mime de l'image d'entrée. Nécessaire : `true` Type: `string` Valeurs valides : `image/jpeg, image/png`
`Body`	Contenu du type mime de l'image d'entrée. Nécessaire : `true` Type: `binary`

Installation des dépendances MXNet sur AWS IoT Greengrass Core

Pour utiliser un connecteur de classification d'images ML, vous devez installer les dépendances du framework Apache MXnet sur le périphérique principal. Le connecteur utilise l'infrastructure pour servir le modèle ML.

> **Note**
>
> Ces connecteurs sont fournis avec une bibliothèque MXNet précompilée, de sorte que vous n'êtes pas obligé d'installer l'infrastructure MXNet elle-même sur l'appareil principal.

AWS IoT Greengrass fournit des scripts pour installer les dépendances pour les plateformes et les appareils courants suivants (ou à utiliser comme référence pour leur installation). Si vous utilisez une autre plateforme ou un autre appareil, consultez la documentation MXNet correspondant à votre configuration.

Avant d'installer les dépendances MXNet, assurez-vous que les bibliothèques système requises (avec les versions minimales spécifiées) sont présentes sur l'appareil.

NVIDIA Jetson TX2

1. Installer CUDA Toolkit 9.0 et cuDNN 7.0. Vous pouvez suivre les instructions de the section called “Configuration d'autres appareils” dans le didacticiel Démarrez.
2. Activez les référentiels univers de sorte que le connecteur peut installer les logiciels open source maintenus par la communauté. Pour plus d'informations, consultez Repositories/Ubuntu dans la documentation Ubuntu.

   a. Ouvrez le fichier `/etc/apt/sources.list`.

   b. Vérifiez que les lignes suivantes ne sont pas commentées :

   ```
   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. Enregistrez une copie du script d'installation suivant dans un fichier nommé `nvidiajtx2.sh` sur l'appareil principal.

   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**
>
> Si OpenCV ne s'installe pas correctement à l'aide de ce script, vous pouvez essayer de générer à partir de la source. Pour plus d'informations, consultez Installation sous Linux dans la documentation OpenCV ou consultez d'autres ressources en ligne pour votre plateforme.

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. Dans le répertoire où vous avez enregistré le fichier, exécutez la commande suivante :

```
sudo nvidiajtx2.sh
```

x86_64 (Ubuntu or Amazon Linux)

1. Enregistrez une copie du script d'installation suivant dans un fichier nommé `x86_64.sh` sur l'appareil principal.

   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**
>
> Si OpenCV ne s'installe pas correctement à l'aide de ce script, vous pouvez essayer de générer à partir de la source. Pour plus d'informations, consultez Installation sous Linux dans la documentation OpenCV ou consultez d'autres ressources en ligne pour votre plateforme.

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. Dans le répertoire où vous avez enregistré le fichier, exécutez la commande suivante :

```
sudo x86_64.sh
```

Armv7 (Raspberry Pi)

1. Enregistrez une copie du script d'installation suivant dans un fichier nommé `armv7l.sh` sur l'appareil principal.

   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**
>
> Si OpenCV ne s'installe pas correctement à l'aide de ce script, vous pouvez essayer de générer à partir de la source. Pour plus d'informations, consultez Installation sous Linux dans la documentation OpenCV ou consultez d'autres ressources en ligne pour votre plateforme.

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. Dans le répertoire où vous avez enregistré le fichier, exécutez la commande suivante :

```
sudo bash armv7l.sh
```

> **Note**
>
> Sur un Raspberry Pi, l'utilisation de `pip` pour installer des dépendances d'apprentissage automatique est une opération gourmande en mémoire qui peut entraîner un dépassement de la capacité mémoire de l'appareil et une absence de réponse. Pour contourner ce problème, vous pouvez augmenter temporairement l'espace d'échange :

Dans `/etc/dphys-swapfile`, augmentez la valeur de la variable `CONF_SWAPSIZE`, puis exécutez la commande suivante pour redémarrer `dphys-swapfile`.

```
/etc/init.d/dphys-swapfile restart
```

Journalisation et résolution des problèmes

Selon les paramètres de votre groupe, les journaux d'événements et d'erreurs sont écrits dans CloudWatch Logs, dans le système de fichiers local, ou dans les deux. Les journaux de ce connecteur utilisent le préfixe `LocalInferenceServiceName`. Si le connecteur se comporte de manière inattendue, vérifiez les journaux du connecteur. Ces derniers contiennent généralement des informations de débogage utiles, telles que les dépendances de bibliothèque ML manquantes ou la cause d'une panne de démarrage d'un connecteur.

Si le AWS IoT Greengrass groupe est configuré pour écrire des journaux locaux, le connecteur écrit des fichiers journaux dans*greengrass-root*`/ggc/var/log/user/`*region*`/aws/`. Pour plus d'informations sur la journalisation de Greengrass, consultez. the section called “Surveillance avec les journaux AWS IoT Greengrass”

Utilisez les informations suivantes pour résoudre les problèmes liés aux connecteurs ML Image Classification.

Bibliothèques système requises

Les onglets suivants répertorient les bibliothèques système requises pour chaque connecteur de classification d'images ML.

ML Image Classification Aarch64 JTX2

d'outils	Version minimale
ld-linux-aarch64. so.1	GLIBC_2.17
libc.so.6	GLIBC_2.17
libcublas.so.9.0	non applicable

d'outils	Version minimale
libcudart.so.9.0	non applicable
libcudnn.so.7	non applicable
libcufft.so.9.0	non applicable
libcurand.so.9.0	non applicable
libcusolver.so.9.0	non applicable
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

d'outils	Version minimale
ld-linux-x86-64. so.2	GCC_4.0.0
libc.so.6	GLIBC_2.4
libgfortran.so.3	GFORTRAN_1.0
libm.so.6	GLIBC_2.23
libpthread.so.0	GLIBC_2.2.5
librt.so.1	GLIBC_2.2.5

d'outils	Version minimale
libstdc++.so.6	CXXABI_1.3.8, GLIBCXX_3.4.21

ML Image Classification Armv7

d'outils	Version minimale
ld-linux-armhf.so.3	GLIBC_2.4
libc.so.6	GLIBC_2.7
libgcc_s.so.1	GCC_4.0.0
libgfortran.so.3	GFORTRAN_1.0
libm.so.6	GLIBC_2.4
libpthread.so.0	GLIBC_2.4
librt.so.1	GLIBC_2.4
libstdc++.so.6	CXXABI_1.3.8, CXXABI_ARM_1.3.3, GLIBCXX_3.4.20

Problèmes

Symptôme	Solution
Sur un appareil Raspberry Pi, le message d'erreur suivant est consigné, et vous n'utilisez pas la caméra : `Failed to initialize libdc1394`	Exécutez la commande suivante pour désactiver le pilote : `sudo ln /dev/null /dev/raw1394` Cette opération est éphémère et le lien symbolique disparaîtra après le redémarrage. Consultez le manuel de distribution de votre

Symptôme	Solution
	système d'exploitation pour savoir comment créer automatiquement le lien au redémarrage.

Licences

Les connecteurs ML Image Classification incluent les logiciels/licences tiers suivants :

- AWS SDK for Python (Boto3)/Licence Apache 2.0
- botocore/Licence Apache 2.0
- dateutil/Licence PSF
- docutils/Licence BSD, licence GPL (General Public License) GNU, licence Python Software Foundation, domaine public
- jmespath/Licence MIT
- s3transfer/Licence Apache 2.0
- urllib3/Licence MIT

- Bibliothèque DNNL (Deep Neural Network Library)/Licence Apache 2.0
- OpenMP* Runtime Library/Voir Intel OpenMP Runtime Library licensing.
- mxnet/Licence Apache 2.0
- six/MIT

Intel OpenMP Runtime Library licensing. Le moteur d'exécution Intel® OpenMP* est doté de deux licences, une licence commerciale (COM) faisant partie des produits de la suite Intel® Parallel Studio XE et une licence open source (OSS) BSD.

Ce connecteur est publié dans le cadre du contrat de licence logicielle Greengrass Core.

Journal des modifications

Le tableau suivant décrit les modifications apportées à chaque version du connecteur.

Version	Modifications
2	Ajout du `MLFeedbackConnectorConfigId` paramètre pour prendre en charge l'utilisation du connecteur ML Feedback pour télécharger les données d'entrée du modèle, publier des prédictions sur un sujet MQTT et publier des métriques sur Amazon CloudWatch.
1	Première version.

Un groupe Greengrass ne peut contenir qu'une seule version du connecteur à la fois. Pour de plus amples informations sur la mise à niveau d'une version de connecteur, veuillez consulter the section called “Mise à niveau des versions du connecteur”.

Consulter aussi

- Intégrer à des services et protocoles à l'aide de connecteurs
- the section called “Démarrer avec les connecteurs (console)”
- the section called “Démarrer avec les connecteurs (CLI)”
- Exécuter l'inférence de Machine Learning
- Algorithme de classification des images dans le guide SageMaker du développeur Amazon

Connecteur de détection d'objets ML

> **Warning**
>
> Ce connecteur a été déplacé dans lePhase de vie prolongée, etAWS IoT Greengrassne publiera pas de mises à jour fournissant des fonctionnalités, des améliorations aux fonctionnalités existantes, des correctifs de sécurité ou des corrections de bugs. Pour plus d'informations, consultez AWS IoT Greengrass Version 1politique de maintenance.

Détection d'objets MLconnecteursfournit un service d'inférence de Machine Learning (ML) qui s'exécute surAWS IoT GreengrassCore. Ce service d'inférence local effectue la détection d'objets à

l'aide d'un modèle de détection d'objets compilé par le SageMaker Compilateur Neo deep learning. Deux types de modèles de détection d'objets sont pris en charge : Détecteur Multibox Detector (SSD) et You Only Look Only Look Once (YOLO) v3. Pour plus d'informations, consultez Exigences du modèle de détection d'objets.

Les fonctions Lambda définies par l'utilisateur utilisent leAWS IoT GreengrassMachine Learning SDK pour soumettre des demandes d'inférence au service d'inférence local. Le service effectue l'inférence locale sur une image d'entrée et renvoie une liste de prédictions pour chaque objet détecté dans l'image. Chaque prédiction contient une catégorie d'objet, un score de fiabilité de prédiction et des coordonnées en pixels qui spécifient un cadre de délimitation autour de l'objet prédit.

AWS IoT Greengrassfournit des connecteurs ML Object Detection pour plusieurs plateformes :

Connecteur	Description et ARN
Détection d'objets ML Aarch64 JTX2	Service d'inférence de détection d'objets pour NVIDIA Jetson TX2. Prend en charge l'accélération GPU. ARN : `arn:aws:greengrass:` *`region`* `::/connectors/ObjectDetectionAarch64JTX2/versions/1`
Détection d'objets ML x86_64	Service d'inférence de détection d'objets pour les plateformes x86_64. ARN : `arn:aws:greengrass:` *`region`* `::/connectors/ObjectDetectionx86-64/versions/1`
Détection d'objets ML ARMv7	Service d'inférence de détection d'objets pour les plateformes ARMv7. ARN : `arn:aws:greengrass:` *`region`* `::/connectors/ObjectDetectionARMv7/versions/1`

Prérequis

Ces connecteurs possèdent les exigences suivantes :

- AWS IoT GreengrassCore Software 1.9.3 ou version ultérieure.
- Pythonversion 3.7 ou 3.8 installée sur l'appareil principal et ajoutée à la variable d'environnement PATH.

> **Note**
>
> Pour utiliser Python 3.8, exécutez la commande suivante pour créer un lien symbolique à partir du dossier d'installation Python 3.7 par défaut vers les binaires Python 3.8 installés.
>
> ```
> sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
> ```
>
> Ceci configure votre appareil de sorte qu'il réponde aux exigence de Python pour AWS IoT Greengrass.

- Dépendances pour le SageMaker Neo Deep Learning Runtime installé sur l'appareil principal (noyau). Pour plus d'informations, consultez the section called “Installation de dépendances d'exécution Neo Deep Learning”.
- Une ressource d'apprentissage machine du groupe Greengrass. La ressource d'apprentissage-machine doit faire référence à un compartiment Amazon S3 qui contient un modèle de détection d'objets. Pour de plus amples informations, veuillez consulterSources du modèle Amazon S3.

> **Note**
>
> Le modèle doit être un détecteur Multibox unique ou un type de modèle de détection d'objet You Look Once v3. Il doit être compilé à l'aide du SageMaker Compilateur Neo deep learning. Pour plus d'informations, consultez Exigences du modèle de détection d'objets.

- LeConnecteur ML Feedbackajouté au groupe Greengrass et configuré. Cette opération est obligatoire uniquement si vous souhaitez utiliser le connecteur pour charger les données d'entrée de modèle et publier les prédictions dans une rubrique MQTT.
- AWS IoT GreengrassKit SDK de Machine Learningv1.1.0 est nécessaire pour interagir avec ce connecteur.

Exigences du modèle de détection d'objets

Les connecteurs ML Object Detection prennent en charge les types de modèles de détection d'objet Single Shot Multibox Detector (SSD) et You Only Look Once (YOLO) v3. Vous pouvez utiliser les composants de détection d'objets fournis par GluonCV pour former le modèle avec votre propre ensemble de données. Vous pouvez également utiliser des modèles préformés depuis GluonCV Model Zoo :

- Modèle SSD préformé
- Modèle YOLO v3 préformé

Votre modèle de détection d'objets doit être formé avec 512 x 512 images d'entrée. Les modèles préformés du GluonCV Model Zoo répondent déjà à cette exigence.

Les modèles de détection d'objets formés doivent être compilés avec le kit SageMaker Compilateur Neo deep learning. Lors de la compilation, assurez-vous que le matériel cible correspond au matériel de votre appareil Greengrass principal. Pour de plus amples informations, veuillez consulter SageMaker Neodans leAmazon SageMaker Manuel du développeur.

Le modèle compilé doit être ajouté en tant que ressource d'apprentissage-machine (Source du modèle Amazon S3) au même groupe Greengrass que le connecteur.

Paramètres du connecteur

Ces connecteurs fournissent les paramètres suivants.

`MLModelDestinationPath`

Chemin d'accès absolu au compartiment Amazon S3 qui contient le modèle d'apprentissage-machine compatible Neo. Il s'agit du chemin de destination spécifié pour la ressource de modèle ML.

Nom d'affichage dans leAWS IoTConsole : Chemin de destination du modèle

Obligatoire`true`

Type: `string`

Modèle valide : `.+`

`MLModelResourceId`

ID de la ressource de ML qui référence le modèle de source.

Nom d'affichage dans leAWS IoTConsole : Ressources ML du groupe Greengrass

Obligatoire`true`

Type: `S3MachineLearningModelResource`

Modèle valide :`^[a-zA-Z0-9:_-]+$`

`LocalInferenceServiceName`

Nom du service d'inférence local. Les fonctions Lambda définies par l'utilisateur appellent le service en transmettant le nom au`invoke_inference_service`Fonction duAWS IoT GreengrassKit SDK de Machine Learning. Pour voir un exemple, consultez the section called “Exemple d'utilisation”.

Nom d'affichage dans leAWS IoTConsole : Nom du service d'inférence local

Obligatoire`true`

Type: `string`

Modèle valide :`^[a-zA-Z0-9][a-zA-Z0-9-]{1,62}$`

`LocalInferenceServiceTimeoutSeconds`

Durée (en secondes) avant laquelle la demande d'inférence est interrompue. La valeur minimale est de 1. La valeur par défaut est 10.

Nom d'affichage dans leAWS IoTConsole : Délai (secondes)

Obligatoire`true`

Type: `string`

Modèle valide :`^[1-9][0-9]*$`

`LocalInferenceServiceMemoryLimitKB`

Quantité de mémoire (en Ko) à laquelle le service a accès. La valeur minimale est de 1.

Nom d'affichage dans leAWS IoTConsole : Limite de mémoire

Obligatoire`true`

Type: `string`

Modèle valide :`^[1-9][0-9]*$`

`GPUAcceleration`

Contexte de calcul de l'UC ou du GPU (accéléré). Cette propriété s'applique au connecteur ML Image Classification Aarch64 JTX2 uniquement.

Nom d'affichage dans leAWS IoTConsole : Accélération GPU

Obligatoire`true`

Type: `string`

Valeurs valides : `CPU` ou `GPU`

`MLFeedbackConnectorConfigId`

ID de la configuration de commentaire à utiliser pour charger les données d'entrée du modèle. Cela doit correspondre à l'ID d'une configuration de commentaire définie pour le connecteur ML Feedback.

Cetteest obligatoire uniquement si vous souhaitez utiliser le connecteur ML Feedback pour charger les données d'entrée du modèle et publier des prédictions dans une rubrique MQTT.

Nom d'affichage dans leAWS IoTConsole : ID de configuration du connecteur ML Feedback

Obligatoire`false`

Type: `string`

Modèle valide :`^$|^[a-zA-Z0-9][a-zA-Z0-9-]{1,62}$`

Exemple de création de connecteur (AWS CLI)

La commande CLI suivante permet de créer un`ConnectorDefinition`avec une version initiale qui contient un connecteur ML Object Detection. Cet exemple crée une instance du connecteur ML Object Detection ARMv7l.

```
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 fonction Lambda de ces connecteurs dispose d'unlongue duréeCycle de vie.

DansAWS IoT Greengrass, vous pouvez ajouter un connecteur à partir duConnecteurs. Pour plus d'informations, consultez the section called “Démarrer avec les connecteurs (console)”.

Données d'entrée

Ces connecteurs acceptent un fichier image comme entrée. Les fichiers image d'entrée doivent être au format `jpeg` ou `png`. Pour plus d'informations, consultez the section called “Exemple d'utilisation”.

Ces connecteurs n'acceptent pas les messages MQTT comme données d'entrée.

Données de sortie

Ces connecteurs renvoient une liste formatée des résultats de prédiction pour les objets identifiés dans l'image d'entrée :

```
    {
        "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
            ],
            ...
        ]
    }
```

Chaque prédiction de la liste est contenue entre crochets et contient six valeurs :

- La première valeur représente la catégorie d'objets prédite pour l'objet identifié. Les catégories d'objets et leurs valeurs correspondantes sont déterminées lors de la formation de votre modèle d'apprentissage automatique de détection d'objets dans le compilateur Neo Deep Learning.
- La deuxième valeur est le score de confiance pour la prédiction de catégorie d'objet. Cela représente la probabilité que la prédiction soit correcte.
- Les quatre dernières valeurs correspondent aux dimensions en pixels qui représentent un cadre de délimitation autour de l'objet prédit dans l'image.

Ces connecteurs ne publient pas de messages MQTT en tant que données de sortie.

Exemple d'utilisation

L'exemple de fonction Lambda suivant utilise leAWS IoT GreengrassKit SDK de Machine Learningpour interagir avec un connecteur de détection d'objets ML.

> **Note**
>
> Vous pouvez télécharger le kit SDK à partir duAWS IoT GreengrassKit SDK de Machine LearningPage des téléchargements.

L'exemple initialise un client de kit de développement logiciel et appelle de façon synchrone la fonction `invoke_inference_service` du SDK pour appeler le service d'inférence local. Il transmet le type d'algorithme, le nom de service, le type d'image et le contenu de l'image. Ensuite, l'exemple analyse la réponse du service pour obtenir les résultats probables (prédictions).

```
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
```

Le`invoke_inference_service`dans la fonctionAWS IoT GreengrassLe kit de développement logiciel Machine Learning accepte les arguments suivants.

Argument	Description
`AlgoType`	Nom du type d'algorithme à utiliser pour l'inférence. Actuellement, seul `object-detection` est pris en charge. Obligatoire`true` Type: `string` Valeurs valides : `object-detection`
`ServiceName`	Nom du service d'inférence local. Utilisez le nom que vous avez spécifié pour le paramètre `LocalInferenceServiceName` lorsque vous avez configuré le connecteur. Obligatoire`true` Type: `string`
`ContentType`	Nom du type mime de l'image d'entrée.

Argument	Description
	Obligatoire`true` Type: `string` Valeurs valides : `image/jpeg, image/png`
`Body`	Contenu du type mime de l'image d'entrée. Obligatoire`true` Type: `binary`

Installation de dépendances d'exécution Neo Deep Learning sur le noyau AWS IoT Greengrass

Les connecteurs ML Object Detection sont regroupés avec le kit SageMaker Exécution Neo deep learning (DLR). Les connecteurs utilisent le runtime pour servir le modèle d'apprentissage-machine. Pour utiliser ces connecteurs, vous devez installer les dépendances pour le DLR sur votre appareil principal.

Avant d'installer les dépendances MXNet, assurez-vous que les bibliothèques système requises (avec les versions minimales spécifiées) sont présentes sur l'appareil.

NVIDIA Jetson TX2

1. Installer CUDA Toolkit 9.0 et cuDNN 7.0. Vous pouvez suivre les instructions de the section called “Configuration d'autres appareils” dans le didacticiel Démarrez.
2. Activez les référentiels univers de sorte que le connecteur peut installer les logiciels open source maintenus par la communauté. Pour plus d'informations, consultez Repositories/Ubuntu dans la documentation Ubuntu.

   a. Ouvrez le fichier `/etc/apt/sources.list`.

   b. Vérifiez que les lignes suivantes ne sont pas commentées :

   ```
   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. Enregistrez une copie du script d'installation suivant dans un fichier nommé `nvidiajtx2.sh` sur l'appareil principal.

```
#!/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**
>
> Si OpenCV ne s'installe pas correctement à l'aide de ce script, vous pouvez essayer de générer à partir de la source. Pour plus d'informations, consultez Installation sous Linux dans la documentation OpenCV ou consultez d'autres ressources en ligne pour votre plateforme.

4. Dans le répertoire où vous avez enregistré le fichier, exécutez la commande suivante :

```
sudo nvidiajtx2.sh
```

x86_64 (Ubuntu or Amazon Linux)

1. Enregistrez une copie du script d'installation suivant dans un fichier nommé `x86_64.sh` sur l'appareil principal.

```
#!/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**
>
> Si OpenCV ne s'installe pas correctement à l'aide de ce script, vous pouvez essayer de générer à partir de la source. Pour plus d'informations, consultez Installation sous

Linux dans la documentation OpenCV ou consultez d'autres ressources en ligne pour votre plateforme.

2. Dans le répertoire où vous avez enregistré le fichier, exécutez la commande suivante :

```
sudo x86_64.sh
```

ARMv7 (Raspberry Pi)

1. Enregistrez une copie du script d'installation suivant dans un fichier nommé `armv7l.sh` sur l'appareil principal.

```
#!/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**
>
> Si OpenCV ne s'installe pas correctement à l'aide de ce script, vous pouvez essayer de générer à partir de la source. Pour plus d'informations, consultez Installation sous Linux dans la documentation OpenCV ou consultez d'autres ressources en ligne pour votre plateforme.

2. Dans le répertoire où vous avez enregistré le fichier, exécutez la commande suivante :

```
sudo bash armv7l.sh
```

> **Note**
>
> Sur un Raspberry Pi, l'utilisation de `pip` pour installer des dépendances d'apprentissage automatique est une opération gourmande en mémoire qui peut entraîner un dépassement de la capacité mémoire de l'appareil et une absence de réponse. Pour contourner ce problème, vous pouvez augmenter temporairement l'espace d'échange. Dans `/etc/dphys-swapfile`, augmentez la valeur de la variable `CONF_SWAPSIZE`, puis exécutez la commande suivante pour redémarrer `dphys-swapfile`.
>
> ```
> /etc/init.d/dphys-swapfile restart
> ```

Journalisation et dépannage

En fonction des paramètres de votre groupe, les journaux d'événements et d'erreurs sont écrits dans CloudWatch Journaux, le système de fichiers local ou les deux. Les journaux de ce connecteur utilisent le préfixe `LocalInferenceServiceName`. Si le connecteur se comporte de manière inattendue, vérifiez les journaux du connecteur. Ces derniers contiennent généralement des informations de débogage utiles, telles que les dépendances de bibliothèque ML manquantes ou la cause d'une panne de démarrage d'un connecteur.

Si l'icôneAWS IoT Greengrassgroupe est configuré pour écrire des journaux locaux, le connecteur écrit les fichiers journaux dans*greengrass-root*`/ggc/var/log/user/`*region*`/aws/`. Pour plus d'informations sur la journalisation Greengrass, consultezthe section called “Surveillance avec les journaux AWS IoT Greengrass”.

Utilisez les informations suivantes pour résoudre les problèmes liés aux connecteurs ML Object Detection.

Bibliothèques système requises

Les onglets suivants répertorient les bibliothèques système requises pour chaque connecteur ML Object Detection.

ML Object Detection Aarch64 JTX2

d'outils	Version minimale
ld-linux-aarch64.so.1	GLIBC_2.17
libc.so.6	GLIBC_2.17
libcublas.so.9.0	ne s'applique pas
libcudart.so.9.0	ne s'applique pas
libcudnn.so.7	ne s'applique pas
libcufft.so.9.0	ne s'applique pas
libcurand.so.9.0	ne s'applique pas
libcusolver.so.9.0	ne s'applique pas
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	ne s'applique pas
libnvrm_gpu.so	ne s'applique pas
libnvrm.so	ne s'applique pas
libnvidia-fatbinaryloader.so.28.2.1	ne s'applique pas
libnvos.so	ne s'applique pas
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

d'outils	Version minimale
ld-linux-x86-64.so.2	GCC_4.0.0
libc.so.6	GLIBC_2.4
libgfortran.so.3	GFORTRAN_1.0
libm.so.6	GLIBC_2.23
libpthread.so.0	GLIBC_2.2.5
librt.so.1	GLIBC_2.2.5
libstdc++.so.6	CXXABI_1.3.8, GLIBCXX_3.4.21

ML Object Detection ARMv7

d'outils	Version minimale
ld-linux-armhf.so.3	GLIBC_2.4
libc.so.6	GLIBC_2.7
libgcc_s.so.1	GCC_4.0.0
libgfortran.so.3	GFORTRAN_1.0
libm.so.6	GLIBC_2.4
libpthread.so.0	GLIBC_2.4
librt.so.1	GLIBC_2.4
libstdc++.so.6	CXXABI_1.3.8, CXXABI_ARM_1.3.3, GLIBCXX_3.4.20

Problèmes

Symptôme	Solution
Sur un appareil Raspberry Pi, le message d'erreur suivant est consigné, et vous n'utilisez pas la caméra : `Failed to initialize libdc1394`	Exécutez la commande suivante pour désactiver le pilote : `sudo ln /dev/null /dev/raw1394` Cette opération est éphémère. Le lien symbolique disparaît après le redémarrage. Consultez le manuel de distribution de votre système d'exploitation pour savoir comment créer automatiquement le lien au redémarrage.

Licences

Les connecteurs ML Object Detection incluent les logiciels et licences tiers suivants :

- AWS SDK for Python (Boto3)/Licence Apache 2.0
- botocore/Licence Apache 2.0
- dateutil/Licence PSF
- docutils/Licence BSD, licence GPL (General Public License) GNU, licence Python Software Foundation, domaine public
- jmespath/Licence MIT
- s3transfer/Licence Apache 2.0
- urllib3/Licence MIT

- Environnement d’exécution Deep Learning/Licence Apache 2.0
- six/MIT

Ce connecteur est libéré sous leContrat de licence du logiciel Greengrass Core.

Consulter aussi

- Intégrer à des services et protocoles à l'aide de connecteurs
- the section called “Démarrer avec les connecteurs (console)”
- the section called “Démarrer avec les connecteurs (CLI)”
- Exécuter l'inférence de Machine Learning
- Algorithme de détection d'objetsdans leAmazon SageMaker Manuel du développeur

Connecteur

Modbus-RTU Protocol Adapconnecteurrecherche des informations depuis les périphériques ModbusAWS IoT Greengrass.

Ce connecteur reçoit les paramètres d'une demande Modbus RTU à partir d'une fonction Lambda définie par l'utilisateur. Il envoie la demande correspondante, puis publie la réponse à partir du périphérique cible comme un message MQTT.

Ce connecteur a les versions suivantes.

Version	ARN
3	`arn:aws:greengrass:  region::/connectors/ModbusRTUProtocolAdapter/versions/3`
2	`arn:aws:greengrass:  region::/connectors/ModbusRTUProtocolAdapter/versions/2`
1	`arn:aws:greengrass:  region::/connectors/ModbusRTUProtocolAdapter/versions/1`

Pour obtenir des informations sur les changements apportés aux versions, veuillez consulter le Journal des modifications.

Prérequis

Ce connecteur possède les critères suivants :

Version 3

- Logiciel AWS IoT Greengrass Core 1.9.3 ou version ultérieure.
- Pythonversion 3.7 ou 3.8 installée sur l'appareil principal et ajoutée à la variable d'environnement PATH.

  > **Note**
  >
  > Pour utiliser Python 3.8, exécutez la commande suivante pour créer un lien symbolique entre le dossier d'installation de Python 3.7 par défaut et les binaires Python 3.8 installés.
  >
  > ```
  > sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
  > ```
  >
  > Ceci configure votre appareil de sorte qu'il réponde aux exigence de Python pour AWS IoT Greengrass.

- Une connexion physique entreAWS IoT Greengrasset les appareils Modbus. Le noyau doit être physiquement connecté au réseau Modbus RTU via un port série (par exemple, un port USB).
- UNressource de périphérique localedans le groupe Greengrass qui pointe vers le port série physique Modbus.
- Une fonction Lambda définie par l'utilisateur qui envoie des paramètres de demande Modbus RTU à ce connecteur. Les paramètres de demande doivent respecter les modèles attendus et inclure les identifiants et les adresses des périphériques cibles sur le réseau Modbus RTU. Pour plus d'informations, consultez the section called “Données d'entrée”.

Versions 1 - 2

- AWS IoT GreengrassLogiciel Core v1.7 ou version ultérieure.
- Pythonversion 2.7 installée sur l'appareil principal et ajoutée à la variable d'environnement PATH.
- Une connexion physique entreAWS IoT Greengrasset les appareils Modbus. Le noyau doit être physiquement connecté au réseau Modbus RTU via un port série (par exemple, un port USB).

- UNressource de périphérique localedans le groupe Greengrass qui pointe vers le port série physique Modbus.
- Une fonction Lambda définie par l'utilisateur qui envoie des paramètres de demande Modbus RTU à ce connecteur. Les paramètres de demande doivent respecter les modèles attendus et inclure les identifiants et les adresses des périphériques cibles sur le réseau Modbus RTU. Pour plus d'informations, consultez the section called “Données d'entrée”.

Paramètres du connecteur

Ce connecteur prend en charge les paramètres suivants :

`ModbusSerialPort-ResourceId`

L'ID de la ressource de l'appareil local qui représente le port série physique Modbus.

> **Note**
>
> Ce connecteur bénéficie d'un accès en lecture/écriture à la ressource.

Nom d'affichage dans laAWS IoTConsole : Ressource du port série Modbus

Obligatoire :`true`

Type: `string`

Modèle valide :`.+`

`ModbusSerialPort`

Chemin d'accès absolu au port série Modbus physique sur le périphérique. Il s'agit du chemin d'accès source qui est spécifié pour la ressource du périphérique local Modbus.

Nom d'affichage dans laAWS IoTConsole : Chemin d'accès source vers la ressource du port série Modbus

Obligatoire :`true`

Type: `string`

Modèle valide :`.+`

Exemple de création de connecteur (AWS CLI)

La commande CLI suivante permet de créer un`ConnectorDefinition`avec une version initiale qui contient le connecteur

```
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 fonction Lambda de ce connecteur possède unlongue duréeCycle de vie.

DansAWS IoT Greengrass, vous pouvez ajouter un connecteur à partir de la consoleConnecteurs. Pour plus d'informations, consultez the section called “Démarrer avec les connecteurs (console)”.

> **Note**
>
> Une fois que vous avez déployé le connecteurAWS IoT Things Graphpour orchestrer les interactions entre les appareils de votre groupe. Pour plus d'informations, consultez Modbus dans le Guide de l'utilisateur AWS IoT Things Graph.

Données d'entrée

Ce connecteur accepte des paramètres de demande Modbus RTU à partir d'une fonction Lambda définie par l'utilisateur dans une rubrique MQTT. Les messages d'entrée doivent être au format JSON.

Filtre de rubrique dans l'abonnement

`modbus/adapter/request`

Propriétés des messages

Le message de demande varie en fonction du type de demande Modbus RTU qu'il représente. Les propriétés suivantes sont requises pour toutes les demandes :

- Dans l'objet `request` :
  - `operation`. Nom de l'opération à exécuter. Par exemple, spécifiez `"operation": "ReadCoilsRequest"` pour lire les bobines. Cette valeur doit être une chaîne Unicode. Pour les opérations prises en charge, veuillez consulter the section called “Demandes et réponses de Modbus RTU”.
  - `device`. L'appareil cible de la requête. Cette valeur doit être comprise entre `0 - 247`.
- La propriété `id`. Un ID pour la demande. Cette valeur est utilisée pour la déduplication des données et est renvoyée en l'état dans la propriété `id` de toutes les réponses, y compris les réponses d'erreur. Cette valeur doit être une chaîne Unicode.

> **Note**
>
> Si votre demande inclut un champ d'adresse, vous devez spécifier la valeur sous la forme d'un entier. Par exemple, `"address": 1`.

Les autres paramètres à inclure dans la demande dépendent de l'opération. Tous les paramètres de demande sont obligatoires, à l'exception du CRC, qui est traité séparément. Pour obtenir des exemples, consultez the section called “Exemples de demandes et de réponses”.

Exemple d'entrée : Demande des spires

```
{
    "request": {
        "operation": "ReadCoilsRequest",
     "device": 1,
     "address": 1,
     "count": 1
    },
    "id": "TestRequest"
}
```

Données de sortie

Ce connecteur publie des réponses aux demandes Modbus RTU entrantes.

Filtre de rubrique dans l'abonnement

`modbus/adapter/response`

Propriétés des messages

Le format du message de réponse varie en fonction de la demande correspondante et du statut de le réponse. Pour obtenir des exemples, consultez the section called “Exemples de demandes et de réponses”.

> **Note**
>
> Une réponse pour une opération d'écriture est simplement un écho de la demande. Même si aucune information constructive n'est renvoyée pour des réponses en écriture, il est recommandé de vérifier l'état de la réponse.

Chaque réponse inclut les propriétés suivantes :

- Dans l'objet `response` :
  - `status`. Le statut d'une demande. Le statut peut avoir l'une des valeurs suivantes :
    - `Success`. La demande a été valide, envoyée au réseau Modbus RTU et une réponse a été renvoyée.
    - `Exception`. La demande a été valide, envoyée au réseau Modbus RTU et une réponse d'exception a été renvoyée. Pour plus d'informations, consultez the section called “Statut de la réponse : Exception”.
    - `No Response`. La demande n'était pas valide et le connecteur a capturé l'erreur avant que la demande n'ait été envoyée via le réseau Modbus RTU. Pour plus d'informations, consultez the section called “Statut de la réponse : Pas de réponse”.
  - `device`. L'appareil auquel la demande a été envoyée.
  - `operation`. Le type de demande qui a été envoyé.
  - `payload`. Le contenu de la réponse qui a été renvoyé. Si le `status` est `No Response`, cet objet contient uniquement une propriété `error` avec la description de l'erreur (par exemple, `"error": "[Input/Output] No Response received from the remote unit"`).
- La propriété `id`. L'ID de la demande, utilisé pour la déduplication des données.

Exemple de sortie : Succès

```
{
    "response" : {
        "status" : "success",
        "device": 1,
     "operation": "ReadCoilsRequest",
     "payload": {
         "function_code": 1,
         "bits": [1]
     }
     },
     "id" : "TestRequest"
}
```

Exemple de sortie : Échec

```
{
    "response" : {
        "status" : "fail",
        "error_message": "Internal Error",
        "error": "Exception",
        "device": 1,
     "operation": "ReadCoilsRequest",
     "payload": {
         "function_code": 129,
         "exception_code": 2
     }
     },
     "id" : "TestRequest"
}
```

Pour obtenir plus d'exemples, consultez the section called “Exemples de demandes et de réponses”.

Demandes et réponses de Modbus RTU

Ce connecteur accepte les paramètres de demande de Modbus RTU en tant que données d'entrée et publie les réponses en tant que données de sortie.

Les opérations communes suivantes sont prises en charge.

Nom de l'opération dans la demande	Code de fonction dans la réponse
ReadCoilsRequest	01
ReadDiscreteInputsRequest	02
ReadHoldingRegistersRequest	03
ReadInputRegistersRequest	04
WriteSingleCoilRequest	05
WriteSingleRegisterRequest	06
WriteMultipleCoilsRequest	15
WriteMultipleRegistersRequest	16
MaskWriteRegisterRequest	22
ReadWriteMultipleRegistersRequest	23

Exemples de demandes et de réponses

Voici des exemples de demandes et de réponses pour les opérations prises en charge.

Read Coils (Lire des spires)

Exemple de requête :

```
{
    "request": {
        "operation": "ReadCoilsRequest",
     "device": 1,
     "address": 1,
     "count": 1
    },
    "id": "TestRequest"
}
```

Exemple de réponse :

```
{
    "response": {
        "status": "success",
        "device": 1,
     "operation": "ReadCoilsRequest",
     "payload": {
         "function_code": 1,
         "bits": [1]
     }
     },
     "id" : "TestRequest"
}
```

Read Discrete Inputs (Lire des entrées discrètes)

Exemple de requête :

```
{
    "request": {
        "operation": "ReadDiscreteInputsRequest",
        "device": 1,
        "address": 1,
        "count": 1
    },
    "id": "TestRequest"
}
```

Exemple de réponse :

```
{
    "response": {
        "status": "success",
        "device": 1,
        "operation": "ReadDiscreteInputsRequest",
        "payload": {
            "function_code": 2,
            "bits": [1]
        }
     },
     "id" : "TestRequest"
}
```

Read Holding Registers (Lire les registres en attente)

Exemple de requête :

```
{
    "request": {
        "operation": "ReadHoldingRegistersRequest",
     "device": 1,
     "address": 1,
     "count": 1
    },
    "id": "TestRequest"
}
```

Exemple de réponse :

```
{
    "response": {
        "status": "success",
        "device": 1,
     "operation": "ReadHoldingRegistersRequest",
     "payload": {
         "function_code": 3,
            "registers": [20,30]
     }
     },
     "id" : "TestRequest"
}
```

Read Input Registers (Lire les registres d'entrée)

Exemple de requête :

```
{
    "request": {
        "operation": "ReadInputRegistersRequest",
     "device": 1,
     "address": 1,
     "value": 1
    },
    "id": "TestRequest"
}
```

Write Single Coil (Écrire une spire unique)

Exemple de requête :

```
{
    "request": {
        "operation": "WriteSingleCoilRequest",
     "device": 1,
     "address": 1,
     "value": 1
    },
    "id": "TestRequest"
}
```

Exemple de réponse :

```
{
    "response": {
        "status": "success",
        "device": 1,
     "operation": "WriteSingleCoilRequest",
     "payload": {
         "function_code": 5,
         "address": 1,
         "value": true
     }
     },
     "id" : "TestRequest"
```

Write Single Register (Écrire un registre unique)

Exemple de requête :

```
{
    "request": {
        "operation": "WriteSingleRegisterRequest",
     "device": 1,
     "address": 1,
     "value": 1
    },
    "id": "TestRequest"
}
```

Write Multiple Coils (Écrire plusieurs spires)

Exemple de requête :

```
{
    "request": {
        "operation": "WriteMultipleCoilsRequest",
     "device": 1,
     "address": 1,
     "values": [1,0,0,1]
    },
    "id": "TestRequest"
}
```

Exemple de réponse :

```
{
    "response": {
        "status": "success",
        "device": 1,
     "operation": "WriteMultipleCoilsRequest",
     "payload": {
         "function_code": 15,
         "address": 1,
         "count": 4
     }
     },
     "id" : "TestRequest"
}
```

Write Multiple Registers (Écrire plusieurs registres)

Exemple de requête :

```
{
    "request": {
        "operation": "WriteMultipleRegistersRequest",
     "device": 1,
     "address": 1,
     "values": [20,30,10]
    },
    "id": "TestRequest"
```

```
}
```

Exemple de réponse :

```
{
    "response": {
        "status": "success",
        "device": 1,
     "operation": "WriteMultipleRegistersRequest",
     "payload": {
         "function_code": 23,
         "address": 1,
         "count": 3
     }
     },
     "id" : "TestRequest"
}
```

Mask Write Register (Masquer le registre d'écriture)

Exemple de requête :

```
{
    "request": {
        "operation": "MaskWriteRegisterRequest",
     "device": 1,
     "address": 1,
        "and_mask": 175,
        "or_mask": 1
    },
    "id": "TestRequest"
}
```

Exemple de réponse :

```
{
    "response": {
        "status": "success",
        "device": 1,
     "operation": "MaskWriteRegisterRequest",
     "payload": {
         "function_code": 22,
```

```
            "and_mask": 0,
            "or_mask": 8
     }
     },
     "id" : "TestRequest"
}
```

Read Write Multiple Registers (Écrire/lire plusieurs registres)

Exemple de requête :

```
{
    "request": {
        "operation": "ReadWriteMultipleRegistersRequest",
     "device": 1,
     "read_address": 1,
        "read_count": 2,
        "write_address": 3,
        "write_registers": [20,30,40]
    },
    "id": "TestRequest"
}
```

Exemple de réponse :

```
{
    "response": {
        "status": "success",
        "device": 1,
     "operation": "ReadWriteMultipleRegistersRequest",
     "payload": {
         "function_code": 23,
         "registers": [10,20,10,20]
     }
     },
     "id" : "TestRequest"
}
```

> **Note**
>
> Les registres renvoyés dans cette réponse sont les registres qui sont lus.

Statut de la réponse : Exception

Des exceptions peuvent se produire lorsque le format de la demande est valide, mais la demande échoue. Dans ce cas, la réponse contient les informations suivantes :

- Le `status` est réglé sur `Exception`.
- Le `function_code` est égal au code de fonction de la requête + 128.
- Le `exception_code` contient le code d'exception. Pour plus d'informations, consultez les codes d'exception de Modbus.

Exemple :

```
{
    "response" : {
        "status" : "fail",
        "error_message": "Internal Error",
        "error": "Exception",
        "device": 1,
     "operation": "ReadCoilsRequest",
     "payload": {
         "function_code": 129,
         "exception_code": 2
     }
     },
     "id" : "TestRequest"
}
```

Statut de la réponse : Pas de réponse

Ce connecteur effectue des vérifications de validation sur la demande Modbus. Par exemple, il vérifie les formats non valides et les champs manquants. Si la validation échoue, le connecteur n'envoie pas la demande. A la place, il renvoie une réponse qui contient les informations suivantes :

- Le `status` est réglé sur `No Response`.
- Le`error`contient la cause de l'erreur.
- Le `error_message` contient le message de l'erreur.

Exemples :

```
{
    "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"
}
```

Si la demande vise un appareil qui n'existe pas ou si le réseau Modbus RTU ne fonctionne pas, vous pouvez obtenir une `ModbusIOException` qui utilise le format No Response.

```
{
    "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"
}
```

Exemple d'utilisation

Suivez les étapes détaillées suivantes pour configurer un exemple de fonction Lambda Python 3.7 que vous pouvez utiliser pour tester le connecteur.

> **Note**
>
> - Si vous utilisez d'autres environnements d'exécution Python, vous pouvez créer un lien symbolique de Python3.x vers Python 3.7.

- Les rubriques Démarrer avec les connecteurs (console) et Démarrer avec les connecteurs (CLI) contiennent des étapes détaillées qui vous montrent comment configurer et déployer un exemple de connecteur Twilio Notifications.

1. Veillez à répondre aux conditions requises pour le connecteur.
2. Créez et publiez une fonction Lambda qui envoie des données d'entrée au connecteur.

   Enregistrez l'exemple de code en tant que fichier PY. Téléchargez et décompressez leAWS IoT GreengrassKit SDK Core pour Python. Ensuite, créez un package zip contenant le fichier PY et le dossier `greengrasssdk` au niveau racine. Ce package zip correspond au package de déploiement que vous chargez surAWS Lambda.

   Une fois que vous avez créé la fonction Lambda Python 3.7, publiez une version de fonction et créez un alias.
3. Configurez votre groupe Greengrass.

   a. Ajoutez la fonction Lambda par son alias (recommandé). Configurez le cycle de vie Lambda comme long (ou`"Pinned": true`dans la CLI).

   b. Ajoutez la ressource de périphérique local requise et accordez un accès en lecture/écriture à la fonction Lambda.

   c. Ajoutez le connecteur et configurez ses paramètres.

   d. Ajoutez des abonnements qui permettent au connecteur de recevoir des données d'entrée et d'envoyer des données de sortie sur des filtres de rubrique pris en charge.

      - Définissez la fonction Lambda en tant que source, le connecteur en tant que cible et utilisez un filtre de rubrique d'entrée pris en charge.
      - Définissez le connecteur en tant que source, AWS IoT Core en tant que cible et utilisez un filtre de rubrique de sortie pris en charge. Vous utilisez cet abonnement pour afficher les messages d'état dans laAWS IoTconsole
4. Déployez le groupe.
5. DansAWS IoTsur la consoleTestabonnez-vous à la rubrique des données de sortie pour afficher les messages d'état du connecteur. L'exemple Lambda à longue durée de vie commence à envoyer des messages immédiatement après le déploiement du groupe.

Lorsque vous avez terminé le test, vous pouvez définir le test, vous pouvez définir le type « à la demande » (ou`"Pinned": false`dans l'interface de ligne de commande) et déployez le groupe. Cela empêche la fonction d'envoyer des messages.

Exemple

L'exemple de fonction Lambda suivant envoie un message d'entrée au connecteur.

```
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
```

Licences

Le connecteur de Modbus-RTU Protocol Adap

- pymodbus/BSD
- pyserial/BSD

Ce connecteur est libéré sous leContrat de licence du logiciel Greengrass Core.

Journal des modifications

Le tableau suivant décrit les modifications apportées à chaque version du connecteur.

Version	Modifications
3	Mise à niveau de l'environnement Lambda vers Python 3.7, ce qui modifie l'exigence d'environ nement exécution.
2	Mise à jour du nom ARNRégion AWSPrise en charge. Amélioration de la journalisation des erreurs.
1	Première version.

Un groupe Greengrass peut contenir une seule version du connecteur à la fois. Pour de plus amples informations sur la mise à niveau d'une version de connecteur, veuillez consulter the section called “Mise à niveau des versions du connecteur”.

Consulter aussi

- Intégrer à des services et protocoles à l'aide de connecteurs
- the section called “Démarrer avec les connecteurs (console)”
- the section called “Démarrer avec les connecteurs (CLI)”

Connecteur adaptateur de protocole Modbus-TCP

L'adaptateur de protocole Modbus-TCPconnecteurcollecte des données à partir d'appareils locaux via le protocole ModBustCP et les publie sur le site sélectionné`StreamManager`flux.

Vous pouvez également utiliser ce connecteur avec l'IoT SiteWise connecteur et votre IoT SiteWise passerelle. Votre passerelle doit fournir la configuration du connecteur. Pour de plus amples informations, veuillez consulterConfigurer une source Modbus TCPdans l'IoT SiteWise Guide de l'utilisateur.

> **Note**
>
> Ce connecteur fonctionne dansAucun conteneurmode d'isolation, pour que vous puissiez le déployer sur unAWS IoT Greengrassgroupe exécuté dans un conteneur Docker.

Ce connecteur possède les versions suivantes.

Version	ARN
3	`arn:aws:greengrass:  region::/connectors/ModbusTCPConnector/versions/3`
2	`arn:aws:greengrass:  region::/connectors/ModbusTCPConnector/versions/2`
1	`arn:aws:greengrass:  region::/connectors/ModbusTCPConnector/versions/1`

Pour obtenir des informations sur les changements apportés aux versions, veuillez consulter le Journal des modifications.

Prérequis

Ce connecteur possède les critères suivants :

Version 1 - 3

- AWS IoT GreengrassCore software v1.10.2 ou version ultérieure.
- Gestionnaire de flux activé sur l'AWS IoT Greengrass.

- Java 8 installé sur l'appareil principal et ajouté au`PATH`variable d'environnement.

> **Note**
>
> Ce connecteur est disponible uniquement dans les régions suivantes :
>
> - ap-southeast-1
> - ap-southeast-2
> - eu-central-1
> - eu-west-1
> - us-east-1
> - us-west-2
> - cn-north-1

Paramètres du connecteur

Ce connecteur prend en charge les paramètres suivants :

`LocalStoragePath`

Le répertoire duAWS IoT Greengrasshôte que l'IoT SiteWise peut écrire des données persistantes dans. Le répertoire par défaut est `/var/sitewise`.

Nom d'affichage dans l'AWS IoTConsole : Chemin de stockage local

Obligatoire`false`

Type: `string`

Modèle valide :`^\s*$|\/.`

`MaximumBufferSize`

Taille maximale en Go pour l'IoT SiteWise utilisation du disque. La taille par défaut est de 10 Go.

Nom d'affichage dans l'AWS IoTConsole : Taille maximale du tampon de disque

Obligatoire`false`

Type: `string`

Modèle valide :`^\s*$|[0-9]+`

`CapabilityConfiguration`

Ensemble de configurations de collecteurs Modbus TCP à partir desquelle le connecteur collecte des données et se connecte.

Nom d'affichage dans l'AWS IoTConsole : CapabilityConfiguration

Obligatoire`false`

Type : Chaîne JSON bien formée qui définit l'ensemble des configurations de commentaire prises en charge.

Voici un exemple de`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
                    }
                }
            ]
        }
    ]
}
```

Exemple de création de connecteur (AWS CLI)

La commande CLI suivante crée un`ConnectorDefinition`avec une version initiale qui contient le connecteur Modbus-TCP Protocol Adapter.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '
{
    "Connectors": [
        {
            "Id": "MyModbusTCPConnector",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/ModbusTCP/
versions/3",
            "Parameters": {
                "capability_configuration": "{\"version\":1,\"namespace\":
\"iotsitewise:modbuscollector:1\",\"configuration\":\"{\"sources\":[{\"type
\":\"ModBusTCPSource\",\"name\":\"SourceName1\",\"measurementDataStreamPrefix
\":\"\",\"endpoint\":{\"ipAddress\":\"127.0.0.1\",\"port\":8081,\"unitId\":1},
\"propertyGroups\":[{\"name\":\"PropertyGroupName\",\"tagPathDefinitions\":[{\"type
\":\"ModBusTCPAddress\",\"tag\":\"TT-001\",\"address\":\"30001\",\"size\":2,
\"srcDataType\":\"hexdump\",\"transformation\":\"noSwap\",\"dstDataType\":\"string
\"}],\"scanMode\":{\"rate\":200,\"type\":\"POLL\"}}],\"destination\":{\"type\":
\"StreamManager\",\"streamName\":\"SiteWise_Stream\",\"streamBufferSize\":10},
\"minimumInterRequestDuration\":200}]}\"}"
            }
        }
    ]
```

```
}'
```

> **Note**
>
> La fonction Lambda de ce connecteur possède unlongue duréeCycle de vie.

Données d'entrée

Ce connecteur n'accepte pas les messages MQTT comme données d'entrée.

Données de sortie

Ce connecteur publie des données dans`StreamManager`. Vous devez configurer le flux de messages de destination. Les messages de sortie ont la structure suivante :

```
{
    "alias": "string",
    "messages": [
        {
            "name": "string",
            "value": boolean|double|integer|string,
            "timestamp": number,
            "quality": "string"
        }
    ]
}
```

Licences

Le connecteur Modbus-TCP Protocol Adapter inclut les logiciels et licences tiers suivants :

- Petri numériqueModbus

Ce connecteur est libéré sous leContrat de licence du logiciel Greengrass Core.

Journal des modifications

Le tableau suivante décrit les modifications apportées à chaque version du connecteur.

Version	Modifications	Date
3 (recommandé)	Cette version contient des correctifs de bogues.	22 décembre 2021
2	Ajout de la prise en charge des chaînes source codées ASCII, UTF8 et ISO8859.	24 mai 2021
1	Première version.	15 décembre 2020

Un groupe Greengrass peut contenir une seule version du connecteur à la fois. Pour de plus amples informations sur la mise à niveau d'une version de connecteur, veuillez consulter the section called “Mise à niveau des versions du connecteur”.

Consulter aussi

- Intégrer à des services et protocoles à l'aide de connecteurs
- the section called “Démarrer avec les connecteurs (console)”
- the section called “Démarrer avec les connecteurs (CLI)”

Raspberry Pi

> **Warning**
>
> Ce connecteur a été déplacé dans lePhase de vie prolongée, etAWS IoT Greengrassne publiera pas de mises à jour fournissant des fonctionnalités, des améliorations à des fonctionnalités existantes, des correctifs de sécurité ou des corrections de bogues. Pour plus d'informations, consultez AWS IoT Greengrass Version 1politique de maintenance.

Le Raspberry PiconnecteurContrôle les broches d'entrée/sortie à usage général (GPIO) sur un appareil principal Raspberry Pi.

Ce connecteur interroge les broches d'entrée à un intervalle spécifié et publie les changements d'état dans des rubriques MQTT. Il accepte également les demandes de lecture et d'écriture adressées

comme des messages MQTT à partir de fonctions Lambda définies par l'utilisateur Les demandes d'écriture sont utilisées pour régler la broche sur haute ou basse tension.

Le connecteur fournit des paramètres que vous utilisez pour désigner des broches d'entrée et de sortie. Ce comportement est configuré avant le déploiement de groupe. Il ne peut pas être modifié au moment de l'exécution.

- Les broches d'entrée peuvent être utilisées pour recevoir des données à partir des appareils périphériques.
- Les broches de sortie peuvent être utilisées pour contrôler les périphériques ou envoyer des données aux périphériques.

Vous pouvez utiliser ce connecteur pour de nombreux scénarios, tels que :

- contrôler des feux vert, orange et rouge d'un feu de circulation.
- contrôler un ventilateur (relié à un relais électrique) basé sur les données provenant d'un capteur d'humidité.
- alerter des employés dans un magasin lorsque les clients appuient sur un bouton.
- utiliser un commutateur de lumière intelligent pour contrôler d'autres appareils IoT.

> **Note**
>
> Ce connecteur n'est pas adapté aux applications qui ont des besoins en temps réel. Des événements de courte durée peuvent être manqués.

Ce connecteur a les versions suivantes.

Version	ARN
3	`arn:aws:greengrass:  region::/connectors/RaspberryPiGPIO/versions/3`

Version	ARN
2	`arn:aws:greengrass:  region::/connectors/RaspberryPiGPIO/versions/2`
1	`arn:aws:greengrass:  region::/connectors/RaspberryPiGPIO/versions/1`

Pour obtenir des informations sur les changements apportés aux versions, veuillez consulter le Journal des modifications.

Prérequis

Ce connecteur possède les critères suivants :

Version 3

- Logiciel AWS IoT Greengrass Core 1.9.3 ou version ultérieure.
- Pythonversion 3.7 installée sur l'appareil principal et ajoutée à la variable d'environnement PATH.
- Raspberry Pi 4 modèle B ou Raspberry Pi 3 modèle B/B+. Vous devez connaître la séquence de broches de votre Raspberry Pi. Pour plus d'informations, consultez the section called “Séquence de broches GPIO”.
- UNressource de périphérique localedans le groupe Greengrass qui pointe vers `/dev/gpiomem`sur le Raspberry Pi. Si vous créez la ressource dans la console, vous devez sélectionnerAjouter automatiquement les autorisations de groupe de système d'exploitation du groupe Linux qui possède la ressourceoption. Dans l'API, définissez`GroupOwnerSetting.AutoAddGroupOwner`propriété vers`true`.
- Le module RPi.GPIO installé sur le Raspberry Pi. Dans Raspbian, ce module est installé par défaut. Vous pouvez utiliser la commande suivante pour le réinstaller :

  ```
  sudo pip install RPi.GPIO
  ```

Versions 1 - 2

- AWS IoT GreengrassLogiciel Core v1.7 ou version ultérieure.
- PythonVersion 2.7 installée sur l'appareil principal et ajoutée à la variable d'environnement PATH.
- Raspberry Pi 4 modèle B ou Raspberry Pi 3 modèle B/B+. Vous devez connaître la séquence de broches de votre Raspberry Pi. Pour plus d'informations, consultez the section called “Séquence de broches GPIO”.
- UNressource de périphérique localedans le groupe Greengrass qui pointe vers `/dev/gpiomem`sur le Raspberry Pi. Si vous créez la ressource dans la console, vous devez sélectionnerAjouter automatiquement les autorisations de groupe de système d'exploitation du groupe Linux qui possède la ressourceoption. Dans l'API, définissez`GroupOwnerSetting.AutoAddGroupOwner`propriété vers`true`.
- Le module RPi.GPIO installé sur le Raspberry Pi. Dans Raspbian, ce module est installé par défaut. Vous pouvez utiliser la commande suivante pour le réinstaller :

  ```
  sudo pip install RPi.GPIO
  ```

Séquence de broches GPIO

Le connecteur GPIO du Raspberry Pi référence les broches GPIO par le schéma de numérotation du système sur puce (SoC) sous-jacent, et non par la disposition physique des broches GPIO. L'ordre physique des broches peut varier selon les versions du Raspberry Pi. Pour de plus amples informations, veuillez consulterGPIOdans la documentation Raspberry Pi.

Le connecteur ne peut pas valider que les broches d'entrée et de sortie que vous configurez se mappent correctement au matériel sous-jacent de votre Raspberry Pi. Si la configuration de broche n'est pas valide, le connecteur renvoie une erreur d'exécution lorsqu'il essaie de démarrer sur le périphérique. Pour résoudre ce problème, reconfigurez le connecteur puis redéployez.

> **Note**
>
> Assurez-vous que les périphériques des broches GPIO sont correctement connectés pour empêcher les dommages des composants.

Paramètres du connecteur

Ce connecteur fournit les paramètres suivants :

`InputGpios`

Une liste séparée par des virgules de numéros de broches GPIO à configurer en tant qu'entrées. Vous pouvez également ajouter `U` pour définir une résistance de tirage de la broche ou `D` pour définir la résistance de rappel. Exemple: `"5,6U,7D"`.

Nom d'affichage dansAWS IoTConsole : Broches d'entrée GPIO

Obligatoire : `false`. Vous devez spécifier les broches d'entrée, de sortie ou les deux.

Type: `string`

Modèle valide :`^$|^[0-9]+[UD]?(,[0-9]+[UD]?)*$`

`InputPollPeriod`

Intervalle (en millisecondes) entre chaque opération d'interrogation, qui vérifie les broches GPIO d'entrée pour les changements d'état. La valeur minimale est de 1.

Cette valeur dépend de votre scénario et des types d'appareils qui sont interrogés. Par exemple, une valeur `50` doit être suffisamment rapide pour détecter une pression de bouton.

Nom d'affichage dansAWS IoTConsole : Période d'interrogation de GPIO

Obligatoire :`false`

Type: `string`

Modèle valide :`^$|^[1-9][0-9]*$`

`OutputGpios`

Une liste séparée par des virgules de nombres de broches GPIO à configurer en tant que sorties. Vous pouvez ajouter `H` pour régler sur un état haut (1) ou `L` pour régler sur un état faible (0). Exemple: `"8H,9,27L"`.

Nom d'affichage dansAWS IoTConsole : Broches de sortie GPIO

Obligatoire : `false`. Vous devez spécifier les broches d'entrée, de sortie ou les deux.

Type: `string`

Modèle valide :`^$|^[0-9]+[HL]?(,[0-9]+[HL]?)*$`

`GpioMem-ResourceId`

ID de la ressource de l'appareil local qui représente `/dev/gpiomem`.

> **Note**
>
> Ce connecteur bénéficie d'un accès en lecture/écriture à la ressource.

Nom d'affichage dansAWS IoTConsole : Resource pour appareil /dev/gpiomem

Obligatoire :`true`

Type: `string`

Modèle valide :`.+`

Exemple de création de connecteur (AWS CLI)

La commande d'interface de ligne de commande suivante crée`ConnectorDefinition`avec une version initiale qui contient le connecteur 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 fonction Lambda de ce connecteur possède unlongue duréecycle de vie.

DansAWS IoT Greengrass, vous pouvez ajouter un connecteur à partir de la consoleConnecteurs. Pour plus d'informations, consultez the section called “Démarrer avec les connecteurs (console)”.

Données d'entrée

Ce connecteur accepte les demandes de lecture ou d'écriture pour les broches GPIO sur deux rubriques MQTT.

- Lire les demandes dans la rubrique `gpio/+/+/read`.
- Écrire les demandes dans la rubrique `gpio/+/+/write`.

Pour publier dans ces rubriques, remplacez respectivement les caractères génériques + par le nom d'objet principal et le nombre de broches cibles. Par exemple :

```
gpio/core-thing-name/gpio-number/read
```

> **Note**
>
> Actuellement, lorsque vous créez un abonnement qui utilise le connecteur GPIO pour Raspberry Pi, vous devez spécifier une valeur pour au moins un des caractères génériques de la rubrique.

Filtre de rubriques : `gpio/+/+/read`

Utilisez cette rubrique pour demander au connecteur de lire l'état de la broche GPIO spécifiée dans la rubrique.

Le connecteur publie la réponse dans la rubrique de sortie correspondante (par exemple, `gpio/core-thing-name/gpio-number/state`).

Propriétés des messages

Aucun. Les messages qui sont envoyés dans cette rubrique sont ignorés.

Filtre de rubriques : `gpio/+/+/write`

Utilisez cette rubrique pour envoyer des demandes d'écriture à une broche GPIO. Cela demande au connecteur de régler la broche GPIO spécifiée dans la rubrique sur faible ou haute tension.

- `0` règle la broche sur faible tension.
- `1` règle la broche sur haute tension.

Le connecteur publie la réponse dans la rubrique `/state` de sortie correspondante (par exemple, `gpio/`*`core-thing-name`*`/`*`gpio-number`*`/state`).

Propriétés des messages

Valeur `0` ou `1`, sous forme d'un nombre entier ou d'une chaîne.

Exemple d'entrée

```
0
```

Données de sortie

Ce connecteur publie des données dans deux rubriques :

- Changements d'états hauts ou faibles dans la rubrique `gpio/+/+/state`.
- Erreurs dans la rubrique `gpio/+/error`.

Filtre de rubriques : `gpio/+/+/state`

Utilisez cette rubrique pour écouter les changements d'état sur les broches d'entrée et les réponses pour les demandes de lecture. Le connecteur renvoie la chaîne `"0"` si la broche est dans un état faible ou `"1"` s'il est dans un état haut.

Lorsqu'il publie dans cette rubrique, le connecteur remplace respectivement les caractères génériques + par le nom d'objet principal et la broche cible. Par exemple :

```
gpio/core-thing-name/gpio-number/state
```

> **Note**
>
> Actuellement, lorsque vous créez un abonnement qui utilise le connecteur GPIO pour Raspberry Pi, vous devez spécifier une valeur pour au moins un des caractères génériques de la rubrique.

Exemple de sortie

```
0
```

Filtre de rubriques : `gpio/+/error`

Utilisez cette rubrique pour écouter les erreurs. Le connecteur publie dans cette rubrique suite à une demande non valide (par exemple, lorsqu'un changement d'état est demandé sur une broche d'entrée).

Lorsqu'il publie dans cette rubrique, le connecteur remplace respectivement le caractère générique + par le nom d'objet principal.

Exemple de sortie

```
{
   "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."
 }
```

Exemple d'utilisation

Suivez les étapes détaillées suivantes pour configurer un exemple de fonction Lambda Python 3.7 que vous pouvez utiliser pour tester le connecteur.

> **Note**
>
> - Si vous utilisez d'autres environnements d'exécution Python, vous pouvez créer un lien symbolique de Python 3.x vers Python 3.7.

- Les rubriques Démarrer avec les connecteurs (console) et Démarrer avec les connecteurs (CLI) contiennent des étapes détaillées qui vous montrent comment configurer et déployer un exemple de connecteur Twilio Notifications.

1. Veillez à répondre aux conditions requises pour le connecteur.
2. Créez et publiez une fonction Lambda qui envoie des données d'entrée au connecteur.

   Enregistrez l'exemple de code en tant que fichier PY. Téléchargez et décompressezAWS IoT GreengrassKit SDK Core pour Python. Ensuite, créez un package zip contenant le fichier PY et le dossier `greengrasssdk` au niveau racine. Ce package zip correspond au package de déploiement que vous chargez surAWS Lambda.

   Après avoir créé la fonction Lambda Python 3.7, publiez une version de fonction et créez un alias.
3. Configurez votre groupe Greengrass.

   a. Ajoutez la fonction Lambda par son alias (recommandé). Configurez le cycle de vie Lambda comme long (ou`"Pinned": true`dans la CLI).

   b. Ajoutez la ressource de périphérique local requise et accordez un accès en lecture/écriture à la fonction Lambda.

   c. Ajoutez le connecteur et configurez ses paramètres.

   d. Ajoutez des abonnements qui permettent au connecteur de recevoir des données d'entrée et d'envoyer des données de sortie sur des filtres de rubrique pris en charge.

      - Définissez la fonction Lambda en tant que source, le connecteur en tant que cible et utilisez un filtre de rubrique d'entrée pris en charge.
      - Définissez le connecteur en tant que source, AWS IoT Core en tant que cible et utilisez un filtre de rubrique de sortie pris en charge. Vous utilisez cet abonnement pour afficher les messages d'état dansAWS IoTconsole
4. Déployez le groupe.
5. DansAWS IoTsur la console,Test, abonnez-vous à la rubrique des données de sortie pour afficher les messages d'état du connecteur. L'exemple de fonction Lambda à longue durée de vie commence à envoyer des messages immédiatement après le déploiement du groupe.

Lorsque vous avez terminé le test, vous pouvez définir le cycle de vie Lambda sur`"Pinned": false`dans l'interface de ligne de commande) et déployez le groupe. Cela empêche la fonction d'envoyer des messages.

Exemple

L'exemple de fonction Lambda suivant envoie un message d'entrée au connecteur. Cet exemple envoie des demandes de lecture pour un ensemble de broches GPIO d'entrée. Il indique comment construire des rubriques à l'aide du nom de l'objet principale et du nombre de broches.

```
import greengrasssdk
import json
import os

iot_client = greengrasssdk.client('iot-data')
INPUT_GPIOS = [6, 17, 22]

thingName = os.environ['AWS_IOT_THING_NAME']

def get_read_topic(gpio_num):
    return '/'.join(['gpio', thingName, str(gpio_num), 'read'])

def get_write_topic(gpio_num):
    return '/'.join(['gpio', thingName, str(gpio_num), 'write'])

def send_message_to_connector(topic, message=''):
    iot_client.publish(topic=topic, payload=str(message))

def set_gpio_state(gpio, state):
    send_message_to_connector(get_write_topic(gpio), str(state))

def read_gpio_state(gpio):
    send_message_to_connector(get_read_topic(gpio))

def publish_basic_message():
    for i in INPUT_GPIOS:
     read_gpio_state(i)

publish_basic_message()

def lambda_handler(event, context):
```

```
    return
```

Licences

Le connecteur Raspberry Pi inclut les logiciels et licences tiers suivants :

- RPi.GPIO/MIT

Ce connecteur est libéré sous leContrat de licence du logiciel Greengrass Core.

Journal des modifications

Le tableau suivant décrit les modifications apportées à chaque version du connecteur.

Version	Modifications
3	Mise à niveau du moteur d'exécution Lambda vers Python 3.7, ce qui modifie l'exigence d'environnement exécution.
2	Mise à jour d'ARN du connecteurRégion AWSPrise en charge.
1	Première version.

Un groupe Greengrass ne peut contenir qu'une seule version du connecteur à la fois. Pour de plus amples informations sur la mise à niveau d'une version de connecteur, veuillez consulter the section called “Mise à niveau des versions du connecteur”.

Consulter aussi

- Intégrer à des services et protocoles à l'aide de connecteurs
- the section called “Démarrer avec les connecteurs (console)”
- the section called “Démarrer avec les connecteurs (CLI)”
- GPIO dans la documentation Raspberry Pi

Connecteur Serial Stream

> **Warning**
>
> Ce connecteur a été déplacé dans lePhase de vie, etAWS IoT Greengrassne publiera pas de mises à jour fournissant des fonctionnalités, des améliorations à des fonctionnalités existantes, des correctifs de sécurité ou des corrections de bogues. Pour plus d'informations, consultez AWS IoT Greengrass Version 1politique de maintenance.

Le flux de sérieconnecteurlit et écrit dans un port série sur unAWS IoT GreengrassAppareil Core.

Ce connecteur prend en charge deux modes de fonctionnement :

- la lecture à la demande. Reçoit les demandes de lecture et d'écriture dans les rubriques MQTT et publie la réponse de l'opération de lecture ou de l'état de l'opération d'écriture.
- l'attente active de lecture. Lit à partir du port série à intervalles réguliers. Ce mode prend également en charge les requêtes de lecture à la demande.

> **Note**
>
> Les requêtes de lecture sont limitées à une longueur de lecture maximale de 63 994 octets. Les requêtes d'écriture sont limitées à une longueur de données maximale de 128 000 octets.

Ce connecteur a les versions suivantes.

Version	ARN
3	`arn:aws:greengrass:  region::/connectors/SerialStream/versions/3`
2	`arn:aws:greengrass:  region::/connectors/SerialStream/versions/2`

Version	ARN
1	`arn:aws:greengrass:  region::/connectors/SerialStream/versions/1`

Pour obtenir des informations sur les changements apportés aux versions, veuillez consulter le Journal des modifications.

Prérequis

Ce connecteur possède les critères suivants :

Version 3

- Logiciel AWS IoT Greengrass Core 1.9.3 ou version ultérieure.
- Pythonversion 3.7 ou 3.8 installée sur l'appareil principal et ajoutée à la variable d'environnement PATH.

  > **Note**
  >
  > Pour utiliser Python 3.8, exécutez la commande suivante pour créer un lien symbolique entre le dossier d'installation de Python 3.7 par défaut et les binaires Python 3.8 installés.
  >
  > ```
  > sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
  > ```
  >
  > Ceci configure votre appareil de sorte qu'il réponde aux exigence de Python pour AWS IoT Greengrass.

- UNressource de périphérique localedans le groupe Greengrass qui pointe vers le port série cible.

  > **Note**
  >
  > Avant de déployer ce connecteur, nous vous recommandons de configurer le port série et de vérifier que vous pouvez y accéder en lecture et en écriture.

Versions 1 - 2

- AWS IoT GreengrassLogiciel Core v1.7 ou version ultérieure.
- Pythonversion 2.7 installée sur l'appareil principal et ajoutée à la variable d'environnement PATH.
- UNressource de périphérique localedans le groupe Greengrass qui pointe vers le port série cible.

> **Note**
>
> Avant de déployer ce connecteur, nous vous recommandons de configurer le port série et de vérifier que vous pouvez y accéder en lecture et en écriture.

Paramètres de connecteur

Ce connecteur fournit les paramètres suivants :

`BaudRate`

Vitesse de transmission de la connexion série.

Nom d'affichage dans leAWS IoTConsole : Fréquence de baud

Obligatoire :`true`

Type: `string`

Valeurs valides : `110, 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 56000, 57600, 115200, 230400`

Modèle d'événement :`^110$|^300$|^600$|^1200$|^2400$|^4800$|^9600$|^14400$|^19200$|^28800$|^38400$|^56000$|^57600$|^115200$|^230400$`

`Timeout`

Délai (en secondes) pour une opération de lecture.

Nom d'affichage dans leAWS IoTConsole : Expiration

Obligatoire :`true`

Type: `string`

Valeurs valides : `1 - 59`

Modèle d'événement :`^([1-9]|[1-5][0-9])$`

`SerialPort`

Chemin d'accès absolu au port série physique sur le périphérique. Il s'agit du chemin d'accès source qui est spécifié pour la ressource du périphérique local.

Nom d'affichage dans leAWS IoTConsole : Port série

Obligatoire :`true`

Type: `string`

Modèle d'événement :`[/a-zA-Z0-9_-]+`

`SerialPort-ResourceId`

ID de la ressource de l'appareil local qui représente le port série physique.

> **Note**
>
> Ce connecteur bénéficie d'un accès en lecture/écriture à la ressource.

Nom d'affichage dans leAWS IoTConsole : Ressource du port série

Obligatoire :`true`

Type: `string`

Modèle d'événement :`[a-zA-Z0-9_-]+`

`PollingRead`

Définit le mode de lecture : attente active de lecture ou lecture à la demande.

- Pour le mode d'attente active de lecture, spécifiez `true`. Dans ce mode, les propriétés `PollingInterval`, `PollingReadType` et `PollingReadLength` sont obligatoires.
- Pour le mode de lecture à la demande, spécifiez `false`. Dans ce mode, le type et les valeurs de longueur sont spécifiés dans la demande de lecture.

Nom d'affichage dans leAWS IoTConsole : Mode lecture

Obligatoire :`true`

Type: `string`

Valeurs valides : `true, false`

Modèle d'événement :`^([Tt][Rr][Uu][Ee]|[Ff][Aa][Ll][Ss][Ee])$`

`PollingReadLength`

Longueur des données (en octets) à lire dans chaque opération d'attente active de lecture. Cela s'applique uniquement lorsque vous utilisez le mode d'attente active de lecture.

Nom d'affichage dans leAWS IoTConsole : Longueur de l'attente active de lecture

Obligatoire : `false`. Cette propriété est obligatoire lorsque`PollingReadest``true`.

Type: `string`

Modèle d'événement :`^(|[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'intervalle (en secondes) auquel l'attente active de lecture a lieu. Cela s'applique uniquement lorsque vous utilisez le mode d'attente active de lecture.

Nom d'affichage dans leAWS IoTConsole : Intervalle d'attente active de lecture

Obligatoire : `false`. Cette propriété est obligatoire lorsque`PollingReadest``true`.

Type: `string`

Valeurs valides : 1 - 999

Modèle d'événement :`^(|[1-9]|[1-9][0-9]|[1-9][0-9][0-9])$`

`PollingReadType`

Type de données que les threads d'interrogation lisent. Cela s'applique uniquement lorsque vous utilisez le mode d'attente active de lecture.

Nom d'affichage dans leAWS IoTConsole : Type d'attente active de lecture

Obligatoire : `false`. Cette propriété est obligatoire lorsque`PollingReadest``true`.

Type: `string`

Valeurs valides : `ascii, hex`

Modèle d'événement :`^(|[Aa][Ss][Cc][Ii][Ii]|[Hh][Ee][Xx])$`

`RtsCts`

Indique s'il convient d'activer le contrôle de flux RTS/CTS. La valeur par défaut est `false`. Pour plus d'informations, consultez RTS, CTS et RTR.

Nom d'affichage dans leAWS IoTConsole : Contrôle de flux RTS/CTS

Obligatoire :`false`

Type: `string`

Valeurs valides : `true, false`

Modèle d'événement :`^(|[Tt][Rr][Uu][Ee]|[Ff][Aa][Ll][Ss][Ee])$`

`XonXoff`

Indique s'il convient d'activer le contrôle de flux de logiciel. La valeur par défaut est `false`. Pour plus d'informations, consultez Software flow control.

Nom d'affichage dans leAWS IoTConsole : Contrôle de flux de logiciel

Obligatoire :`false`

Type: `string`

Valeurs valides : `true, false`

Modèle d'événement :`^(|[Tt][Rr][Uu][Ee]|[Ff][Aa][Ll][Ss][Ee])$`

`Parity`

Parité du port série. La valeur par défaut est `N`. Pour plus d'informations, consultez Parity.

Nom d'affichage dans leAWS IoTConsole : Parité de port série

Obligatoire :`false`

Type: `string`

Valeurs valides : `N, E, O, S, M`

Modèle d'événement :`^(|[NEOSMneosm])$`

Exemple de création de connecteur (AWS CLI)

La commande suivante d'CLI crée une`ConnectorDefinition`avec une version initiale qui contient le connecteur Serial Stream. Elle configure le connecteur pour le mode d'attente active de lecture.

```
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"
            }
        }
    ]
}'
```

DansAWS IoT Greengrass, vous pouvez ajouter un connecteur à partir de la consoleConnecteurs. Pour plus d'informations, consultez the section called “Démarrer avec les connecteurs (console)”.

Données d'entrée

Ce connecteur accepte les demandes de lecture ou d'écriture pour les ports série dans deux rubriques MQTT. Les messages d'entrée doivent être au format JSON.

- Lire les demandes dans la rubrique `serial/+/read/#`.
- Écrire les demandes dans la rubrique `serial/+/write/#`.

Pour publier dans ces rubriques, remplacez le caractère générique + par le nom d'objet principal et le caractère générique # par le chemin d'accès au port série. Par exemple :

```
serial/core-thing-name/read/dev/serial-port
```

Filtre de rubriques : `serial/+/read/#`

Utilisez cette rubrique pour envoyer les demandes de lecture à la demande à une broche série. Les requêtes de lecture sont limitées à une longueur de lecture maximale de 63 994 octets.

Propriétés des messages

`readLength`

Longueur de données à lire à partir du port série.

Obligatoire :`true`

Type: `string`

Modèle d'événement :`^[1-9][0-9]*$`

`type`

Type de données à lire.

Obligatoire :`true`

Type: `string`

Valeurs valides : `ascii, hex`

Modèle d'événement :`(?i)^(ascii|hex)$`

`id`

ID arbitraire de la demande. Cette propriété est utilisée pour mapper une demande d'entrée à une réponse de sortie.

Obligatoire :`false`

Type: `string`

Modèle d'événement :`.+`

Exemple d'entrée

```
{
    "readLength": "30",
    "type": "ascii",
    "id": "abc123"
}
```

Filtre de rubriques : `serial/+/write/#`

Utilisez cette rubrique pour envoyer des demandes d'écriture à une broche série. Les requêtes d'écriture sont limitées à une longueur de données maximale de 128 000 octets.

Propriétés des messages

`data`

Chaîne à écrire dans le port série.

Obligatoire :`true`

Type: `string`

Modèle d'événement :`^[1-9][0-9]*$`

`type`

Type de données à lire.

Obligatoire :`true`

Type: `string`

Valeurs valides : `ascii, hex`

Modèle d'événement :`^(ascii|hex|ASCII|HEX)$`

`id`

ID arbitraire de la demande. Cette propriété est utilisée pour mapper une demande d'entrée à une réponse de sortie.

Obligatoire :`false`

Type: `string`

Modèle d'événement :`.+`

Exemple d'entrée : Demande ASCII

```
{
    "data": "random serial data",
    "type": "ascii",
    "id": "abc123"
```

```
}
```

Exemple d'entrée : demande hex

```
{
    "data": "base64 encoded data",
    "type": "hex",
    "id": "abc123"
}
```

Données de sortie

Le connecteur publie des données de sortie dans deux rubriques :

- Informations sur le statut à partir du connecteur dans la rubrique `serial/+/status/#`.
- réponses à partir des demandes de lecture dans la rubrique `serial/+/read_response/#`.

Pour publier dans cette rubrique, le connecteur remplace le caractère générique + par le nom d'objet principal et le caractère générique # par le chemin d'accès au port série. Par exemple :

```
serial/core-thing-name/status/dev/serial-port
```

Filtre de rubriques : `serial/+/status/#`

Utilisez cette rubrique pour écouter le statut des demandes de lecture et d'écriture. Si une propriété `id` est incluse dans la demande, elle est renvoyée dans la réponse.

Exemple de sortie : Succès

```
{
    "response": {
        "status": "success"
    },
    "id": "abc123"
}
```

Exemple de sortie : Échec

Un échec de réponse inclut une propriété `error_message` qui décrit l'erreur ou l'expiration rencontrée lors de l'opération de lecture ou d'écriture.

```
{
    "response": {
        "status": "fail",
        "error_message": "Could not write to port"
    },
    "id": "abc123"
}
```

Filtre de rubriques : `serial/+/read_response/#`

Utilisez cette rubrique pour recevoir les données de réponse à partir d'une opération de lecture. Les données de réponse sont codées en Base64 si le type est `hex`.

Exemple de sortie

```
{
    "data": "output of serial read operation"
    "id": "abc123"
}
```

Exemple d'utilisation

Suivez les étapes détaillées suivantes pour configurer un exemple de fonction Lambda Python 3.7 que vous pouvez utiliser pour tester le connecteur.

> **Note**
>
> - Si vous utilisez d'autres environnements d'exécution Python, vous pouvez créer un lien symbolique de Python 3.x vers Python 3.7.
> - Les rubriques Démarrer avec les connecteurs (console) et Démarrer avec les connecteurs (CLI) contiennent des étapes détaillées qui vous montrent comment configurer et déployer un exemple de connecteur Twilio Notifications.

1. Veillez à répondre aux conditions requises pour le connecteur.
2. Créez et publiez une fonction Lambda qui envoie des données d'entrée au connecteur.

   Enregistrez l'exemple de code en tant que fichier PY. Téléchargez et décompressez leAWS IoT GreengrassKit SDK Core pour Python. Ensuite, créez un package zip contenant le fichier PY

et le dossier `greengrasssdk` au niveau racine. Ce package zip correspond au package de déploiement que vous chargez surAWS Lambda.

Après avoir créé la fonction Lambda Python 3.7, publiez une version de fonction et créez un alias.

3. Configurez votre groupe Greengrass.

   a. Ajoutez la fonction Lambda par son alias (recommandé). Configurez le cycle de vie Lambda comme long (ou`"Pinned": true`dans la CLI).

   b. Ajoutez la ressource de périphérique local requise et accordez un accès en lecture/écriture à la fonction Lambda.

   c. Ajoutez le connecteur à votre groupe et configurez ses paramètres.

   d. Ajoutez au groupe des abonnements qui permettent au connecteur de recevoir des données d'entrée et d'envoyer des données de sortie sur des filtres de rubrique pris en charge.

      - Définissez la fonction Lambda en tant que source, le connecteur en tant que cible et utilisez un filtre de rubrique d'entrée pris en charge.
      - Définissez le connecteur en tant que source, AWS IoT Core en tant que cible et utilisez un filtre de rubrique de sortie pris en charge. Vous utilisez cet abonnement pour afficher les messages d'état dans laAWS IoTconsole

4. Déployez le groupe.

5. DansAWS IoTConsole, sur laTestabonnez-vous à la rubrique des données de sortie pour afficher les messages d'état du connecteur. L'exemple de fonction Lambda à longue durée de vie commence à envoyer des messages immédiatement après le déploiement du groupe.

   Lorsque vous avez terminé le test, vous pouvez définir le cycle de vie Lambda sur le type à la demande (ou`"Pinned": false`dans l'interface de ligne de commande) et déployez le groupe. Cela empêche la fonction d'envoyer des messages.

Exemple

L'exemple de fonction Lambda suivant envoie un message d'entrée au connecteur.

```
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
```

Licences

Le connecteur Serial Stream inclut les logiciels et licences tiers suivants :

- pyserial/BSD

Ce connecteur est libéré sous leContrat de licence du logiciel Greengrass Core.

Journal des modifications

Le tableau suivante décrit les modifications intervenues dans chaque version du connecteur.

Version	Modifications
3	Mise à niveau du moteur d'exécution Lambda vers Python 3.7, ce qui modifie l'exigence d'environnement exécution.
2	Mise à jour ARN du connecteur pourRégion AWSPrise en charge.

Version	Modifications
1	Première version.

Un groupe Greengrass peut contenir une seule version du connecteur à la fois. Pour de plus amples informations sur la mise à niveau d'une version de connecteur, veuillez consulter the section called “Mise à niveau des versions du connecteur”.

Consulter aussi

- Intégrer à des services et protocoles à l'aide de connecteurs
- the section called “Démarrer avec les connecteurs (console)”
- the section called “Démarrer avec les connecteurs (CLI)”

ServiceNow MetricBase Connecteur d'intégration

> **Warning**
>
> Ce connecteur a été déplacé dans lePhase de vie prolongée, etAWS IoT Greengrassne publiera pas de mises à jour fournissant des fonctionnalités, des améliorations à des fonctionnalités existantes, des correctifs de sécurité ou des corrections de bogues. Pour plus d'informations, consultez AWS IoT Greengrass Version 1politique de maintenance.

Le ServiceNow MetricBase Integrationconnecteurpublie les métriques de séries temporelles à partir de périphériques Greengrass ServiceNow MetricBase. Cela vous permet de stocker, analyser et visualiser les données en séries chronologiques à partir de l'environnement Greengrass Core et d'agir sur les événements locaux.

Ce connecteur reçoit des données en séries chronologiques dans une rubrique MQTT et publie les données dans le ServiceNow API à intervalles réguliers.

Vous pouvez utiliser ce connecteur pour prendre en charge des scénarios, tels que :

- Créer des alertes basées sur le seuil et des alarmes en fonction des données en séries chronologiques collectées depuis les appareils Greengrass.

- Utiliser des données de services chronologiques des appareils Greengrass avec des applications personnalisées reposant sur le ServiceNow .

Ce connecteur a les versions suivantes.

Version	ARN
4	`arn:aws:greengrass:region::/connectors/ServiceNowMetricBaseIntegration/versions/4`
3	`arn:aws:greengrass:region::/connectors/ServiceNowMetricBaseIntegration/versions/3`
2	`arn:aws:greengrass:region::/connectors/ServiceNowMetricBaseIntegration/versions/2`
1	`arn:aws:greengrass:region::/connectors/ServiceNowMetricBaseIntegration/versions/1`

Pour obtenir des informations sur les changements apportés aux versions, veuillez consulter le Journal des modifications.

Prérequis

Ce connecteur possède les critères suivants :

Version 3 - 4

- AWS IoT GreengrassLogiciel Core 1.9.3 ou version ultérieure.AWS IoT Greengrassdoit être configuré pour prendre en charge les secrets locaux, comme décrit dansExigences relatives aux secrets.

> **Note**
>
> Cette exigence inclut l'autorisation d'accès à vos Secrets Manager secrets. Si vous utilisez le rôle de service Greengrass par défaut, Greengrass est autorisé à obtenir les valeurs des secrets dont les noms commencent pargreengrass-.

- Pythonversion 3.7 ou 3.8 installée sur l'appareil principal et ajoutée à la variable d'environnement PATH.

  > **Note**
  >
  > Pour utiliser Python 3.8, exécutez la commande suivante pour créer un lien symbolique entre le dossier d'installation de Python 3.7 par défaut et les binaires Python 3.8 installés.
  >
  > ```
  > sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
  > ```
  >
  > Ceci configure votre appareil de sorte qu'il réponde aux exigence de Python pour AWS IoT Greengrass.

- UN ServiceNow avec un abonnement activé à MetricBase. DansEn outre, une métrique et une table de métriques doivent être créées dans le compte. Pour de plus amples informations, veuillez consulterMetricBasedans le ServiceNow .
- Un secret de type texte dansAWS Secrets Managerqui stocke le nom d'utilisateur et le mot de passe de connexion à votre ServiceNow instance avec authentification de base. Le secret doit contenir les clés « utilisateur » et « mot de passe » avec des valeurs correspondantes. Pour de plus amples informations, veuillez consulterCréation d'un secret basiquedans leAWS Secrets ManagerGuide de l'utilisateur.
- Ressource de secret dans le groupe Greengrass qui référence le secret du Secrets Manager. Pour plus d'informations, consultez Déployer des secrets sur Core.

Versions 1 - 2

- AWS IoT GreengrassLogiciel Core 1.7 ou version ultérieure.AWS IoT Greengrassdoit être configuré pour prendre en charge les secrets locaux, comme décrit dansExigences relatives aux secrets.

> **Note**
>
> Cette exigence inclut l'autorisation d'accès à vos Secrets Manager secrets. Si vous utilisez le rôle de service Greengrass par défaut, Greengrass est autorisé à obtenir les valeurs des secrets dont les noms commencent pargreengrass-.

- Pythonversion 2.7 installée sur l'appareil principal et ajoutée à la variable d'environnement PATH.
- UN ServiceNow avec un abonnement activé à MetricBase. DansEn outre, une métrique et une table de métriques doivent être créées dans le compte. Pour de plus amples informations, veuillez consulterMetricBasedans le ServiceNow .
- Un secret de type texte dansAWS Secrets Managerqui stocke le nom d'utilisateur et le mot de passe de connexion à votre ServiceNow instance avec authentification de base. Le secret doit contenir les clés « utilisateur » et « mot de passe » avec des valeurs correspondantes. Pour de plus amples informations, veuillez consulterCréation d'un secret basiquedans leAWS Secrets ManagerGuide de l'utilisateur.
- Ressource de secret dans le groupe Greengrass qui référence le secret du Secrets Manager. Pour plus d'informations, consultez Déployer des secrets sur Core.

Paramètres du connecteur

Ce connecteur fournit les paramètres suivants :

Version 4

`PublishInterval`

Nombre maximal de secondes à attendre entre la publication des événements ServiceNow. La valeur maximale est 900.

Le connecteur publie dans ServiceNow quand`PublishBatchSize`est atteinte`PublishInterval`Expires on.

Nom d'affichage dans laAWS IoTConsole : Intervalle de publication en secondes

Obligatoire :`true`

Type: `string`

Valeurs valides : `1 - 900`

Modèle de code :`[1-9]|[1-9]\d|[1-9]\d\d|900`

`PublishBatchSize`

Nombre maximal de valeurs de métriques qui peuvent être traitées par lot avant qu'elles ne soient publiées dans ServiceNow.

Le connecteur publie dans ServiceNow quand`PublishBatchSize`est atteinte`PublishInterval`Expires on.

Nom d'affichage dans laAWS IoTConsole : Publication de taille de lot

Obligatoire :`true`

Type: `string`

Modèle de code :`^[0-9]+$`

`InstanceName`

Nom de l'instance utilisée pour la connexion à ServiceNow.

Nom d'affichage dans laAWS IoTConsole : Nom de ServiceNow exemple

Obligatoire :`true`

Type: `string`

Modèle de code :`.+`

`DefaultTableName`

Nom de la table contenant le`GlideRecord`associé à la série chronologique MetricBase Base de données. La propriété `table` de la charge utile du message d'entrée peut être utilisée pour remplacer cette valeur.

Nom d'affichage dans laAWS IoTConsole : Nom de la table qui contient la métrique

Obligatoire :`true`

Type: `string`

Modèle de code :`.+`

`MaxMetricsToRetain`

Nombre maximal de métriques à enregistrer dans la mémoire avant qu'elles ne soient remplacées par de nouvelles métriques.

Cette limite s'applique lorsqu'il n'y a aucune connexion à Internet et que le connecteur démarre pour mettre en tampon les métriques à publier ultérieurement. Lorsque le tampon est plein, les métriques les plus anciennes sont remplacées par de nouvelles métriques.

> **Note**
>
> Les métriques ne sont pas enregistrées si le processus hôte du connecteur est interrompu. Par exemple, cela peut se produire pendant le déploiement du groupe ou lorsque le périphérique redémarre.

Cette valeur doit être supérieure à la taille du lot et suffisamment grande pour contenir les messages basés sur le taux entrant des messages MQTT.

Nom d'affichage dans laAWS IoTConsole : Nombre maximum de métriques à conserver en mémoire

Obligatoire :`true`

Type: `string`

Modèle de code :`^[0-9]+$`

`AuthSecretArn`

Le secret dans leAWS Secrets Managerqui stockez le ServiceNow Nom d'utilisateur et mot de passe. Il doit s'agir d'un secret de type texte. Le secret doit contenir les clés « utilisateur » et « mot de passe » avec des valeurs correspondantes.

Nom d'affichage dans laAWS IoTConsole : ARN du secret d'authentification

Obligatoire :`true`

Type: `string`

Modèle de code :`arn:aws:secretsmanager:[a-z0-9\-]+:[0-9]{12}:secret:([a-zA-Z0-9\\]+/)*[a-zA-Z0-9/_+=,.@\-]+-[a-zA-Z0-9]+`

`AuthSecretArn-ResourceId`

Ressource de secret dans le groupe qui référence le secret de Secrets Manager pour le ServiceNow Informations d'identification .

Nom d'affichage dans laAWS IoTConsole : Ressource du jeton d'authentification

Obligatoire :`true`

Type: `string`

Modèle de code :`.+`

`IsolationMode`

Mode conteneurisation de ce connecteur. La valeur par défaut est `GreengrassContainer`, ce qui signifie que le connecteur s'exécute dans un environnement d'exécution isolé à l'intérieur du conteneur AWS IoT Greengrass.

> **Note**
>
> Le paramètre de conteneurisation par défaut pour le groupe ne s'applique pas aux connecteurs.

Nom d'affichage dans laAWS IoTConsole : Mode d'isolation du conteneur

Obligatoire :`false`

Type: `string`

Valeurs valides : `GreengrassContainer` ou `NoContainer`

Modèle de code :`^NoContainer$|^GreengrassContainer$`

Version 1 - 3

`PublishInterval`

Nombre maximal de secondes à attendre entre la publication des événements ServiceNow. La valeur maximale est 900.

Le connecteur publie dans ServiceNow quand`PublishBatchSize`est atteinte`PublishInterval`Expires on.

Nom d'affichage dans laAWS IoTConsole : Intervalle de publication en secondes

Obligatoire :`true`

Type: `string`

Valeurs valides : `1 - 900`

Modèle de code :`[1-9]|[1-9]\d|[1-9]\d\d|900`

`PublishBatchSize`

Nombre maximal de valeurs de métriques qui peuvent être traitées par lot avant qu'elles ne soient publiées dans ServiceNow.

Le connecteur publie dans ServiceNow quand`PublishBatchSize`est atteinte`PublishInterval`Expires on.

Nom d'affichage dans laAWS IoTConsole : Publication de taille de lot

Obligatoire :`true`

Type: `string`

Modèle de code :`^[0-9]+$`

`InstanceName`

Nom de l'instance utilisée pour la connexion à ServiceNow.

Nom d'affichage dans laAWS IoTConsole : Nom de ServiceNow exemple

Obligatoire :`true`

Type: `string`

Modèle de code :`.+`

`DefaultTableName`

Nom de la table contenant le`GlideRecord`associé à la série chronologique MetricBase Base de données. La propriété `table` de la charge utile du message d'entrée peut être utilisée pour remplacer cette valeur.

Nom d'affichage dans laAWS IoTConsole : Nom de la table qui contient la métrique

Obligatoire :`true`

Type: `string`

Modèle de code :`.+`

`MaxMetricsToRetain`

Nombre maximal de métriques à enregistrer dans la mémoire avant qu'elles ne soient remplacées par de nouvelles métriques.

Cette limite s'applique lorsqu'il n'y a aucune connexion à Internet et que le connecteur démarre pour mettre en tampon les métriques à publier ultérieurement. Lorsque le tampon est plein, les métriques les plus anciennes sont remplacées par de nouvelles métriques.

> **Note**
>
> Les métriques ne sont pas enregistrées si le processus hôte du connecteur est interrompu. Par exemple, cela peut se produire pendant le déploiement du groupe ou lorsque le périphérique redémarre.

Cette valeur doit être supérieure à la taille du lot et suffisamment grande pour contenir les messages basés sur le taux entrant des messages MQTT.

Nom d'affichage dans laAWS IoTConsole : Nombre maximum de métriques à conserver en mémoire

Obligatoire :`true`

Type: `string`

Modèle de code :`^[0-9]+$`

`AuthSecretArn`

Le secret dans leAWS Secrets Managerqui stockez le ServiceNow Nom d'utilisateur et mot de passe. Il doit s'agir d'un secret de type texte. Le secret doit contenir les clés « utilisateur » et « mot de passe » avec des valeurs correspondantes.

Nom d'affichage dans laAWS IoTConsole : ARN du secret d'authentification

Obligatoire :`true`

Type: `string`

Modèle de code :`arn:aws:secretsmanager:[a-z0-9\-]+:[0-9]{12}:secret:([a-zA-Z0-9\\]+/)*[a-zA-Z0-9/_+=,.@\-]+-[a-zA-Z0-9]+`

`AuthSecretArn-ResourceId`

Ressource de secret dans le groupe qui référence le secret de Secrets Manager pour le ServiceNow Informations d'identification .

Nom d'affichage dans laAWS IoTConsole : Ressource du jeton d'authentification

Obligatoire :`true`

Type: `string`

Modèle de code :`.+`

Exemple de création de connecteur (AWS CLI)

La commande de ligne de commande suivante crée un élément`ConnectorDefinition`avec une version initiale qui contient le ServiceNow MetricBase Connecteur d'intégration.

```
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 fonction Lambda de ce connecteur possède unlongue duréeCycle de vie.

DansAWS IoT Greengrass, vous pouvez ajouter un connecteur à partir de la consoleConnecteurs. Pour plus d'informations, consultez the section called “Démarrer avec les connecteurs (console)”.

Données d'entrée

Ce connecteur accepte les métriques de séries temporelles dans une rubrique MQTT et publie les métriques dans ServiceNow. Les messages d'entrée doivent être au format JSON.

Filtre de rubrique dans l'abonnement

`servicenow/metricbase/metric`

Propriétés des messages

`request`

Informations sur la table, l'enregistrement et la métrique. Cette requête représente l'objet `seriesRef` dans une demande POST en séries chronologiques. Pour plus d'informations, consultez Clotho Time Series API - POST.

Obligatoire :`true`

Type :`object`qui inclut les propriétés suivantes :

`subject`

`sys_id` de l'enregistrement spécifique dans la table.

Obligatoire :`true`

Type: `string`

`metric_name`

Nom du champ de la métrique.

Obligatoire :`true`

Type: `string`

`table`

Nom du tableau dans lequel stocker l'enregistrement. Spécifiez cette valeur pour remplacer le paramètre `DefaultTableName`.

Obligatoire :`false`

Type: `string`

`value`

Valeur du point de données individuel.

Obligatoire :`true`

Type: `float`

`timestamp`

Horodatage du point de données individuel. La valeur par défaut est l'heure actuelle.

Obligatoire :`false`

Type: `string`

Exemple d'entrée

```
{
    "request": {
        "subject":"ef43c6d40a0a0b5700c77f9bf387afe3",
        "metric_name":"u_count",
        "table": "u_greengrass_app"
        "value": 1.0,
        "timestamp": "2018-10-14T10:30:00"
```

```
    }
}
```

Données de sortie

Ce connecteur publie des informations d'état sous forme de données de sortie dans une rubrique MQTT.

Filtre de rubrique dans l'abonnement

```
servicenow/metricbase/metric/status
```

Exemple de sortie : Succès

```
{
    "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
            }
        ]
    }
}
```

Exemple de sortie : Échec

```
{
    "response": {
        "error": "InvalidInputException",
        "error_message": "metric value is invalid",
        "status": "fail"
    }
```

```
}
```

> **Note**
>
> Si le connecteur détecte une erreur réessayable (par exemple, des erreurs de connexion), il tente à nouveau la publication dans le lot suivant.

Exemple d'utilisation

Suivez les étapes détaillées suivantes pour configurer un exemple de fonction Lambda Python 3.7 que vous pouvez utiliser pour tester le connecteur.

> **Note**
>
> - Si vous utilisez d'autres environnements d'exécution Python, vous pouvez créer un lien symbolique de Python 3.x vers Python 3.7.
> - Les rubriques Démarrer avec les connecteurs (console) et Démarrer avec les connecteurs (CLI) contiennent des étapes détaillées qui vous montrent comment configurer et déployer un exemple de connecteur Twilio Notifications.

1. Veillez à répondre aux conditions requises pour le connecteur.
2. Créez et publiez une fonction Lambda qui envoie des données d'entrée au connecteur.

   Enregistrez l'exemple de code en tant que fichier PY. Téléchargez et décompressez le fichierAWS IoT GreengrassSDK Core pour Python. Ensuite, créez un package zip contenant le fichier PY et le dossier `greengrasssdk` au niveau racine. Ce package zip correspond au package de déploiement que vous chargez surAWS Lambda.

   Après avoir créé la fonction Lambda Python 3.7, publiez une version de fonction et créez un alias.
3. Configurez votre groupe Greengrass.

   a. Ajoutez la fonction Lambda par son alias (recommandé). Configurez le cycle de vie Lambda comme long (ou`"Pinned": true`dans la CLI).

   b. Ajoutez la ressource secrète requise et accordez l'accès en lecture à la fonction Lambda.

c. Ajoutez le connecteur et configurez ses paramètres.

d. Ajoutez des abonnements qui permettent au connecteur de recevoir des données d'entrée et d'envoyer des données de sortie sur des filtres de rubrique pris en charge.

- Définissez la fonction Lambda en tant que source, le connecteur en tant que cible et utilisez un filtre de rubrique d'entrée pris en charge.
- Définissez le connecteur en tant que source, AWS IoT Core en tant que cible et utilisez un filtre de rubrique de sortie pris en charge. Vous utilisez cet abonnement pour afficher les messages d'état dans laAWS IoTconsole

4. Déployez le groupe.
5. DansAWS IoT, sur leTestabonnez-vous à la rubrique des données de sortie pour afficher les messages d'état du connecteur. L'exemple de fonction Lambda à longue durée de vie commence à envoyer des messages immédiatement après le déploiement du groupe.

   Lorsque vous avez terminé le test, vous pouvez définir le cycle de vie Lambda sur le type « à la demande »`"Pinned": false`dans l'interface de ligne de commande) et déployez le groupe. Cela empêche la fonction d'envoyer des messages.

Exemple

L'exemple de fonction Lambda suivant envoie un message d'entrée au connecteur.

```
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
```

Licences

Le ServiceNow MetricBase Le connecteur d'intégration inclut les logiciels et licences tiers suivants :

- pysnow/MIT

Ce connecteur est libéré sousContrat de licence du logiciel Greengrass Core.

Journal des modifications

Le tableau suivant décrit les modifications apportées à chaque version du connecteur.

Version	Modifications
4	Ajout du paramètre `IsolationMode` pour configurer le mode de conteneurisation du connecteur.
3	Mise à niveau du moteur d'exécution Lambda vers Python 3.7, ce qui modifie l'exigence d'environnement exécution.
2	Correctif pour réduire la journalisation excessive.
1	Première version.

Un groupe Greengrass peut contenir une seule version du connecteur à la fois. Pour de plus amples informations sur la mise à niveau d'une version de connecteur, veuillez consulter the section called “Mise à niveau des versions du connecteur”.

Consulter aussi

- Intégrer à des services et protocoles à l'aide de connecteurs
- the section called “Démarrer avec les connecteurs (console)”
- the section called “Démarrer avec les connecteurs (CLI)”

connecteur SNS

Le connecteur SNS publie des messages sur une rubrique Amazon SNS. Cela permet aux serveurs Web, aux adresses de messagerie et aux abonnés à d'autres messages de répondre à des événements dans le groupe Greengrass.

Ce connecteur reçoit des informations de message SNS dans une rubrique MQTT, puis envoie le message à une rubrique SNS spécifiée. Vous pouvez éventuellement utiliser des fonctions Lambda personnalisées pour implémenter une logique de filtrage ou de formatage sur les messages avant leur publication sur ce connecteur.

Ce connecteur possède les versions suivantes.

Version	ARN
4	`arn:aws:greengrass:region::/connectors/SNS/versions/4`
3	`arn:aws:greengrass:region::/connectors/SNS/versions/3`
2	`arn:aws:greengrass:region::/connectors/SNS/versions/2`
1	`arn:aws:greengrass:region::/connectors/SNS/versions/1`

Pour obtenir des informations sur les changements apportés aux versions, veuillez consulter le Journal des modifications.

Prérequis

Ce connecteur possède les critères suivants :

Version 3 - 4

- Logiciel AWS IoT Greengrass Core 1.9.3 ou version ultérieure.
- Python version 3.7 ou 3.8 installé sur le périphérique principal et ajouté à la variable d'environnement PATH.

  > **Note**
  >
  > Pour utiliser Python 3.8, exécutez la commande suivante pour créer un lien symbolique entre le dossier d'installation par défaut de Python 3.7 et les fichiers binaires Python 3.8 installés.
  >
  > ```
  > sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
  > ```
  >
  > Ceci configure votre appareil de sorte qu'il réponde aux exigence de Python pour AWS IoT Greengrass.

- Rubrique SNS configurée. Pour plus d'informations, consultez Création d'une rubrique Amazon SNS dans le Guide du développeur Amazon Simple Notification Service.
- Le rôle de groupe Greengrass est configuré pour autoriser l'`sns:Publish`action sur l'Amazon SNSTopic cible, comme illustré dans l'exemple de politique IAM suivant.

  ```
  {
      "Version":"2012-10-17",
      "Statement":[
          {
              "Sid":"Stmt1528133056761",
              "Action":[
                  "sns:Publish"
              ],
              "Effect":"Allow",
              "Resource":[
                  "arn:aws:sns:region:account-id:topic-name"
  ```

```
            ]
        }
    ]
}
```

Ce connecteur vous permet de remplacer la rubrique par défaut de façon dynamique dans la charge utile des messages d'entrée. Si votre implémentation utilise cette fonctionnalité, la politique IAM doit `sns:Publish` autoriser l'accès à tous les sujets cibles. Vous pouvez octroyer un accès précis ou conditionnel aux ressources (par exemple, en utilisant un schéma d'attribution de nom avec caractère générique *).

Pour l'exigence de rôle de groupe, vous devez configurer le rôle de manière à accorder les autorisations requises et à vous assurer que le rôle a été ajouté au groupe. Pour plus d'informations, consultez the section called “Gérer le rôle de groupe (console)” ou the section called “Gérer le rôle de groupe (interface de ligne de commande)”.

Versions 1 - 2

- AWS IoT GreengrassLogiciel principal v1.7 ou version ultérieure.
- Python version 2.7 installé sur le périphérique principal et ajouté à la variable d'environnement PATH.
- Rubrique SNS configurée. Pour plus d'informations, consultez Création d'une rubrique Amazon SNS dans le Guide du développeur Amazon Simple Notification Service.
- Le rôle de groupe Greengrass est configuré pour autoriser l'`sns:Publish`action sur l'Amazon SNSTopic cible, comme illustré dans l'exemple de politique IAM suivant.

```
{
    "Version":"2012-10-17",
    "Statement":[
        {
            "Sid":"Stmt1528133056761",
            "Action":[
                "sns:Publish"
            ],
            "Effect":"Allow",
            "Resource":[
                "arn:aws:sns:region:account-id:topic-name"
            ]
        }
```

```
    ]
}
```

Ce connecteur vous permet de remplacer la rubrique par défaut de façon dynamique dans la charge utile des messages d'entrée. Si votre implémentation utilise cette fonctionnalité, la politique IAM doit `sns:Publish` autoriser l'accès à tous les sujets cibles. Vous pouvez octroyer un accès précis ou conditionnel aux ressources (par exemple, en utilisant un schéma d'attribution de nom avec caractère générique *).

Pour l'exigence de rôle de groupe, vous devez configurer le rôle de manière à accorder les autorisations requises et à vous assurer que le rôle a été ajouté au groupe. Pour plus d'informations, consultez the section called “Gérer le rôle de groupe (console)” ou the section called “Gérer le rôle de groupe (interface de ligne de commande)”.

Paramètres du connecteur

Ce connecteur fournit les paramètres suivants :

Version 4

`DefaultSNSArn`

ARN de la rubrique SNS par défaut vers lequel publier les messages. La rubrique de destination peut être remplacée par la propriété `sns_topic_arn` dans la charge utile du message d'entrée.

> **Note**
>
> Le rôle de groupe doit permettre l'autorisation `sns:Publish` à toutes les rubriques cibles. Pour plus d'informations, consultez the section called “Prérequis”.

Nom affiché dans la AWS IoT console : ARN de la rubrique SNS par défaut

Nécessaire : `true`

Type : `string`

Modèle valide : `arn:aws:sns:([a-z]{2}-[a-z]+-\d{1}):(\d{12}):([a-zA-Z0-9-_]+)$`

`IsolationMode`

Mode conteneurisation de ce connecteur. La valeur par défaut est `GreengrassContainer`, ce qui signifie que le connecteur s'exécute dans un environnement d'exécution isolé à l'intérieur du conteneur AWS IoT Greengrass.

> **Note**
>
> Le paramètre de conteneurisation par défaut pour le groupe ne s'applique pas aux connecteurs.

Nom affiché dans la AWS IoT console : mode d'isolation du conteneur

Nécessaire : `false`

Type : `string`

Valeurs valides : `GreengrassContainer` ou `NoContainer`

Modèle valide : `^NoContainer$|^GreengrassContainer$`

Versions 1 - 3

`DefaultSNSArn`

ARN de la rubrique SNS par défaut vers lequel publier les messages. La rubrique de destination peut être remplacée par la propriété `sns_topic_arn` dans la charge utile du message d'entrée.

> **Note**
>
> Le rôle de groupe doit permettre l'autorisation `sns:Publish` à toutes les rubriques cibles. Pour plus d'informations, consultez the section called “Prérequis”.

Nom affiché dans la AWS IoT console : ARN de la rubrique SNS par défaut

Nécessaire : `true`

Type : `string`

Modèle valide : `arn:aws:sns:([a-z]{2}-[a-z]+-\d{1}):(\d{12}):([a-zA-Z0-9-_]+)$`

Exemple de création de connecteur (AWS CLI)

La commande CLI suivante crée un `ConnectorDefinition` avec une version initiale contenant le connecteur 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"
            }
        }
    ]
}'
```

Dans la AWS IoT Greengrass console, vous pouvez ajouter un connecteur depuis la page Connecteurs du groupe. Pour plus d'informations, consultez the section called “Démarrer avec les connecteurs (console)”.

Données d'entrée

Ce connecteur accepte les informations des messages SNS sur un sujet MQTT, puis publie le message tel quel sur le sujet SNS cible. Les messages d'entrée doivent être au format JSON.

Filtre de rubrique dans l'abonnement

`sns/message`

Propriétés des messages

`request`

Informations sur le message à envoyer à la rubrique SNS.

Nécessaire : `true`

Type : `object` qui inclut les propriétés suivantes :

`message`

Contenu du message sous forme de chaîne ou au format JSON. Pour obtenir des exemples, consultez Exemple d'entrée.

Pour envoyer le fichier JSON, la propriété `message_structure` doit être définie sur `json` et le message doit être un objet JSON encodé en chaîne qui contient une clé `default`.

Nécessaire : `true`

Type : `string`

Modèle valide : `.*`

`subject`

Objet du message.

Nécessaire : `false`

Type : texte ASCII, 100 caractères maximum. Il doit commencer par une lettre, un nombre ou un signe de ponctuation. Il ne doit pas inclure de sauts de ligne ou de caractères de contrôle.

Modèle valide : `.*`

`sns_topic_arn`

ARN de la rubrique SNS vers lequel publier les messages. S'il est spécifié, le connecteur publie dans cette rubrique au lieu de la rubrique par défaut.

> **Note**
>
> Le rôle de groupe doit permettre l'autorisation `sns:Publish` à n'importe quelles rubriques cibles. Pour plus d’informations, consultez the section called “Prérequis”.

Nécessaire : `false`

Type : `string`

Modèle valide : `arn:aws:sns:([a-z]{2}-[a-z]+-\d{1}):(\d{12}):([a-zA-Z0-9-_]+)$`

`message_structure`

Structure du message.

Obligatoire : `false`. Cela doit être spécifié pour envoyer un message JSON.

Type : `string`

Valeurs valides : `json`

`id`

ID arbitraire de la demande. Cette propriété est utilisée pour mapper une demande d'entrée à une réponse de sortie. Lorsque spécifiée, la propriété `id` dans l'objet de réponse est définie sur cette valeur. Si vous n'utilisez pas cette fonctionnalité, vous pouvez omettre cette propriété ou spécifier une chaîne vide.

Nécessaire : `false`

Type : `string`

Modèle valide : `.*`

Restrictions

La taille du message est délimitée par la taille maximale d'un message SNS de 256 Ko.

Exemple d'entrée : message de type chaîne

Cet exemple envoie un message de type chaîne. Il spécifie la propriété `sns_topic_arn` facultative, qui remplace la rubrique de destination par défaut.

```
{
    "request": {
        "subject": "Message subject",
        "message": "Message data",
        "sns_topic_arn": "arn:aws:sns:region:account-id:topic2-name"
    },
    "id": "request123"
```

```
}
```

Exemple d'entrée : message JSON

Cet exemple envoie un message sous forme d'objet JSON encodé en chaîne incluant la clé `default`.

```
{
    "request": {
        "subject": "Message subject",
        "message": "{ \"default\": \"Message data\" }",
        "message_structure": "json"
    },
    "id": "request123"
}
```

Données de sortie

Ce connecteur publie des informations d'état sous forme de données de sortie dans une rubrique MQTT.

Filtre de rubrique dans l'abonnement

`sns/message/status`

Exemple de sortie : réussite

```
{
    "response": {
        "sns_message_id": "f80a81bc-f44c-56f2-a0f0-d5af6a727c8a",
        "status": "success"
    },
    "id": "request123"
}
```

Exemple de sortie : échec

```
{
   "response" : {
        "error": "InvalidInputException",
        "error_message": "SNS Topic Arn is invalid",
        "status": "fail"
```

```
    },
    "id": "request123"
}
```

Exemple d'utilisation

Suivez les étapes de haut niveau suivantes pour configurer un exemple de fonction Lambda en Python 3.7 que vous pouvez utiliser pour tester le connecteur.

> **Note**
>
> - Si vous utilisez d'autres environnements d'exécution Python, vous pouvez créer un lien symbolique entre Python3.x et Python 3.7.
> - Les rubriques Démarrer avec les connecteurs (console) et Démarrer avec les connecteurs (CLI) contiennent des étapes détaillées qui vous montrent comment configurer et déployer un exemple de connecteur Twilio Notifications.

1. Veillez à répondre aux conditions requises pour le connecteur.

   Pour l'exigence de rôle de groupe, vous devez configurer le rôle de manière à accorder les autorisations requises et à vous assurer que le rôle a été ajouté au groupe. Pour plus d'informations, consultez the section called “Gérer le rôle de groupe (console)” ou the section called “Gérer le rôle de groupe (interface de ligne de commande)”.

2. Créez et publiez une fonction Lambda qui envoie des données d'entrée au connecteur.

   Enregistrez l'exemple de code en tant que fichier PY. Téléchargez et décompressez le SDK AWS IoT Greengrass principal pour Python. Ensuite, créez un package zip contenant le fichier PY et le dossier `greengrasssdk` au niveau racine. Ce package zip est le package de déploiement vers lequel vous effectuez le téléchargementAWS Lambda.

   Après avoir créé la fonction Lambda Python 3.7, publiez une version de la fonction et créez un alias.

3. Configurez votre groupe Greengrass.

   a. Ajoutez la fonction Lambda par son alias (recommandé). Configurez le cycle de vie Lambda comme étant de longue durée (ou dans `"Pinned": true` la CLI).

b. Ajoutez le connecteur et configurez ses paramètres.

c. Ajoutez des abonnements qui permettent au connecteur de recevoir des données d'entrée et d'envoyer des données de sortie sur des filtres de rubrique pris en charge.

- Définissez la fonction Lambda comme source, le connecteur comme cible et utilisez un filtre de rubrique d'entrée compatible.
- Définissez le connecteur en tant que source, AWS IoT Core en tant que cible et utilisez un filtre de rubrique de sortie pris en charge. Vous utilisez cet abonnement pour afficher les messages d'état dans la AWS IoT console.

4. Déployez le groupe.
5. Dans la AWS IoT console, sur la page Test, abonnez-vous à la rubrique des données de sortie pour consulter les messages d'état du connecteur. L'exemple de fonction Lambda a une longue durée de vie et commence à envoyer des messages immédiatement après le déploiement du groupe.

   Lorsque vous avez terminé les tests, vous pouvez définir le cycle de vie Lambda à la demande (ou `"Pinned": false` dans la CLI) et déployer le groupe. Cela empêche la fonction d'envoyer des messages.

Exemple

L'exemple suivant de fonction Lambda envoie un message d'entrée au connecteur.

```
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
```

Licences

Le connecteur SNS inclut les logiciels/licences tiers suivants :

- AWS SDK for Python (Boto3)/Licence Apache 2.0
- botocore/Licence Apache 2.0
- dateutil/Licence PSF
- docutils/Licence BSD, licence GPL (General Public License) GNU, licence Python Software Foundation, domaine public
- jmespath/Licence MIT
- s3transfer/Licence Apache 2.0
- urllib3/Licence MIT

Ce connecteur est publié dans le cadre du contrat de licence logicielle Greengrass Core.

Journal des modifications

Le tableau suivant décrit les modifications apportées à chaque version du connecteur.

Version	Modifications
4	Ajout du paramètre `IsolationMode` pour configurer le mode de conteneurisation du connecteur.
3	Mise à niveau de l'environnement d'exécution Lambda vers Python 3.7, ce qui modifie les exigences d'exécution.

Version	Modifications
2	Correctif pour réduire la journalisation excessive.
1	Première version.

Un groupe Greengrass ne peut contenir qu'une seule version du connecteur à la fois. Pour de plus amples informations sur la mise à niveau d'une version de connecteur, veuillez consulter the section called “Mise à niveau des versions du connecteur”.

Consultez aussi

- Intégrer à des services et protocoles à l'aide de connecteurs
- the section called “Démarrer avec les connecteurs (console)”
- the section called “Démarrer avec les connecteurs (CLI)”
- Publier une action dans la documentation Boto 3
- Qu'est-ce qu'Amazon Simple Notification Service ? dans le Guide du développeur Amazon Simple Notification Service

Connecteur d'intégration Splunk Enterprise

> **Warning**
>
> Ce connecteur a été déplacé dans lePhase de durée, etAWS IoT Greengrassne publiera pas de mises à jour fournissant des fonctionnalités, des améliorations à des fonctionnalités existantes, des correctifs de sécurité ou des corrections de bogues. Pour plus d'informations, consultez AWS IoT Greengrass Version 1politique de maintenance.

L'intégration Splunk Enterpriseconnecteurpublie des données à partir d'appareils Greengrass dans Splunk. Cela vous permet d'utiliser Splunk pour surveiller et analyser l'environnement Greengrass Core et d'agir sur les événements locaux. Le connecteur s'intègre avec HTTP Event Collector (HEC). Pour plus d'informations, consultez la section Introduction to HTTP Event Collector dans la documentation Splunk.

Ce connecteur reçoit la journalisation et les données d'événement dans une rubrique MQTT et publie les données en l'état dans l'API Splunk.

Vous pouvez utiliser ce connecteur pour prendre en charge des scénarios professionnels, par exemple :

- les opérateurs peuvent utiliser des données périodiques provenant de capteurs et d'actionneurs (par exemple, la lecture des températures, de la pression et de l'eau) pour déclencher des alarmes lorsque les valeurs dépassent un certain seuil.
- les développeurs utilisent des données collectées à partir de machines industrielles pour créer des modèles de ML qui peuvent surveiller l'équipement pour les problèmes potentiels.

Ce connecteur a les versions suivantes.

Version	ARN
4	`arn:aws:greengrass:  region::/connectors/SplunkIntegration/versions/4`
3	`arn:aws:greengrass:  region::/connectors/SplunkIntegration/versions/3`
2	`arn:aws:greengrass:  region::/connectors/SplunkIntegration/versions/2`
1	`arn:aws:greengrass:  region::/connectors/SplunkIntegration/versions/1`

Pour obtenir des informations sur les changements apportés aux versions, veuillez consulter le Journal des modifications.

Prérequis

Ce connecteur possède les critères suivants :

Version 3 - 4

- AWS IoT GreengrassLogiciel Core 1.9.3 ou version ultérieure.AWS IoT Greengrassdoit être configuré pour prendre en charge les secrets locaux, comme décrit dansExigences relatives aux secrets.

  > **Note**
  >
  > Cette exigence inclut l'autorisation d'accès à vos secrets Secrets Manager. Si vous utilisez le rôle de service Greengrass par défaut, Greengrass a l'autorisation d'obtenir les valeurs des secrets dont les noms commencent pargreengrass.

- Pythonversion 3.7 ou 3.8 installée sur l'appareil principal et ajoutée à la variable d'environnement PATH.

  > **Note**
  >
  > Pour utiliser Python 3.8, exécutez la commande suivante pour créer un lien symbolique entre le dossier d'installation de Python 3.7 par défaut et les binaires Python 3.8 installés.
  >
  > ```
  > sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
  > ```
  >
  > Ceci configure votre appareil de sorte qu'il réponde aux exigence de Python pour AWS IoT Greengrass.

- La fonctionnalité HTTP Event Collector doit être activée dans Splunk. Pour de plus amples informations, veuillez consulter Set up and use HTTP eEvent Collector in Splunk Web dans la documentation Splunk.
- Un secret de type texte dans AWS Secrets Manager qui stocke votre jeton Splunk HTTP Event Collector. Pour de plus amples informations, veuillez consulterÀ propos des jetons Event Collectordans la documentation Splunk etCréation d'un secret basiquedans leAWS Secrets ManagerGuide de l'utilisateur.

> **Note**
>
> Pour créer le secret dans la console Secrets Manager, entrez votre token sur letexte brutOnglet. N'incluez pas de guillemets ni d'autres mises en forme Dans l'API, spécifiez le jeton comme valeur pour le`SecretString`propriété.

- Une ressource du secret dans le groupe Greengrass qui référence le secret Secrets Manager. Pour plus d'informations, consultez Déployer des secrets sur Core.

Versions 1 - 2

- AWS IoT GreengrassLogiciel Core 1.7 ou version ultérieure.AWS IoT Greengrassdoit être configuré pour prendre en charge les secrets locaux, comme décrit dansExigences relatives aux secrets.

  > **Note**
  >
  > Cette exigence inclut l'autorisation d'accès à vos secrets Secrets Manager. Si vous utilisez le rôle de service Greengrass par défaut, Greengrass a l'autorisation d'obtenir les valeurs des secrets dont les noms commencent pargreengrass.

- Pythonversion 2.7 installée sur l'appareil principal et ajoutée à la variable d'environnement PATH.
- La fonctionnalité HTTP Event Collector doit être activée dans Splunk. Pour de plus amples informations, veuillez consulter Set up and use HTTP eEvent Collector in Splunk Web dans la documentation Splunk.
- Un secret de type texte dans AWS Secrets Manager qui stocke votre jeton Splunk HTTP Event Collector. Pour de plus amples informations, veuillez consulterÀ propos des jetons Event Collectordans la documentation Splunk etCréation d'un secret basiquedans leAWS Secrets ManagerGuide de l'utilisateur.

  > **Note**
  >
  > Pour créer le secret dans la console Secrets Manager, entrez votre token sur letexte brutOnglet. N'incluez pas de guillemets ni d'autres mises en forme Dans l'API, spécifiez le jeton comme valeur pour le`SecretString`propriété.

- Une ressource du secret dans le groupe Greengrass qui référence le secret Secrets Manager. Pour plus d'informations, consultez Déployer des secrets sur Core.

Paramètres du connecteur

Ce connecteur fournit les paramètres suivants :

Version 4

`SplunkEndpoint`

Point de terminaison de votre instance Splunk. Cette valeur doit contenir le protocole, le nom d'hôte et le port.

Nom d'affichage dans leAWS IoTConsole : Point de Point de terminaison

Obligatoire :`true`

Type: `string`

Modèle :`^(http:\/\/|https:\/\/)?[a-z0-9]+([-.]{1}[a-z0-9]+)*.[a-z]{2,5}(:[0-9]{1,5})?(\/.*)?$`

`MemorySize`

Quantité de mémoire (en Ko) allouée au connecteur.

Nom d'affichage dans leAWS IoTConsole : Taille de la mémoire

Obligatoire :`true`

Type: `string`

Modèle :`^[0-9]+$`

`SplunkQueueSize`

Nombre maximal d'éléments à enregistrer dans la mémoire avant que les éléments ne soient envoyés ou supprimés. Lorsque cette limite est atteinte, les éléments les plus anciens de la file d'attente sont remplacés par des éléments plus récents. Cette limite s'applique généralement lorsqu'il n'y a pas de connexion à Internet.

Nom d'affichage dans leAWS IoTConsole : Nombre maximum d'éléments à conserver

Obligatoire :`true`

Type: `string`

Modèle :`^[0-9]+$`

`SplunkFlushIntervalSeconds`

Intervalle (en secondes) pour publier les données reçues dans Splunk HEC. La valeur maximale est 900. Pour configurer le connecteur afin qu'il publie les éléments au fur et à mesure qu'elles sont reçues (sans traitement par lot), spécifiez 0.

Nom d'affichage dans leAWS IoTConsole : Intervalle de publication Splunk

Obligatoire :`true`

Type: `string`

Modèle :`[0-9]|[1-9]\d|[1-9]\d\d|900`

`SplunkTokenSecretArn`

Le secret dansAWS Secrets Managerqui stocke le jeton Splunk. Il doit s'agir d'un secret de type texte.

Nom d'affichage dans leAWS IoTConsole : ARN du secret du jeton d'authentification Splunk

Obligatoire :`true`

Type: `string`

Modèle :`arn:aws:secretsmanager:[a-z]{2}-[a-z]+-\d{1}:\d{12}?:secret:[a-zA-Z0-9-_]+-[a-zA-Z0-9-_]+`

`SplunkTokenSecretArn-ResourceId`

La ressource de secret dans le groupe Greengrass qui référence le secret Splunk.

Nom d'affichage dans leAWS IoTConsole : Ressource du jeton d'authentification Splunk Enterprise

Obligatoire :`true`

Type: `string`

Modèle :`.+`

`SplunkCustomCALocation`

Chemin d'accès au fichier d'autorité de certification (CA) personnalisée pour Splunk (par exemple, `/etc/ssl/certs/splunk.crt`).

Nom d'affichage dans leAWS IoTConsole : Emplacement du fichier du fichier d'autorité de certification personnalisée

Obligatoire :`false`

Type: `string`

Modèle :`^$|/.*`

`IsolationMode`

Mode conteneurisation de ce connecteur. La valeur par défaut est `GreengrassContainer`, ce qui signifie que le connecteur s'exécute dans un environnement d'exécution isolé à l'intérieur du conteneur AWS IoT Greengrass.

> **Note**
>
> Le paramètre de conteneurisation par défaut pour le groupe ne s'applique pas aux connecteurs.

Nom d'affichage dans leAWS IoTConsole : Mode d'isolation du conteneur

Obligatoire :`false`

Type: `string`

Valeurs valides : `GreengrassContainer` ou `NoContainer`

Modèle :`^NoContainer$|^GreengrassContainer$`

Version 1 - 3

`SplunkEndpoint`

Point de terminaison de votre instance Splunk. Cette valeur doit contenir le protocole, le nom d'hôte et le port.

Nom d'affichage dans leAWS IoTConsole : Point de Point de terminaison

Obligatoire :`true`

Type: `string`

Modèle :`^(http:\/\/|https:\/\/)?[a-z0-9]+([-.]{1}[a-z0-9]+)*.[a-z]{2,5}(:[0-9]{1,5})?(\/.*)?$`

`MemorySize`

Quantité de mémoire (en Ko) allouée au connecteur.

Nom d'affichage dans leAWS IoTConsole : Taille de la mémoire

Obligatoire :`true`

Type: `string`

Modèle :`^[0-9]+$`

`SplunkQueueSize`

Nombre maximal d'éléments à enregistrer dans la mémoire avant que les éléments ne soient envoyés ou supprimés. Lorsque cette limite est atteinte, les éléments les plus anciens de la file d'attente sont remplacés par des éléments plus récents. Cette limite s'applique généralement lorsqu'il n'y a pas de connexion à Internet.

Nom d'affichage dans leAWS IoTConsole : Nombre maximum d'éléments à conserver

Obligatoire :`true`

Type: `string`

Modèle :`^[0-9]+$`

`SplunkFlushIntervalSeconds`

Intervalle (en secondes) pour publier les données reçues dans Splunk HEC. La valeur maximale est 900. Pour configurer le connecteur afin qu'il publie les éléments au fur et à mesure qu'elles sont reçues (sans traitement par lot), spécifiez 0.

Nom d'affichage dans leAWS IoTConsole : Intervalle de publication Splunk

Obligatoire :`true`

Type: `string`

Modèle :`[0-9]|[1-9]\d|[1-9]\d\d|900`

`SplunkTokenSecretArn`

Le secret dansAWS Secrets Managerqui stocke le jeton Splunk. Il doit s'agir d'un secret de type texte.

Nom d'affichage dans leAWS IoTConsole : ARN du secret du jeton d'authentification Splunk

Obligatoire :`true`

Type: `string`

Modèle :`arn:aws:secretsmanager:[a-z]{2}-[a-z]+-\d{1}:\d{12}?:secret:[a-zA-Z0-9-_]+-[a-zA-Z0-9-_]+`

`SplunkTokenSecretArn-ResourceId`

La ressource de secret dans le groupe Greengrass qui référence le secret Splunk.

Nom d'affichage dans laAWS IoTConsole : Ressource du jeton d'authentification Splunk Enterprise

Obligatoire :`true`

Type: `string`

Modèle :`.+`

`SplunkCustomCALocation`

Chemin d'accès au fichier d'autorité de certification (CA) personnalisée pour Splunk (par exemple, `/etc/ssl/certs/splunk.crt`).

Nom d'affichage dans laAWS IoTConsole : Emplacement du fichier du fichier d'autorité de certification personnalisée

Obligatoire :`false`

Type: `string`

Modèle :`^$|/.*`

Exemple de création de connecteur (AWS CLI)

La commande CLI suivante crée un`ConnectorDefinition`avec une version initiale qui contient le connecteur d'intégration Splunk.

```
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 fonction Lambda de ce connecteur possède unlongue duréecycle de vie.

DansAWS IoT Greengrass, vous pouvez ajouter un connecteur à partir de la consoleConnecteurs. Pour plus d'informations, consultez the section called “Démarrer avec les connecteurs (console)”.

Données d'entrée

Ce connecteur accepte la journalisation et les données d'événement dans une rubrique MQTT et publie les données reçues en l'état dans l'API Splunk. Les messages d'entrée doivent être au format JSON.

Filtre de rubrique dans l'abonnement

`splunk/logs/put`

Propriétés des messages

`request`

Données d'événements à envoyer à l'API Splunk. Les événements doivent répondre aux spécifications de l'API services/collector.

Obligatoire :`true`

Type : `object`. Seul le`event`la propriété est obligatoire.

`id`

ID arbitraire de la demande. Cette propriété est utilisée pour mapper une demande d'entrée à un statut de sortie.

Obligatoire :`false`

Type: `string`

Restrictions

Toutes les limites imposées par l'API Splunk s'appliquent lorsque ce connecteur est utilisé. Pour plus d'informations, consultez services/collector.

Exemple d'entrée

```
{
    "request": {
        "event": "some event",
        "fields": {
            "severity": "INFO",
            "category": [
                "value1",
                "value2"
            ]
        }
    },
    "id": "request123"
}
```

Données de sortie

Ce connecteur publie des données de sortie dans deux rubriques :

- Informations de statut dans la rubrique `splunk/logs/put/status`.
- Erreurs dans la rubrique `splunk/logs/put/error`.

Filtre de rubriques : `splunk/logs/put/status`

Utilisez cette rubrique pour écouter le statut des demandes. Chaque fois que le connecteur envoie à l'API Splunk un lot de données reçues, il publie la liste des ID des demandes qui ont réussi et celles qui ont échoué.

Exemple de sortie

```
{
    "response": {
        "succeeded": [
            "request123",
            ...
        ],
        "failed": [
            "request789",
            ...
        ]
    }
}
```

Filtre de rubriques : `splunk/logs/put/error`

Utilisez cette rubrique pour écouter les erreurs du connecteur. La propriété `error_message` qui décrit l'erreur ou l'expiration rencontrée lors du traitement de la demande.

Exemple de sortie

```
{
    "response": {
        "error": "UnauthorizedException",
        "error_message": "invalid splunk token",
        "status": "fail"
    }
```

```
}
```

> **Note**
>
> Si le connecteur détecte une erreur réessayable (par exemple, des erreurs de connexion), il tente à nouveau la publication dans le lot suivant.

Exemple d'utilisation

Suivez les étapes détaillées suivantes pour configurer un exemple de fonction Lambda Python 3.7 que vous pouvez utiliser pour tester le connecteur.

> **Note**
>
> - Si vous utilisez d'autres environnements d'exécution Python, vous pouvez créer un lien symbolique de Python 3.x vers Python 3.7.
> - Les rubriques Démarrer avec les connecteurs (console) et Démarrer avec les connecteurs (CLI) contiennent des étapes détaillées qui vous montrent comment configurer et déployer un exemple de connecteur Twilio Notifications.

1. Veillez à répondre aux conditions requises pour le connecteur.
2. Créez et publiez une fonction Lambda qui envoie des données d'entrée au connecteur.

   Enregistrez l'exemple de code en tant que fichier PY. Téléchargez et décompressez le fichierAWS IoT GreengrassSDK de base pour Python. Ensuite, créez un package zip contenant le fichier PY et le dossier `greengrasssdk` au niveau racine. Ce package zip correspond au package de déploiement que vous chargez surAWS Lambda.

   Après avoir créé la fonction Lambda Python 3.7, publiez une version de fonction et créez un alias.
3. Configurez votre groupe Greengrass.

   a. Ajoutez la fonction Lambda par son alias (recommandé). Configurez le cycle de vie Lambda comme long (ou`"Pinned": true`dans la CLI).

   b. Ajoutez la ressource de secret requise et accordez l'accès en lecture à la fonction Lambda.

c. Ajoutez le connecteur et configurez ses paramètres.

d. Ajoutez des abonnements qui permettent au connecteur de recevoir des données d'entrée et d'envoyer des données de sortie sur des filtres de rubrique pris en charge.

   - Définissez la fonction Lambda en tant que source, le connecteur en tant que cible et utilisez un filtre de rubrique d'entrée pris en charge.
   - Définissez le connecteur en tant que source, AWS IoT Core en tant que cible et utilisez un filtre de rubrique de sortie pris en charge. Vous utilisez cet abonnement pour afficher les messages d'état dans laAWS IoTconsole

4. Déployez le groupe.

5. DansAWS IoTsur la console, dans leTest, abonnez-vous à la rubrique des données de sortie pour afficher les messages d'état du connecteur. L'exemple de fonction Lambda à longue durée de vie commence à envoyer des messages immédiatement après le déploiement du groupe.

   Lorsque vous avez terminé le test, vous pouvez définir le cycle de vie Lambda sur le type « à la demande » (`"Pinned": false`dans l'interface de ligne de commande) et déployez le groupe. Cela empêche la fonction d'envoyer des messages.

Exemple

L'exemple de fonction Lambda suivant envoie un message d'entrée au connecteur.

```
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
```

Licences

Ce connecteur est libéré sous leContrat de licence du logiciel Greengrass Core.

Journal des modifications

Le tableau suivant décrit les modifications apportées à chaque version du connecteur.

Version	Modifications
4	Ajout du paramètre `IsolationMode` pour configurer le mode de conteneurisation du connecteur.
3	Mise à niveau du moteur d'exécution Lambda vers Python 3.7, ce qui modifie l'exigence d'environnement d'exécution
2	Correctif pour réduire la journalisation excessive.
1	Première version.

Un groupe Greengrass peut contenir une seule version du connecteur à la fois. Pour de plus amples informations sur la mise à niveau d'une version de connecteur, veuillez consulter the section called “Mise à niveau des versions du connecteur”.

Consulter aussi

- Intégrer à des services et protocoles à l'aide de connecteurs
- the section called “Démarrer avec les connecteurs (console)”

- the section called “Démarrer avec les connecteurs (CLI)”

Connecteur Twilio Notifications

> **Warning**
>
> Ce connecteur a été déplacé dans lePhase de vie, etAWS IoT Greengrassne publiera pas de mises à jour fournissant des fonctionnalités, des améliorations à des fonctionnalités existantes, des correctifs de sécurité ou des corrections de bogues. Pour plus d'informations, consultez AWS IoT Greengrass Version 1politique de maintenance.

Les notifications Twilioconnecteurfait des appels téléphoniques automatisés ou envoie des SMS via Twilio. Vous pouvez utiliser ce connecteur pour envoyer des notifications en réponse à des événements dans le groupe Greengrass. Pour les appels téléphoniques, le connecteur peut transmettre un message vocal au destinataire.

Ce connecteur reçoit les informations du message Twilio dans une rubrique MQTT, puis déclenche une notification Twilio.

> **Note**
>
> Pour obtenir un didacticiel qui montre comment utiliser le connecteur Twilio Notifications, consultezthe section called “Démarrer avec les connecteurs (console)”outhe section called “Démarrer avec les connecteurs (CLI)”.

Ce connecteur a les versions suivantes.

Version	ARN
5	`arn:aws:greengrass:  region::/connectors/TwilioNotifications/versions/5`
4	`arn:aws:greengrass:  region::/connectors/TwilioNotifications/versions/4`

Version	ARN
3	`arn:aws:greengrass:  region::/connectors/TwilioNotifications/versions/3`
2	`arn:aws:greengrass:  region::/connectors/TwilioNotifications/versions/2`
1	`arn:aws:greengrass:  region::/connectors/TwilioNotifications/versions/1`

Pour obtenir des informations sur les changements apportés aux versions, veuillez consulter le Journal des modifications.

Prérequis

Ce connecteur possède les critères suivants :

Version 4 - 5

- AWS IoT GreengrassLogiciel Core 1.9.3 ou version ultérieure.AWS IoT Greengrassdoit être configuré pour prendre en charge les secrets locaux, comme décrit dansExigences relatives aux secrets.

  > **Note**
  >
  > Cette exigence inclut l'autorisation d'accès à vos secrets Secrets Manager. Si vous utilisez le rôle de service Greengrass par défaut, Greengrass possède l'autorisation d'obtenir les valeurs des secrets dont les noms commencent pargreengrass..

- Pythonversion 3.7 ou 3.8 installée sur l'appareil principal et ajoutée à la variable d'environnement PATH.

> **Note**
>
> Pour utiliser Python 3.8, exécutez la commande suivante pour créer un lien symbolique entre le dossier d'installation de Python 3.7 par défaut et les binaires Python 3.8 installés.
>
> ```
> sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
> ```
>
> Ceci configure votre appareil de sorte qu'il réponde aux exigence de Python pour AWS IoT Greengrass.

- Un SID de compte Twilio, un jeton d'authentification et un numéro de téléphone Twilio. Une fois que vous avez créé un projet Twilio, ces valeurs sont disponibles sur le tableau de bord du projet.

  > **Note**
  >
  > Vous pouvez utiliser un compte d'essai Twilio. Si vous utilisez un compte d'essai, vous devez ajouter des numéros de téléphone qui ne sont pas des destinataires Twilio à une liste de numéros de téléphone vérifiés. Pour de plus amples informations, veuillez consulterComment utiliser votre compte d'essai Twilio gratuit.

- Un secret de type texte dans AWS Secrets Manager qui stocke le jeton d'authentification Twilio. Pour de plus amples informations, veuillez consulterCréation d'un secret basiquedans leAWS Secrets ManagerGuide de l'utilisateur.

  > **Note**
  >
  > Pour créer le secret dans la console Secrets Manager, entrez votre token sur letexte brutOnglet ats N'incluez pas de guillemets ni d'autres mises en forme Dans l'API, spécifiez le jeton en tant que valeur pour`SecretString`propriété.

- Une ressource du secret dans le groupe Greengrass qui référence le secret du Secrets Manager. Pour plus d'informations, consultez Déployer des secrets sur Core.

Versions 1 - 3

- AWS IoT GreengrassLogiciel Core v1.7 ou version ultérieure.AWS IoT Greengrassdoit être configuré pour prendre en charge les secrets locaux, comme décrit dansExigences relatives aux secrets.

  > **Note**
  >
  > Cette exigence inclut l'autorisation d'accès à vos secrets Secrets Manager. Si vous utilisez le rôle de service Greengrass par défaut, Greengrass possède l'autorisation d'obtenir les valeurs des secrets dont les noms commencent pargreengrass..

- Pythonversion 2.7 installée sur l'appareil principal et ajoutée à la variable d'environnement PATH.
- Un SID de compte Twilio, un jeton d'authentification et un numéro de téléphone Twilio. Une fois que vous avez créé un projet Twilio, ces valeurs sont disponibles sur le tableau de bord du projet.

  > **Note**
  >
  > Vous pouvez utiliser un compte d'essai Twilio. Si vous utilisez un compte d'essai, vous devez ajouter des numéros de téléphone qui ne sont pas des destinataires Twilio à une liste de numéros de téléphone vérifiés. Pour de plus amples informations, veuillez consulterComment utiliser votre compte d'essai Twilio gratuit.

- Un secret de type texte dans AWS Secrets Manager qui stocke le jeton d'authentification Twilio. Pour de plus amples informations, veuillez consulterCréation d'un secret basiquedans leAWS Secrets ManagerGuide de l'utilisateur.

  > **Note**
  >
  > Pour créer le secret dans la console Secrets Manager, entrez votre token sur letexte brutOnglet ats N'incluez pas de guillemets ni d'autres mises en forme Dans l'API, spécifiez le jeton en tant que valeur pour`SecretString`propriété.

- Une ressource du secret dans le groupe Greengrass qui référence le secret du Secrets Manager. Pour plus d'informations, consultez Déployer des secrets sur Core.

Paramètres du connecteur

Ce connecteur fournit les paramètres suivants :

Version 5

`TWILIO_ACCOUNT_SID`

Le SID du compte Twilio qui est utilisé pour appeler l'API Twilio.

Nom d'affichage dans événementAWS IoTConsole : ACCOUNT du compte Twilio

Obligatoire :`true`

Type: `string`

Modèle de Modèle :`.+`

`TwilioAuthTokenSecretArn`

L'ARN du secret Secrets Manager qui stocke le jeton d'authentification Twilio.

> **Note**
>
> Il est utilisé pour accéder à la valeur du secret local sur le noyau.

Nom d'affichage dans événementAWS IoTConsole : ARN du secret du jeton d'authentification Twilio

Obligatoire :`true`

Type: `string`

Modèle de Modèle :`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 de la ressource du secret dans le groupe Greengrass qui référence le secret du jeton d'authentification Twilio.

Nom d'affichage dans événementAWS IoTConsole : Ressource du jeton d'authentification Twilio

Obligatoire :`true`

Type: `string`

Modèle de Modèle :`.+`

`DefaultFromPhoneNumber`

Le numéro de téléphone activé par Twilio que Twilio utilise pour envoyer des messages. Twilio utilise ce numéro pour initier le texte ou l'appel.

- Si vous ne configurez pas un numéro de téléphone par défaut, vous devez spécifier un numéro de téléphone dans la propriété `from_number` dans le corps du message d'entrée.
- Si vous configurez un numéro de téléphone par défaut, vous pouvez, si vous le souhaitez, remplacer le numéro par défaut en spécifiant la propriété `from_number` dans le corps du message d'entrée.

Nom d'affichage dans événementAWS IoTConsole : À partir de numéro de téléphone

Obligatoire :`false`

Type: `string`

Modèle de Modèle :`^$|\+[0-9]+`

`IsolationMode`

Mode conteneurisation de ce connecteur. La valeur par défaut est `GreengrassContainer`, ce qui signifie que le connecteur s'exécute dans un environnement d'exécution isolé à l'intérieur du conteneur AWS IoT Greengrass.

> **Note**
>
> Le paramètre de conteneurisation par défaut pour le groupe ne s'applique pas aux connecteurs.

Nom d'affichage dans événementAWS IoTConsole : Mode d'isolation du conteneur

Obligatoire :`false`

Type: `string`

Valeurs valides : `GreengrassContainer` ou `NoContainer`

Modèle de Modèle :`^NoContainer$|^GreengrassContainer$`

Version 1 - 4

`TWILIO_ACCOUNT_SID`

Le SID du compte Twilio qui est utilisé pour appeler l'API Twilio.

Nom d'affichage dans événementAWS IoTConsole : ACCOUNT du compte Twilio

Obligatoire :`true`

Type: `string`

Modèle de Modèle :`.+`

`TwilioAuthTokenSecretArn`

L'ARN du secret Secrets Manager qui stocke le jeton d'authentification Twilio.

> **Note**
>
> Il est utilisé pour accéder à la valeur du secret local sur le noyau.

Nom d'affichage dans laAWS IoTConsole : ARN du secret du jeton d'authentification Twilio

Obligatoire :`true`

Type: `string`

Modèle de Modèle :`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 de la ressource du secret dans le groupe Greengrass qui référence le secret du jeton d'authentification Twilio.

Nom d'affichage dans laAWS IoTConsole : Ressource du jeton d'authentification Twilio

Obligatoire :`true`

Type: `string`

Modèle de Modèle :`.+`

`DefaultFromPhoneNumber`

Le numéro de téléphone activé par Twilio que Twilio utilise pour envoyer des messages. Twilio utilise ce numéro pour initier le texte ou l'appel.

- Si vous ne configurez pas un numéro de téléphone par défaut, vous devez spécifier un numéro de téléphone dans la propriété `from_number` dans le corps du message d'entrée.
- Si vous configurez un numéro de téléphone par défaut, vous pouvez, si vous le souhaitez, remplacer le numéro par défaut en spécifiant la propriété `from_number` dans le corps du message d'entrée.

Nom d'affichage dans événementAWS IoTConsole : À partir de numéro de téléphone

Obligatoire :`false`

Type: `string`

Modèle de Modèle :`^$|\+[0-9]+`

Exemple de création de connecteur (AWS CLI)

L'exemple de commande CLI suivant crée un élément`ConnectorDefinition`avec une version initiale qui contient le connecteur 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"
            }
        }
    ]
```

```
}'
```

Pour consulter des didacticiels qui expliquent comment ajouter le connecteur Twilio Notifications à un groupe, consultezthe section called “Démarrer avec les connecteurs (CLI)”etthe section called “Démarrer avec les connecteurs (console)”.

Données d'entrée

Ce connecteur accepte les informations des messages Twilio dans deux rubriques MQTT. Les messages d'entrée doivent être au format JSON.

- Informations des SMS dans la rubrique `twilio/txt`.
- Informations des messages vocaux dans la rubrique `twilio/call`.

> **Note**
>
> La charge utile d'un message entrant peut inclure un SMS (`message`) ou un message vocal (`voice_message_location`), mais pas les deux.

Filtre des rubriques : **`twilio/txt`**

Propriétés des messages

`request`

Informations sur la notification Twilio.

Obligatoire :`true`

Type :`object`qui inclut les propriétés suivantes :

`recipient`

Destinataire du message. Un seul destinataire est pris en charge.

Obligatoire :`true`

Type :`object`qui inclut les propriétés suivantes :

`name`

le nom du destinataire.

Obligatoire :`true`

Type: `string`

Modèle de Modèle :`.*`

`phone_number`

le numéro de téléphone du destinataire.

Obligatoire :`true`

Type: `string`

Modèle de Modèle :`\+[1-9]+`

`message`

le contenu du message du SMS. Seuls les SMS sont pris en charge dans cette rubrique. Pour les messages vocaux, utilisez `twilio/call`.

Obligatoire :`true`

Type: `string`

Modèle de Modèle :`.+`

`from_number`

Le numéro de téléphone de l'expéditeur. Twilio utilise ce numéro pour initier le message. Cette propriété est obligatoire si le paramètre `DefaultFromPhoneNumber` n'est pas configuré. Si `DefaultFromPhoneNumber` est configuré, vous pouvez utiliser cette propriété pour remplacer le numéro par défaut.

Obligatoire :`false`

Type: `string`

Modèle de Modèle :`\+[1-9]+`

`retries`

Nombre de nouvelles tentatives. La valeur par défaut est 0.

Obligatoire :`false`

Type: `integer`

`id`

ID arbitraire de la demande. Cette propriété est utilisée pour mapper une demande d'entrée à une réponse de sortie.

Obligatoire :`true`

Type: `string`

Modèle de Modèle :`.+`

Exemple d'entrée

```
{
    "request": {
        "recipient": {
            "name": "Darla",
            "phone_number": "+12345000000",
            "message": "Hello from the edge"
        },
        "from_number": "+19999999999",
        "retries": 3
    },
    "id": "request123"
}
```

Filtre des rubriques : **`twilio/call`**

Propriétés des messages

`request`

Informations sur la notification Twilio.

Obligatoire :`true`

Type :`object`qui inclut les propriétés suivantes :

`recipient`

Destinataire du message. Un seul destinataire est pris en charge.

Obligatoire :`true`

Type :`object`qui inclut les propriétés suivantes :

`name`

le nom du destinataire.

Obligatoire :`true`

Type: `string`

Modèle de Modèle :`.+`

`phone_number`

le numéro de téléphone du destinataire.

Obligatoire :`true`

Type: `string`

Modèle de Modèle :`\+[1-9]+`

`voice_message_location`

URL du contenu audio du message vocal. Elle doit être en format TwiML. Seuls les messages vocaux sont pris en charge dans cette rubrique. Pour les SMS, utilisez `twilio/txt`.

Obligatoire :`true`

Type: `string`

Modèle de Modèle :`.+`

`from_number`

Le numéro de téléphone de l'expéditeur. Twilio utilise ce numéro pour initier le message. Cette propriété est obligatoire si le paramètre `DefaultFromPhoneNumber` n'est pas configuré. Si `DefaultFromPhoneNumber` est configuré, vous pouvez utiliser cette propriété pour remplacer le numéro par défaut.

Obligatoire :`false`

Type: `string`

Modèle de Modèle :`\+[1-9]+`

`retries`

Nombre de nouvelles tentatives. La valeur par défaut est 0.

Obligatoire :`false`

Type: `integer`

`id`

ID arbitraire de la demande. Cette propriété est utilisée pour mapper une demande d'entrée à une réponse de sortie.

Obligatoire :`true`

Type: `string`

Modèle de Modèle :`.+`

Exemple d'entrée

```
{
    "request": {
        "recipient": {
            "name": "Darla",
            "phone_number": "+12345000000",
            "voice_message_location": "https://some-public-TwiML"
        },
        "from_number": "+19999999999",
        "retries": 3
    },
    "id": "request123"
}
```

Données de sortie

Ce connecteur publie des informations d'état sous forme de données de sortie dans une rubrique MQTT.

Filtre de rubrique dans l'abonnement

`twilio/message/status`

Exemple de sortie : Succès

```
{
    "response": {
        "status": "success",
        "payload": {
            "from_number": "+19999999999",
            "messages": {
                "message_status": "queued",
                "to_number": "+12345000000",
                "name": "Darla"
            }
        }
    },
    "id": "request123"
}
```

Exemple de sortie : Échec

```
{
    "response": {
        "status": "fail",
        "error_message": "Recipient name cannot be None",
        "error": "InvalidParameter",
        "payload": None
        }
    },
    "id": "request123"
}
```

La propriété `payload` dans la sortie est la réponse de l'API Twilio lorsque le message est envoyé. Si le connecteur détecte que les données d'entrée ne sont pas valides (par exemple, elles ne spécifient pas un champ d'entrée obligatoire), le connecteur renvoie une erreur et définit la valeur sur `None`. Voici quelques exemples de charges utiles :

```
{
    'from_number':'+19999999999',
    'messages': {
        'name':'Darla',
        'to_number':'+12345000000',
        'message_status':'undelivered'
    }
```

```
}
```

```
{
    'from_number':'+19999999999',
    'messages': {
        'name':'Darla',
        'to_number':'+12345000000',
        'message_status':'queued'
    }
}
```

Exemple d'utilisation

Suivez les étapes détaillées suivantes pour configurer un exemple de fonction Lambda Python 3.7 que vous pouvez utiliser pour tester le connecteur.

> **Note**
>
> Lethe section called “Démarrer avec les connecteurs (console)”etthe section called “Démarrer avec les connecteurs (CLI)”les sujets contiennent end-to-end qui montrent comment configurer, déployer et tester le connecteur Twilio Notifications.

1. Veillez à répondre aux conditions requises pour le connecteur.
2. Créez et publiez une fonction Lambda qui envoie des données d'entrée au connecteur.

   Enregistrez l'exemple de code en tant que fichier PY. Téléchargez et décompressez le fichierAWS IoT GreengrassKit SDK de chemin d'accès. Ensuite, créez un package zip contenant le fichier PY et le dossier `greengrasssdk` au niveau racine. Ce package zip correspond au package de déploiement que vous chargez surAWS Lambda.

   Après avoir créé la fonction Lambda Python 3.7, publiez une version de fonction et créez un alias.
3. Configurez votre groupe Greengrass.

   a. Ajoutez la fonction Lambda par son alias (recommandé). Configurez le cycle de vie Lambda comme long (ou`"Pinned": true`dans la CLI).

   b. Ajoutez la ressource secrète requise et accordez l'accès en lecture à la fonction Lambda.

c. Ajoutez le connecteur et configurez ses paramètres.

d. Ajoutez des abonnements qui permettent au connecteur de recevoir des données d'entrée et d'envoyer des données de sortie sur des filtres de rubrique pris en charge.

   - Définissez la fonction Lambda en tant que source, le connecteur en tant que cible et utilisez un filtre de rubrique d'entrée pris en charge.
   - Définissez le connecteur en tant que source, AWS IoT Core en tant que cible et utilisez un filtre de rubrique de sortie pris en charge. Vous utilisez cet abonnement pour afficher les messages d'état dans laAWS IoTconsole

4. Déployez le groupe.
5. DansAWS IoTConsole,Test, abonnez-vous à la rubrique des données de sortie pour afficher les messages d'état du connecteur. L'exemple de fonction Lambda à longue durée de vie commence à envoyer des messages immédiatement après le déploiement du groupe.

   Lorsque vous avez terminé le test, vous pouvez définir le cycle de vie Lambda sur le type « à la demande » (`"Pinned": false`dans l'interface de ligne de commande) et déployez le groupe. Cela empêche la fonction d'envoyer des messages.

Exemple

L'exemple de fonction Lambda suivant envoie un message d'entrée au connecteur. Cet exemple déclenche un message texte.

```
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
```

Licats

Le connecteur Twilio Notifications inclut les logiciels et licences tiers suivants :

- twilio de python/MIT

Ce connecteur est relâché sousContrat de licence du logiciel Greengrass Core.

Journal des modifications

Le tableau suivante décrit les modifications apportées à chaque version du connecteur.

Version	Modifications
5	Ajout du paramètre `IsolationMode` pour configurer le mode de conteneurisation du connecteur.
4	Mise à niveau du moteur d'exécution Lambda vers Python 3.7, ce qui modifie l'exigence d'environnement exécution.
3	Correctif pour réduire la journalisation excessive.
2	Correctifs de bogues mineurs et améliorations.

Version	Modifications
1	Première version.

Un groupe Greengrass peut contenir une seule version du connecteur à la fois. Pour de plus amples informations sur la mise à niveau d'une version de connecteur, veuillez consulter the section called “Mise à niveau des versions du connecteur”.

Consulter aussi

- Intégrer à des services et protocoles à l'aide de connecteurs
- the section called “Démarrer avec les connecteurs (console)”
- the section called “Démarrer avec les connecteurs (CLI)”
- Twilio API Reference

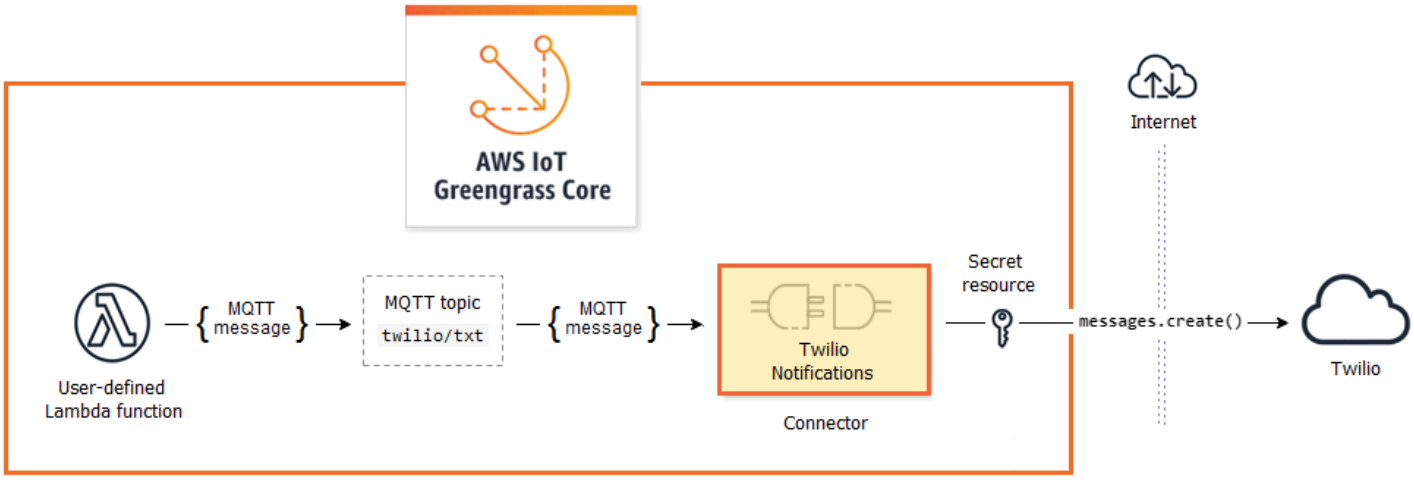
Mise en route avec les connecteurs Greengrass (console)

Cette fonction est disponible pourAWS IoT GreengrassCore v1.7 et versions ultérieures.

Ce didacticiel explique comment utiliser la AWS Management Console pour travailler avec les connecteurs.

Utilisez des connecteurs pour accélérer votre cycle de développement. Les connecteurs sont des modules réutilisables prédéfinis qui contribuent à interagir avec les services, les protocoles et les ressources. Ils peuvent vous aider à déployer plus rapidement une logique métier pour les appareils Greengrass. Pour plus d'informations, consultez Intégrer à des services et protocoles à l'aide de connecteurs.

Dans ce didacticiel, vous configurez et déployez leNotifications Twilioconnecteur. Le connecteur reçoit des informations du message Twilio en tant que données d'entrée, puis déclenche un SMS Twilio. Le flux de données est illustré dans le diagramme suivant.



Une fois que vous avez configuré le connecteur, vous créez une fonction Lambda et un abonnement.

- La fonction évalue les données simulées à partir d'un capteur de température. Elle publie les informations du message Twilio sous certaines conditions dans une rubrique MQTT. Il s'agit de la rubrique à laquelle le connecteur est abonné.
- L'abonnement autorise la fonction à publier dans la rubrique et le connecteur à recevoir des données provenant de la rubrique.

Le connecteur Twilio Notifications nécessite un jeton d'authentification Twilio pour interagir avec l'API Twilio. Le jeton est un secret de type texte dans AWS Secrets Manager et référencé à partir d'une ressource de groupe. Cela permet à AWS IoT Greengrass de créer une copie locale du secret sur le noyau Greengrass, où il est chiffré et mis à la disposition du connecteur. Pour plus d'informations, consultez Déployer des secrets sur Core.

Le didacticiel contient les étapes détaillées suivantes :

1. Créer un secret Secrets Manager
2. Ajouter une ressource de secret à un groupe
3. Ajouter un connecteur au groupe
4. Créer un package de déploiement de fonction Lambda
5. Création d'une fonction Lambda
6. Ajouter une fonction au groupe
7. Ajouter des abonnements au groupe

8. Déployer le groupe
9. the section called “Tester la solution”

Le didacticiel devrait prendre environ 20 minutes.

Prérequis

Pour suivre ce didacticiel, vous devez disposer des éléments suivants :

- Un groupe Greengrass et un appareil principal (noyau) Greengrass (version 1.9.3 ou ultérieure). Pour savoir comment créer un groupe et un service principal Greengrass, consultez Commencer avec AWS IoT Greengrass. Le didacticiel Mise en route comprend également les étapes d'installation du logiciel AWS IoT Greengrass Core.
- Python 3.7 installé sur l'appareil principal (noyau) AWS IoT Greengrass.
- AWS IoT Greengrassdoit être configuré pour prendre en charge les secrets locaux, comme décrit dansExigences relatives aux secrets.

  > **Note**
  >
  > Cette exigence inclut l'autorisation d'accès à vos secrets Secrets Manager. Si vous utilisez le rôle de service Greengrass par défaut, Greengrass est autorisé à obtenir les valeurs des secrets dont les noms commencent pargreengrass.

- Un SID de compte Twilio, un jeton d'authentification et un numéro de téléphone Twilio. Une fois que vous avez créé un projet Twilio, ces valeurs sont disponibles sur le tableau de bord du projet.

  > **Note**
  >
  > Vous pouvez utiliser un compte d'essai Twilio. Si vous utilisez un compte d'essai, vous devez ajouter des numéros de téléphone qui ne sont pas des destinataires Twilio à une liste de numéros de téléphone vérifiés. Pour de plus amples informations, veuillez consulterComment utiliser votre compte d'essai Twilio.

Étape 1 : Créer un secret Secrets Manager

Au cours de cette étape, vous utilisez la console AWS Secrets Manager pour créer un secret de type texte pour votre jeton d'authentification Twilio.

1. Connectez-vous à la console AWS Secrets Manager.

   > **Note**
   >
   > Pour plus d'informations sur ce processus, consultezÉtape 1 : Créez et stockez votre secret dansAWS Secrets Managerdans leAWS Secrets ManagerGuide de l'utilisateur.

2. Choisissez Store a new secret (Stocker un nouveau secret).
3. UnderChoisissez le type de secret, choisissezOther type of secret (Autre type de secret).
4. Sous Spécifiez les paires clé/valeur à être stockées pour ce secret, dans l'onglet Texte brut, saisissez votre jeton d'authentification Twilio. Supprimez tout le formatage JSON et entrez uniquement la valeur du jeton.
5. Keepaws/secretsmanagersélectionné pour la clé de chiffrement, puis choisissezSuivant.

   > **Note**
   >
   > Vous n'êtes pas facturé parAWS KMSsi vous utilisez la valeur par défautAWSclé gérée que Secrets Manager crée dans votre compte.

6. Pour Secret name, entrez **`greengrass-TwilioAuthToken`** et choisissez Next.

   > **Note**
   >
   > Par défaut, le rôle de service Greengrass permetAWS IoT Greengrasspour obtenir la valeur des secrets dont le nom commence pargreengrass. Pour plus d'informations, consultez les exigences liées aux secrets.

7. Ce didacticiel n'exige pas la rotation. Par conséquent, choisissez Désactiver la rotation automatique, puis choisissezSuivant.
8. Dans la page Révision, passez en revue vos paramètres, puis choisissez Stocker.

   Ensuite, vous créez une ressource de secret dans votre groupe Greengrass qui référence le secret.

Étape 2 : Ajouter une ressource de secret à un groupe Greengrass

Au cours de cette étape, vous ajoutez une ressource de secret au groupe Greengrass. Cette ressource est une référence au secret que vous avez créé à l'étape précédente.

1. DansAWS IoTVolde navigation de la console, sousGérer, développez.Appareils Greengrass, puis choisissezGroups (V1).
2. Choisissez le groupe auquel vous souhaitez ajouter la ressource de secret.
3. Sur la page de configuration de groupe, choisissez l'Ressources, puis faites défiler l'écran jusqu'à laSecretsSection. LeSecretsaffiche les ressources de secret qui appartiennent au groupe. Vous pouvez ajouter, modifier et supprimer des ressources de secret à partir de cette section.

   > **Note**
   >
   > La console vous permet également de créer une ressource secrète et secrète lorsque vous configurez un connecteur ou une fonction Lambda. Pour ce faire, vous pouvez utiliser leConfigurer les paramètres deou la page de la fonction LambdaRessources.

4. ChoisissezAdditionsousSecretsSection.
5. Dans la pageAjouter une ressource de secretpage, entrez**`MyTwilioAuthToken`**pour laNom de la ressource.
6. PourSecret, choisissezgreengrassTwilioAuthToken.
7. DansSélectionnez les étiquettes (facultatif), la section AWSCURRENT Le label de mise en scène représente la version la plus récente du secret. Cette étiquette est toujours incluse dans une ressource de secret.

   > **Note**
   >
   > Ce didacticiel nécessite la AWSCURRENT étiquette uniquement. Si vous le souhaitez, vous pouvez inclure les étiquettes qui sont requises par votre fonction ou votre connecteur Lambda.

8. Choisissez Add resource (Ajouter ressource).

Étape 3 : Ajouter un connecteur au groupe Greengrass

Au cours de cette étape, vous configurez les paramètres pour laConnecteur Twilio Notificationset ajoutez-le au groupe.

1. Sur la page de configuration de groupe, choisissez Connecteurs, puis choisissez Ajouter un connecteur.
2. Dans la pageAjouter un connecteur, choisissezNotifications Twilio.
3. Choisissez la version .
4. DansConfigurationSection:
   - PourRessource du jeton d'authentification Twilio, entrez la ressource que vous avez créée à l'étape précédente.

     > **Note**
     >
     > Lorsque vous entrez la ressource, leARN du secret du jeton d'authentification Twilioest remplie pour vous.

   - Pour Par défaut à partir de numéro de téléphone, saisissez votre numéro de téléphone activé Twilio.
   - Pour le SID du compte Twilio, saisissez le SID de votre compte Twilio.
5. Choisissez Add resource (Ajouter ressource).

Étape 4 : Créer un package de déploiement de fonction Lambda

Pour créer une fonction Lambda, vous devez d'abord créer une fonction Lambdapackage de déploiementqui contient le code de la fonction et les dépendances. Les fonctions Lambda de Greengrass nécessitentAWS IoT GreengrassKit SDK Corepour des tâches telles que la communication avec les messages MQTT dans l'environnement principal et l'accès aux secrets locaux. Ce didacticiel crée une fonction Python, de sorte que vous utilisiez la version Python du SDK dans le package de déploiement.

1. De laAWS IoT GreengrassKit SDK Corepage de téléchargement, téléchargez leAWS IoT GreengrassSDK Core pour Python sur votre ordinateur.
2. Décompressez le package téléchargé pour obtenir le kit SDK. Le kit SDK est représenté par le dossier `greengrasssdk`.

3. Enregistrez la fonction de code Python suivante dans un fichier local nommé `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. Compressez les éléments suivants dans un fichier nommé `temp_monitor_python.zip`. Lorsque vous créez le fichier zip, insérez uniquement le code et ses dépendances, pas le dossier dans lequel il se trouve.

- temp_monitor.py. Logique d'application.
- greengrasssdk. Bibliothèque requise pour toutes les fonctions Lambda Python Greengrass qui publient des messages MQTT.

Il s'agit du package de déploiement de votre fonction Lambda.

Maintenant, créez une fonction Lambda qui utilise le package de déploiement.

Étape 5 : Créez une fonction Lambda dans laAWS Lambdaconsole

Au cours de cette étape, vous utilisez laAWS Lambdapour créer une fonction Lambda et pour la configurer afin qu'elle utilise votre package de déploiement. Vous publiez ensuite une version de fonction et créez un alias.

1. Créez d'abord la fonction Lambda.

   a. Dans AWS Management Console, choisissez Services et ouvrez la console AWS Lambda.

   b. ChoisissezCréation de fonctionpuis choisissezCréer à partir de zéro.

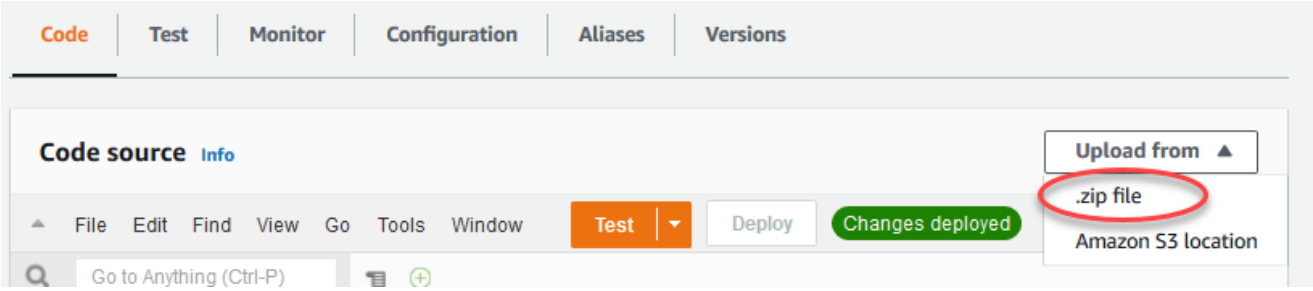
   c. Dans la section Informations de base, spécifiez les valeurs suivantes :

      - Sous Nom de la fonction, saisissez **`TempMonitor`**.
      - Pour Runtime, sélectionnez Python 3.7.
      - PourAutorisations, conservez le paramètre par défaut. Cela crée un rôle d'exécution qui accorde des autorisations Lambda de base. Ce rôle n'est pas utilisé parAWS IoT Greengrass.

   d. Dans le bas de la page, choisissez Create function.

2. Enregistrez ensuite le gestionnaire et téléchargez le package de déploiement de votre fonction Lambda.

   a. Dans la pageCodeonglet, sousSource de code, choisissezChargement à partir de. Dans le menu déroulant, choisissezfichier .zip.



b. ChoisissezCharger, puis choisissez votre`temp_monitor_python.zip`package de déploiement. Ensuite, choisissez Enregistrer.

c. Dans la pageCodepour la fonction, sousParamètres d'exécution, choisissezModifier, puis entrez les valeurs suivantes.

- Pour Runtime, sélectionnez Python 3.7.
- Pour Handler (Gestionnaire), entrez **`temp_monitor.function_handler`**.

d. Choisissez Save (Enregistrer).

> **Note**
>
> LeTestsur la pageAWS Lambdala console ne fonctionne pas avec cette fonction. LeAWS IoT GreengrassLe SDK principal ne contient pas de modules nécessaires pour exécuter vos fonctions Greengrass Lambda indépendamment dans leAWS Lambdaconsole Ces modules (par exemple,`greengrass_common`) sont fournis aux fonctions après leur déploiement dans votre cœur Greengrass.

3. Publiez maintenant la première version de votre fonction Lambda et créez unalias pour la version.

> **Note**
>
> Les groupes Greengrass peuvent référencer une fonction Lambda par alias (recommandé) ou par version. L'utilisation d'un alias facilite la gestion des mises à jour de code, car vous n'avez pas besoin de changer votre table d'abonnement ou votre définition de groupe lorsque le code de fonction est mis à jour. Au lieu de cela, il vous suffit de pointer l'alias vers la nouvelle version de la fonction.

a. Dans le menu Actions, sélectionnez Publier une nouvelle version.

b. Dans Description de la version, saisissez **`First version`**, puis choisissez Publish.

c. Dans la pageTempMonitor : 1page de configuration, depuis la pageActions, choisissez, choisissezCréer un alias.

d. Sur la page Create a new alias, utilisez les valeurs suivantes :

- Pour Name (Nom), saisissez **`GG_TempMonitor`**.
- Pour Version, choisissez 1.

> **Note**
>
> AWS IoT Greengrassne prend pas en charge les alias Lambda pour$LATESTversions.

e. Sélectionnez Create (Créer).

Vous pouvez désormais ajouter la fonction Lambda à votre groupe Greengrass.

Étape 6 : Ajouter une fonction Lambda au groupe Greengrass

Au cours de cette étape, vous ajoutez la fonction Lambda au groupe, puis configurez son cycle de vie et ses variables d'environnement. Pour plus d'informations, consultez the section called “Contrôle de l'exécution de la fonction Greengrass Lambda”.

1. Sur la page de configuration de groupe, choisissez l'Fonctions LambdaOnglet.
2. UnderFonctions Lambda, choisissezAddition.
3. Dans la pageAjouter une fonction Lambda, choisissezTempMonitorpour votre fonction Lambda
4. PourVersion de la fonction Lambda, choisissezAlias : GG_TempMonitor.
5. ChoisissezAjouter une fonction Lambda.

Étape 7 : Ajouter des abonnements au groupe Greengrass

Au cours de cette étape, vous allez ajouter un abonnement qui permet à la fonction Lambda d'envoyer des données d'entrée au connecteur. Le connecteur définit les rubriques MQTT auxquelles il est abonné, de sorte que cet abonnement utilise l'une des rubriques. Il s'agit de la même rubrique dans laquelle l'exemple de fonction effectue la publication.

Dans le cadre de ce didacticiel, vous pouvez également créer des abonnements qui autorisent la fonction à recevoir des relevés simulés de température de AWS IoT et autorisent AWS IoT à recevoir les informations d'état du connecteur.

1. Sur la page de configuration de groupe, choisissez l'Subscriptions, puis choisissezAjouter un abonnement.
2. Dans la pageCréer un abonnement, configurez la source et la cible, comme suit :
   a. PourType de source, choisissezFonction Lambda, puis choisissezTempMonitor.
   b. PourTarget type (Type de cible), choisissezConnecteur, puis choisissezNotifications Twilio.
3. PourFiltre de rubriques, choisissez**`twilio/txt`**.
4. Choisissez Create subscription (Créer un abonnement).
5. Répétez les étapes 1 à 4 pour créer un abonnement qui permet à AWS IoT de publier des messages dans la fonction.
   a. PourType de source, choisissezService, puis choisissezCloud IoT.
   b. PourSélectionnez une cible, choisissezFonction Lambda, puis choisissezTempMonitor.
   c. Pour Filtre de rubrique, tapez **`temperature/input`**.
6. Répétez les étapes 1 à 4 pour créer un abonnement qui permet au connecteur de publier des messages dans AWS IoT.
   a. PourType de source, choisissezConnecteur, puis choisissezNotifications Twilio.
   b. PourTarget type (Type de cible), choisissezService, puis choisissezCloud IoT.
   c. Pour Topic filter, **`twilio/message/status`** est entré pour vous. Il s'agit de la rubrique prédéfinie dans laquelle le connecteur effectue la publication.

Étape 8 : Déploiement du groupe Greengrass

Déployer le groupe sur l'appareil principal (noyau)

1. Vérifiez les éléments suivants :AWS IoT Greengrasscore est en cours d'exécution. Dans la fenêtre de terminal de votre Raspberry Pi, exécutez les commandes suivantes, si nécessaire.
   a. Pour vérifier si le démon est en cours d'exécution :

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

Si la sortie contient une entrée `root` pour `/greengrass/ggc/packages/ggc-version/bin/daemon`, le démon est en cours d'exécution.

> **Note**
>
> La version du chemin d'accès dépend de la version du logiciel AWS IoT Greengrass Core installée sur votre appareil principal.

b. Pour démarrer le démon :

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

2. Sur la page de configuration du groupe, choisissezDéploiement.
3. a. DansFonctions Lambda, sous l'ongletFonctions Lambdasection, sélectionnezDétecteur IPet choisissezModifier.

   b. DansModifier les paramètres du détecteur IP, sélectionnezDétecter et remplacer automatiquement les points de terminaison des courtiers MQTT.

   c. Choisissez Save (Enregistrer).

Les appareils peuvent ainsi acquérir automatiquement des informations de connectivité pour le noyau, telles que l'adresse IP, le DNS et le numéro de port. La détection automatique est recommandée, mais AWS IoT Greengrass prend également en charge les points de terminaison spécifiés manuellement. Vous êtes uniquement invité à indiquer la méthode de découverte lors du déploiement initial du groupe.

> **Note**
>
> Si vous y êtes invité, autorisez la création deRôle de service Greengrasset associez-le à votreCompte AWSdans les actuelsRégion AWS. Ce rôle permet àAWS IoT Greengrasspour accéder à vos ressources dansAWSServices .

La page Déploiements indique l'horodatage, l'ID de version et l'état du déploiement. Une fois terminé, le statut affiché pour le déploiement doit afficher l'état du déploiement estTerminé.

Pour bénéficier d'une aide à la résolution des problèmes, consultez Résolution des problèmes.

> **Note**
>
> Un groupe Greengrass peut contenir une seule version du connecteur à la fois. Pour de plus amples informations sur la mise à niveau d'une version de connecteur, veuillez consulter the section called “Mise à niveau des versions du connecteur”.

Tester la solution

1. Dans la pageAWS IoTpage d'accueil de la console, choisissezTest.
2. PourS'abonner à la rubrique, utilisez les valeurs suivantes, puis choisissezS'abonner. Le connecteur Twilio Notifications publie des informations de statut dans cette rubrique.

Propriété	Valeur
rubrique abonnement	twilio/message/status
Affichage de la charge utile MQTT	Affichage des charges utiles sous forme de chaînes

3. PourPublier dans la rubrique, utilisez les valeurs suivantes, puis choisissezPublierpour appeler la fonction.

Propriété	Valeur
Sujet	temperature/input
Message	Remplacez*nom-du destinataire* avec un nom et*recipient-phone-number* avec le numéro de téléphone du destinataire du SMS. Exemple : `+12345000000`

Propriété	Valeur

	<pre>{
	   "to_name": " *recipient-name* ",
	   "to_number": " *recipient-phone-number* ",
	   "temperature": 31
	}</pre>
	Si vous utilisez un compte d'essai, vous devez ajouter des numéros de téléphone qui ne sont pas des destinataires Twilio à une liste de numéros de téléphone vérifiés. Pour de plus amples informations, veuillez consulterVerify votre numéro de téléphone.

En cas de réussite, le destinataire reçoit le SMS et la console affiche le statut `success` depuis les données de sortie.

Maintenant, passez la `temperature` dans le message d'entrée à **29** et publiez. Étant donné que cette valeur est inférieure à 30, le TempMonitor ne déclenche pas de message Twilio.

Consulter aussi

- Intégrer à des services et protocoles à l'aide de connecteurs
- the section called “AWS- connecteurs Greengrass fournis”

Démarrage avec les connecteurs Greengrass (CLI)

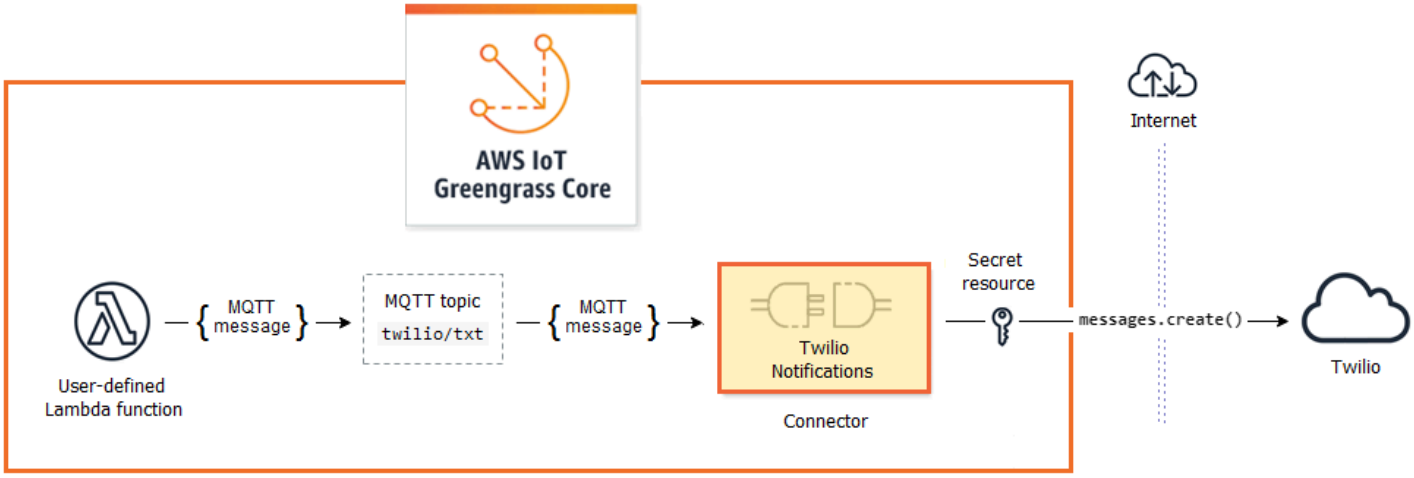
Cette fonction est disponible pourAWS IoT GreengrassCore v1.7 et versions ultérieures.

Ce didacticiel explique comment utiliser la AWS CLI pour travailler avec les connecteurs.

Utilisez des connecteurs pour accélérer votre cycle de développement. Les connecteurs sont des modules réutilisables prédéfinis qui contribuent à interagir avec les services, les protocoles et les ressources. Ils peuvent vous aider à déployer plus rapidement une logique métier pour les appareils

Greengrass. Pour plus d'informations, consultez Intégrer à des services et protocoles à l'aide de connecteurs.

Dans ce didacticiel, vous configurez et déployez leNotifications Twilioconnecteur. Le connecteur reçoit des informations du message Twilio en tant que données d'entrée, puis déclenche un SMS Twilio. Le flux de données est illustré dans le diagramme suivant.



Une fois que vous avez configuré le connecteur, vous créez une fonction Lambda et un abonnement.

- La fonction évalue les données simulées à partir d'un capteur de température. Elle publie les informations du message Twilio sous certaines conditions dans une rubrique MQTT. Il s'agit de la rubrique à laquelle le connecteur est abonné.
- L'abonnement autorise la fonction à publier dans la rubrique et le connecteur à recevoir des données provenant de la rubrique.

Le connecteur Twilio Notifications nécessite un jeton d'authentification Twilio pour interagir avec l'API Twilio. Le jeton est un secret de type texte dans AWS Secrets Manager et référencé à partir d'une ressource de groupe. Cela permet à AWS IoT Greengrass de créer une copie locale du secret sur le noyau Greengrass, où il est chiffré et mis à la disposition du connecteur. Pour plus d'informations, consultez Déployer des secrets sur Core.

Le didacticiel contient les étapes détaillées suivantes :

1. Créer un secret Secrets Manager
2. Créer une version et une définition de ressource

3. Créer une version et une définition de connecteur
4. Créer un package de déploiement de fonction Lambda
5. Création d'une fonction Lambda
6. Créer une version et une définition de fonction
7. Créer une version et une définition d'abonnement
8. Créer une version de groupe
9. Créer un déploiement
10. the section called “Tester la solution”

Le didacticiel devrait prendre environ 30 minutes.

Utilisation de l'API AWS IoT Greengrass

Il est utile de comprendre les modèles suivants lorsque vous utilisez des groupes Greengrass et des composants de groupe (par exemple, les fonctions, les connecteurs et les ressources du groupe).

- En haut de la hiérarchie, un composant possède un objet de définition qui est un conteneur d'objets de version. En revanche, une version est un conteneur pour les connecteurs, les fonctions ou d'autres types de composants.
- Lorsque vous déployez vers le noyau Greengrass, vous déployez une version de groupe spécifique. Une version de groupe peut contenir une version de chaque type de composant. Un noyau est obligatoire, mais les autres sont inclus en fonction des besoins.
- Les versions sont immuables. Ainsi, vous devez créer de nouvelles versions lorsque vous souhaitez apporter des modifications.

> **Tip**
>
> Si vous recevez une erreur lorsque vous exécutez une commande AWS CLI, ajoutez le paramètre `--debug`, puis relancez la commande pour obtenir plus d'informations sur l'erreur.

L'API AWS IoT Greengrass vous permet de créer plusieurs définitions pour un type de composant. Par exemple, vous pouvez créer un objet `FunctionDefinition` à chaque fois que vous créez une `FunctionDefinitionVersion` ou vous pouvez ajouter de nouvelles versions à une définition existante. Cette flexibilité vous permet de personnaliser votre système de gestion de version.

Prérequis

Pour suivre ce didacticiel, vous devez disposer des éléments suivants :

- Un groupe Greengrass et un appareil principal (noyau) Greengrass (version 1.9.3 ou ultérieure). Pour savoir comment créer un groupe et un service principal Greengrass, consultez Commencer avec AWS IoT Greengrass. Le didacticiel Mise en route comprend également les étapes d'installation du logiciel AWS IoT Greengrass Core.
- Python 3.7 installé sur l'appareil principal (noyau) AWS IoT Greengrass.
- AWS IoT Greengrassdoit être configuré pour prendre en charge les secrets locaux, comme décrit dans leExigences relatives aux secrets.

  > **Note**
  >
  > Cette exigence inclut l'autorisation d'accès à vos Secrets Manager. Si vous utilisez le rôle de service Greengrass par défaut, Greengrass est autorisé à obtenir les valeurs des secrets dont les noms commencent pargreengrass.

- Un SID de compte Twilio, un jeton d'authentification et un numéro de téléphone Twilio. Une fois que vous avez créé un projet Twilio, ces valeurs sont disponibles sur le tableau de bord du projet.

  > **Note**
  >
  > Vous pouvez utiliser un compte d'essai Twilio. Si vous utilisez un compte d'essai, vous devez ajouter des numéros de téléphone qui ne sont pas des destinataires Twilio à une liste de numéros de téléphone vérifiés. Pour de plus amples informations, veuillez consulterComment travailler avec votre compte d'essai Twilio.

- AWS CLI installée et configurée sur votre machine. Pour de plus amples informations, veuillez consulterInstallation deAWS Command Line InterfaceetConfiguration deAWS CLIdans leAWS Command Line InterfaceGuide de l'utilisateur.

Les exemples de ce didacticiel sont écrits pour Linux et d'autres systèmes Unix. Si vous utilisez Windows, consultezSpécification des valeurs des paramètres pour l'AWS Command Line Interfacepour en savoir plus sur les différences de syntaxe.

Si la commande contient une chaîne JSON, le didacticiel fournit un exemple qui possède le fichier JSON sur une seule ligne. Sur certains systèmes, il peut être plus facile de modifier et exécuter des commandes à l'aide de ce format.

Étape 1 : Créer un secret Secrets Manager

Au cours de cette étape, vous utilisez l'API AWS Secrets Manager pour créer un secret pour votre jeton AUTH Twilio.

1. D'abord, créez le secret.

   - Remplacez*twilio-auth-token*avec votre jeton d'authentification Twilio.

   ```
   aws secretsmanager create-secret --name greengrass-TwilioAuthToken --secret-
   string twilio-auth-token
   ```

   > **Note**
   >
   > Par défaut, le rôle de service Greengrass permetAWS IoT Greengrasspour obtenir la valeur des secrets dont le nom commence àgreengrass. Pour plus d'informations, consultez les exigences liées aux secrets.

2. Copiez l'ARN du secret à partir de la sortie. Vous utilisez cela pour créer la ressource du secret et pour configurer le connecteur Twilio Notifications.

Étape 2 : Créer une version et une définition de ressource

Au cours de cette étape, vous utilisez leAWS IoT GreengrassAPI pour créer une ressource de Secrets Manager.

1. Créez une définition de ressource qui inclut une version initiale.

   - Remplacez *secret-arn* avec l'ARN de la clé du secret que vous avez copiée à l'étape précédente.

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. Copiez l'`LatestVersionArn` de la définition de ressource à partir de la sortie. Vous utilisez cette valeur pour ajouter la version de définition de ressource à la version de groupe que vous déployez pour le noyau.

Étape 3 : Créer une version et une définition de connecteur

Dans cette étape, vous configurez les paramètres pour le connecteur Twilio Notifications.

1. Créez une définition de connecteur avec une version initiale.

   - Remplacez *`account-sid`* par votre SID de compte Twilio.
   - Remplacez*`ARN secret`*avec leARNde votre secret Secrets Manager. Le connecteur utilise ceci pour obtenir la valeur du secret local.

- Remplacez le *phone-number* avec votre numéro de téléphone Twilio. Twilio utilise celui-ci pour initier le SMS. Cela peut être remplacé dans la charge utile du message d'entrée. Utilisez le format suivant : +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` est l'ID que vous avez utilisé à l'étape précédente pour créer les ressources de secret.

2. Copiez l'`LatestVersionArn` de la définition de connecteur à partir de la sortie. Vous utilisez cette valeur pour ajouter la version de définition de connecteur à la version de groupe que vous déployez pour le noyau.

Étape 4 : Créer un package de déploiement de fonction Lambda

Pour créer une fonction Lambda, vous devez d'abord créer une fonction Lambdapackage de déploiementqui contient le code de fonction et les dépendances. Les fonctions Lambda de Greengrass nécessitentAWS IoT GreengrassKit SDK Corepour des tâches telles que la communication avec les messages MQTT dans l'environnement principal et l'accès aux secrets locaux. Ce didacticiel crée une fonction Python, de sorte que vous utilisiez la version Python du SDK dans le package de déploiement.

1. À partir de laAWS IoT GreengrassKit SDK Corepage de téléchargement, téléchargez leAWS IoT GreengrassSDK Core pour Python sur votre ordinateur.
2. Décompressez le package téléchargé pour obtenir le kit SDK. Le kit SDK est représenté par le dossier `greengrasssdk`.
3. Enregistrez la fonction de code Python suivante dans un fichier local nommé `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. Compressez les éléments suivants dans un fichier nommé `temp_monitor_python.zip`. Lorsque vous créez le fichier zip, insérez uniquement le code et ses dépendances, pas le dossier dans lequel il se trouve.

   - temp_monitor.py. Logique d'application.
   - greengrasssdk. Bibliothèque requise pour toutes les fonctions Lambda Python Greengrass qui publient des messages MQTT.

   Il s'agit du package de déploiement de votre fonction Lambda.

Étape 5 : Création d'une fonction Lambda

Maintenant, créez une fonction Lambda qui utilise le package de déploiement.

1. Créez un rôle IAM pour vous permettre de transmettre l'ARN du rôle lorsque vous créez la fonction.

   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 Greengrassn'utilise pas ce rôle car les autorisations pour vos fonctions Greengrass Lambda sont spécifiées dans le rôle de groupe Greengrass. Dans le cadre de ce didacticiel, vous créez un rôle vide.

2. Copiez la `Arn` à partir de la sortie.
3. Utilisation de l'AWS LambdaAPI pour créer le TempMonitor . La commande suivante suppose que le fichier ZIP se trouve dans le répertoire actuel.

   - Remplacez *`role-arn`* par l'`Arn` que vous avez copié.

```
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. Publiez une version de la fonction.

```
aws lambda publish-version --function-name TempMonitor --description 'First
 version'
```

5. Créez un alias pour la version publiée.

   Les groupes Greengrass peuvent référencer une fonction Lambda par alias (recommandé) ou par version. L'utilisation d'un alias facilite la gestion des mises à jour de code, car vous n'avez pas besoin de changer votre table d'abonnement ou votre définition de groupe lorsque le code de fonction est mis à jour. Au lieu de cela, il vous suffit de pointer l'alias vers la nouvelle version de fonction.

   > **Note**
   >
   > AWS IoT Greengrassne prend pas en charge les alias Lambda pour$LATESTversions.

   ```
   aws lambda create-alias --function-name TempMonitor --name GG_TempMonitor --
   function-version 1
   ```

6. Copiez la `AliasArn` à partir de la sortie. Vous utilisez cette valeur lorsque vous configurez la fonction pour AWS IoT Greengrass et lorsque vous créez un abonnement.

Maintenant, vous êtes prêt à configurer la fonction pour AWS IoT Greengrass.

Étape 6 : Créer une version et une définition de fonction

Pour utiliser une fonction Lambda sur unAWS IoT Greengrasscore, vous créez une version de définition de fonction qui fait référence à la fonction Lambda en alias et définit la configuration au niveau du groupe. Pour plus d'informations, consultez the section called “Contrôle de l'exécution de la fonction Greengrass Lambda”.

1. Créez une définition de fonction qui inclut une version initiale.

   - Remplacez *`alias-arn`* par l'`AliasArn` que vous avez copié lorsque vous avez créé 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. Copiez la `LatestVersionArn` à partir de la sortie. Vous utilisez cette valeur pour ajouter la version de définition de fonction à la version de groupe que vous déployez pour le noyau.
3. Copiez la `Id` à partir de la sortie. Vous utiliserez cette valeur ultérieurement lorsque vous mettrez à jour la fonction.

Étape 7 : Créer une version et une définition d'abonnement

Au cours de cette étape, vous allez ajouter un abonnement qui permet à la fonction Lambda d'envoyer des données d'entrée au connecteur. Le connecteur définit les rubriques MQTT auxquelles il est abonné, de sorte que cet abonnement utilise l'une des rubriques. Il s'agit de la même rubrique dans laquelle l'exemple de fonction effectue la publication.

Dans le cadre de ce didacticiel, vous pouvez également créer des abonnements qui autorisent la fonction à recevoir des relevés simulés de température de AWS IoT et autorisent AWS IoT à recevoir les informations d'état du connecteur.

1. Création d'une définition d'abonnement qui contient une version initiale qui inclut les abonnements.

- Remplacez *alias-arn* par l'`AliasArn` que vous avez copié lorsque vous avez créé l'alias de cette fonction. Utilisez cet ARN pour les deux abonnements qui l'utilisent.

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. Copiez la `LatestVersionArn` à partir de la sortie. Vous utilisez cette valeur pour ajouter la version de définition d'abonnement à la version de groupe que vous déployez pour le noyau.

Étape 8 : Créer une version de groupe

Maintenant, vous êtes prêt à créer une version de groupe qui contient tous les éléments que vous souhaitez déployer. Pour ce faire, vous devez créer une version de groupe qui fait référence à la version cible de chaque type de composant.

Tout d'abord, obtenez l'ID de groupe et l'ARN de la version de définition du noyau. Ces valeurs sont requises pour créer la version de groupe.

1. Obtenir l'ID du groupe et la dernière version du groupe :

   a. Obtenez les ID du groupe et de la version de groupe Greengrass cible. Cette procédure suppose qu'il s'agit du dernier groupe et de la dernière version du groupe. La requête suivante renvoie le groupe créé le plus récemment.

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

   Vous pouvez également procéder à une interrogation par nom. Les noms de groupe ne devant pas nécessairement être uniques, plusieurs groupes peuvent être renvoyés.

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

   > **Note**
   >
   > Vous trouverez également ces valeurs dans leAWS IoTconsole L'ID du groupe s'affiche sur la page Paramètres du groupe. Les ID de version de groupe sont affichés dans leDéploiementsonglet.

   b. Copiez l'`Id` du groupe cible à partir de la sortie. Vous utilisez cela pour obtenir la version de définition du noyau et lorsque vous déployez le groupe.

c. Copiez l'élément `LatestVersion` à partir de la sortie (ID de la dernière version ajoutée au groupe). Vous utilisez cela pour obtenir la version de la définition du noyau.

2. Obtenir l'ARN de la version de la définition du noyau :

   a. Obtenez la version de groupe. Pour cette étape, nous supposons que la dernière version de groupe inclut une version de définition du noyau.

      - Remplacez *group-id* par l'`Id` que vous avez copié pour le groupe.
      - Remplacez*group-version-id*avec le`LatestVersion`que vous avez copié pour le groupe.

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

   b. Copiez la `CoreDefinitionVersionArn` à partir de la sortie.

3. Créer une version de groupe.

   - Remplacez *group-id* par l'`Id` que vous avez copié pour le groupe.
   - Remplacez*core-definition-version-arn*avec le`CoreDefinitionVersionArn`que vous avez copié pour la version de la définition du noyau.
   - Remplacez*resource-definition-version-arn*avec le`LatestVersionArn`que vous avez copié pour la définition de ressource.
   - Remplacez*connector-definition-version-arn*avec le`LatestVersionArn`que vous avez copié pour la définition du connecteur.
   - Remplacez*function-definition-version-arn*avec le`LatestVersionArn`que vous avez copié pour la définition de fonction.
   - Remplacez*subscription-definition-version-arn*avec le`LatestVersionArn`que vous avez copié pour la définition d'abonnement.

```
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. Copiez la valeur `Version` à partir de la sortie. Il s'agit de l'ID de la version de groupe. Vous utilisez cette valeur pour déployer la version de groupe.

Étape 9 : Créer un déploiement

Déployer le groupe sur l'appareil principal (noyau)

1. Sur votre appareil principal, assurez-vous que le démon AWS IoT Greengrass est en cours d'exécution.

   a. Pour vérifier si le démon est en cours d'exécution :

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

   Si la sortie contient une entrée `root` pour `/greengrass/ggc/packages/1.11.6/bin/daemon`, le démon est en cours d'exécution.

   b. Pour démarrer le démon :

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

2. Créer un déploiement.

   - Remplacez *group-id* par l'`Id` que vous avez copié pour le groupe.
   - Remplacez*group-version-id*avec le`Version`que vous avez copié pour la nouvelle version de groupe.

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

3. Copiez la `DeploymentId` à partir de la sortie.

4. Obtenir le statut du déploiement.

   - Remplacez *group-id* par l'`Id` que vous avez copié pour le groupe.

- Remplacez *deployment-id* par l'`DeploymentId` que vous avez copié pour le déploiement.

```
aws greengrass get-deployment-status \
--group-id group-id \
--deployment-id deployment-id
```

Si le statut est`Success`, le déploiement a réussi. Pour bénéficier d'une aide à la résolution des problèmes, consultez Résolution des problèmes.

Tester la solution

1. Dans la pageAWS IoTpage d'accueil de la console, choisissezTest.
2. PourS'abonner à la rubrique, utilisez les valeurs suivantes, puis choisissezS'abonner. Le connecteur Twilio Notifications Twilio Notifications Twilio Notifications Twilio

Propriété	Valeur
rubrique abonnement	twilio/message/status
Affichage de la charge utile MQTT	Affichage des charges utiles sous forme de chaînes

3. PourPublier dans la rubrique, utilisez les valeurs suivantes, puis choisissezPublierpour appeler la fonction.

Propriété	Valeur
Sujet	temperature/input

Message	Remplacez*recipient-name* avec un nom et*recipient-phone-number* avec le numéro de téléphone du destinataire du SMS. Exemple : `+12345000000`
	`{`
	`"to_name": " recipient-name ",`

Propriété	Valeur

	<pre>"to_number": " recipient-phone-number ",
	"temperature": 31
	}</pre>
	Si vous utilisez un compte d'essai, vous devez ajouter des numéros de téléphone qui ne sont pas des destinataires Twilio à une liste de numéros de téléphone vérifiés. Pour de plus amples informations, veuillez consulterVerify your Personal Phone Number.

En cas de réussite, le destinataire reçoit le SMS et la console affiche le statut `success` depuis les données de sortie.

Maintenant, passez la `temperature` dans le message d'entrée à **29** et publiez. Étant donné que cette valeur est inférieure à 30, le TempMonitor ne déclenche pas de message Twilio.

Consulter aussi

- Intégrer à des services et protocoles à l'aide de connecteurs
- the section called “AWS- connecteurs Greengrass fournis”
- the section called “Démarrer avec les connecteurs (console)”
- AWS Secrets Managercommandesdans leAWS CLIRéférence des commandes
- AWS Identity and Access ManagementCommandes (IAM)dans leAWS CLIRéférence des commandes
- AWS Lambdacommandesdans leAWS CLIRéférence des commandes
- AWS IoT Greengrasscommandesdans leAWS CLIRéférence des commandes

API RESTful de découverte de Greengrass

Tous les appareils clients qui communiquent avec unAWS IoT Greengrass noyau doivent appartenir à un groupe Greengrass. Chaque groupe doit avoir un noyau Greengrass. L'API Discovery permet aux appareils de récupérer les informations requises pour se connecter à un noyau Greengrass appartenant au même groupe Greengrass que l'appareil client. Lorsqu'un appareil client est connecté pour la première fois, il peut se connecter auAWS IoT Greengrass service et utiliser l'API Discovery pour rechercher :

- Le groupe auquel il appartient Un appareil client peut appartenir à 10 groupes au maximum.
- L'adresse IP et le port du noyau Greengrass dans le groupe.
- Le certificat de l'autorité de certification du groupe, qui peut être utilisé pour authentifier l'appareil principal Greengrass (noyau).

> **Note**
>
> Les appareils clients peuvent également utiliser lesAWS IoT kits de la Greengrass. Pour plus d'informations, veuillez consulter SDK pour les appareils AWS IoT.

Pour utiliser cette API, envoyez des demandes HTTP au point de terminaison de l'API Discovery. Par exemple :

```
https://greengrass-ats.iot.region.amazonaws.com:port/greengrass/discover/thing/thing-name
```

Pour obtenir la liste des régions et des points de terminaison Amazon Web Services pris en charge pour l'APIAWS IoT Greengrass Discovery, consultez les AWS IoT Greengrasspoints de terminaison et les quotas dans le Références générales AWS. Cette API est réservée aux plans de données. Les points de terminaison pour la gestion de groupe et les opérations AWS IoT Core sont différents des points de terminaison de l'API Discovery.

Requête

La demande contient les en-têtes HTTP standard et est envoyée au point de terminaison de découverte Greengrass, comme illustré dans les exemples suivants.

Le numéro de port varie selon que le noyau est configuré pour envoyer le trafic HTTPS via le port 8443 ou le port 443. Pour plus d'informations, veuillez consulter the section called “Connexion au port 443 ou via un proxy réseau”.

Port 8443

```
HTTP GET https://greengrass-ats.iot.region.amazonaws.com:8443/greengrass/discover/
thing/thing-name
```

Port 443

```
HTTP GET https://greengrass-ats.iot.region.amazonaws.com:443/greengrass/discover/
thing/thing-name
```

Les clients qui se connectent sur le port 443 doivent implémenter l'extension TLS ALPN (Application Layer Protocol Negotiation) et transmettre`x-amzn-http-ca` le`ProtocolNameList`.`ProtocolName` Pour plus d'informations, consultez la section Protocoles du Guide duAWS IoT développeur.

> **Note**
>
> Ces exemples utilisent le point de terminaison Amazon Trust Services (ATS), qui est utilisé avec les certificats CA racine ATS (recommandé). Les points de terminaison doivent correspondre au type de certificat CA racine. Pour plus d'informations, veuillez consulter the section called “Les points de terminaison du service doivent correspondre au type de certificat”.

Réponse

En cas de réussite, la réponse comprend les en-têtes HTTP standard, plus le code et le corps suivants :

```
HTTP 200
BODY: response document
```

Pour plus d'informations, veuillez consulter Exemple de documents de réponse à une découverte.

Acvery Service Service Service

La récupération des informations de connectivité nécessite une stratégie qui permet au mandataire de réaliser l'action `greengrass:Discover`. L'authentification mutuelle TLS avec un certificat client est la seule forme d'authentification acceptée. Voici un exemple de stratégie qui autorise un mandataire à effectuer cette action :

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Action": "greengrass:Discover",
        "Resource": ["arn:aws:iot:us-west-2:123456789012:thing/MyThingName"]
     }]
}
```

Exemple de documents de réponse à une découverte

Le document suivant présente la réponse pour un appareil client membre d'un groupe doté d'un certificat Greengrass Core, d'un point de terminaison et d'un certificat CA de groupe :

```
{
  "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-----"
      ]
```

```
    }
  ]
}
```

Le document suivant présente la réponse pour un appareil client membre de deux groupes avec un cœur Greengrass, plusieurs points de terminaison et plusieurs certificats CA de groupe :

```
{
  "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 groupe Greengrass doit définir exactement un noyau Greengrass. Toute réponse du service AWS IoT Greengrass qui contient une liste de noyaux Greengrass ne contient qu'un seul noyau Greengrass.

Si cURL est installé, vous pouvez tester la demande de découverte. Par exemple :

```
$ curl --cert 1a23bc4d56.cert.pem --key 1a23bc4d56.private.key https://greengrass-
ats.iot.us-west-2.amazonaws.com:8443/greengrass/discover/thing/MyDevice
{"GGGroups":[{"GGGroupId":"1234a5b6-78cd-901e-2fgh-3i45j6k1789","Cores":
[{"thingArn":"arn:aws:iot:us-west-2:1234567
89012:thing/MyFirstGroup_Core","Connectivity":
[{"Id":"AUTOIP_192.168.1.4_1","HostAddress":"192.168.1.5","PortNumber
":8883,"Metadata":""}]}],"CAs":["-----BEGIN CERTIFICATE-----\ncert-contents\n-----END
 CERTIFICATE-----\n"]}]}
```

Sécurité dans AWS IoT Greengrass

Chez AWS, la sécurité dans le cloud est notre priorité numéro 1. En tant que client AWS, vous bénéficiez d'un centre de données et d'une architecture réseau conçus pour répondre aux exigences des organisations les plus pointilleuses en termes de sécurité.

La sécurité est une responsabilité partagée entre AWS et vous. Le modèle de responsabilité partagée décrit ceci comme la sécurité du cloud et la sécurité dans le cloud :

- Sécurité du cloud : AWS est responsable de la protection de l'infrastructure qui exécute des services AWS dans le cloud AWS Cloud. AWS vous fournit également les services que vous pouvez utiliser en toute sécurité. Des auditeurs tiers testent et vérifient régulièrement l'efficacité de notre sécurité dans le cadre des AWS programmes de conformité. Pour en savoir plus sur les programmes de conformité qui s'appliquent à AWS IoT Greengrass, consultez Services AWS concernés par le programme de conformité.
- Sécurité dans le cloud : votre responsabilité est déterminée par le AWSservice que vous utilisez. Vous êtes également responsable d'autres facteurs, y compris la sensibilité de vos données, les exigences de votre entreprise, et la législation et la réglementation applicables.

Lorsque vous utilisez AWS IoT Greengrass, vous êtes également responsable de la sécurisation de vos appareils, de la connexion au réseau local et des clés privées.

Cette documentation vous aide à comprendre comment appliquer le modèle de responsabilité partagée lors de l'utilisation de AWS IoT Greengrass. Les rubriques suivantes expliquent comment configurer AWS IoT Greengrass pour répondre à vos objectifs de sécurité et de conformité. Vous apprendrez également à utiliser d'autres services AWS pour surveiller et sécuriser vos ressources AWS IoT Greengrass.

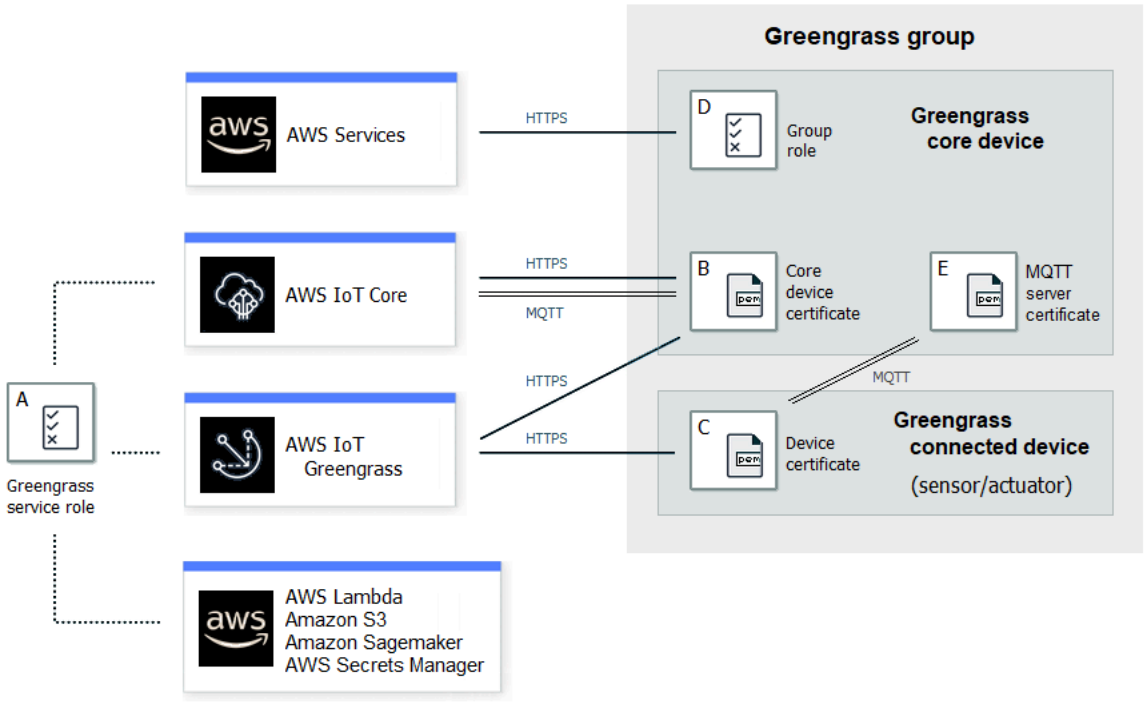
Rubriques

- Sécurité de l'infrastructure dans AWS IoT Greengrass
- Configuration et analyse des vulnérabilités dans AWS IoT Greengrass
- AWS IoT Greengrass et interface des points de terminaison VPC (AWS PrivateLink)
- Bonnes pratiques de sécurité pour AWS IoT Greengrass

Vue d'ensemble de la AWS IoT Greengrass sécurité

AWS IoT Greengrass utilise des certificats X.509, des AWS IoT politiques, des politiques et des rôles IAM pour sécuriser les applications qui s'exécutent sur les appareils de votre environnement Greengrass local.

Le schéma suivant montre les composants du modèle AWS IoT Greengrass de sécurité :



A - Rôle de service Greengrass

Rôle IAM créé par le client et assumé AWS IoT Greengrass lors de l'accès à vos AWS ressources depuis AWS IoT Core AWS Lambda, et à d'autres services. AWS Pour de plus amples informations, veuillez consulter the section called “Rôle de service Greengrass”.

B - Certificat de l'appareil noyau

Un certificat X.509 utilisé pour authentifier un noyau Greengrass avec et. AWS IoT Core AWS IoT Greengrass Pour de plus amples informations, veuillez consulter the section called “Authentification et autorisation d'appareil”.

C - Certificat d'appareil

Certificat X.509 utilisé pour authentifier un appareil client, également appelé appareil connecté, avec AWS IoT Core et. AWS IoT Greengrass Pour de plus amples informations, veuillez consulter the section called “Authentification et autorisation d'appareil”.

D - Rôle de groupe

Rôle IAM créé par le client AWS IoT Greengrass lorsqu'il appelle des AWS services à partir d'un noyau Greengrass.

Vous utilisez ce rôle pour spécifier les autorisations d'accès dont vos fonctions et connecteurs Lambda définis par l'utilisateur ont besoin pour AWS accéder à des services tels que DynamoDB. Vous l'utilisez également pour autoriser l'exportation AWS IoT Greengrass des flux du gestionnaire de flux vers AWS des services et l'écriture dans CloudWatch Logs. Pour de plus amples informations, veuillez consulter the section called “Rôle de groupe Greengrass”.

> **Note**
>
> AWS IoT Greengrass n'utilise pas le rôle d'exécution Lambda spécifié dans AWS Lambda pour la version cloud d'une fonction Lambda.

E - Certificat de serveur MQTT

Certificat utilisé pour l'authentification mutuelle TLS (Transport Layer Security) entre un appareil principal de Greengrass et des appareils clients du groupe Greengrass. Le certificat est signé par le certificat CA du groupe, qui est stocké dans le AWS Cloud.

Flux de connexion des appareils

Cette section décrit comment les appareils clients se connectent au AWS IoT Greengrass service et aux appareils principaux de Greengrass. Les appareils clients sont AWS IoT Core des appareils enregistrés qui appartiennent au même groupe Greengrass que le périphérique principal.

- Un appareil Greengrass Core utilise son certificat d'appareil, sa clé privée et le certificat CA AWS IoT Core racine pour se connecter au AWS IoT Greengrass service. Sur l'appareil principal, l'objet `crypto` du fichier de configuration spécifie le chemin d'accès au fichier pour ces éléments.
- L'appareil noyau Greengrass télécharge les informations d'adhésion au groupe à partir du service AWS IoT Greengrass .
- Lorsqu'un déploiement est effectué sur l'appareil noyau Greengrass, le gestionnaire de certificats de l'appareil (DCM) s'occupe de la gestion des certificats de serveur local pour l'appareil noyau Greengrass.
- Un appareil client se connecte au AWS IoT Greengrass service à l'aide de son certificat de périphérique, de sa clé privée et du certificat de l'autorité de certification AWS IoT Core racine. Une fois la connexion établie, l'appareil client utilise le service Greengrass Discovery pour trouver l'adresse IP de son appareil principal Greengrass. L'appareil client télécharge également le certificat CA du groupe, qui est utilisé pour l'authentification mutuelle TLS avec le périphérique principal Greengrass.
- Un appareil client tente de se connecter au périphérique principal de Greengrass en transmettant son certificat d'appareil et son identifiant client. Si l'ID client correspond au nom de l'appareil client et que le certificat est valide (il fait partie du groupe Greengrass), la connexion est établie. Dans le cas contraire, la connexion est arrêtée.

La AWS IoT politique relative aux appareils clients doit `greengrass:Discover` autoriser les appareils clients à découvrir les informations de connectivité relatives au cœur. Pour de plus amples informations sur cette instruction de stratégie, veuillez consulter the section called “Acvery Service Service Service”.

Configuration de AWS IoT Greengrass la sécurité

Pour configurer la sécurité de votre application Greengrass :

1. Créez n'importe AWS IoT Core quel objet pour votre appareil Greengrass principal.
2. Générez une paire de clés et un certificat d'appareil pour votre appareil noyau Greengrass.
3. Créez et attachez une stratégie AWS IoT au certificat d'appareil. Le certificat et la politique permettent à l'appareil principal de Greengrass d'accéder aux services AWS IoT Core et AWS IoT Greengrass d'accéder à ces services. Pour de plus amples informations, veuillez consulter Stratégie AWS IoT minimale pour l'appareil principal (noyau).

> **Note**
>
> L'utilisation de variables de politique d'objets (`iot:Connection.Thing.*`) dans la AWS IoT politique d'un appareil principal n'est pas prise en charge. Le noyau utilise le même certificat d'appareil pour établir plusieurs connexions AWS IoT Core , mais l'ID client d'une connexion peut ne pas correspondre exactement au nom de l'objet principal.

4. Créez un rôle de service Greengrass. Ce rôle IAM autorise l'accès AWS IoT Greengrass aux ressources d'autres AWS services en votre nom. Cela permet d' AWS IoT Greengrass effectuer des tâches essentielles, telles que la récupération des AWS Lambda fonctions et la gestion des ombres de l'appareil.

   Vous pouvez utiliser le même rôle de service entre Région AWS s, mais il doit être associé Compte AWS au vôtre partout Région AWS où vous l'utilisez AWS IoT Greengrass.

5. (Facultatif) Créez un rôle de groupe Greengrass. Ce rôle IAM autorise les fonctions et connecteurs Lambda exécutés sur un noyau Greengrass à appeler des services. AWS Par exemple, le connecteur Kinesis Firehose nécessite une autorisation pour écrire des enregistrements dans un flux de diffusion Amazon Data Firehose.

   Vous ne pouvez attacher qu'un seul rôle à un groupe Greengrass.

6. Créez un AWS IoT Core objet pour chaque appareil qui se connecte à votre Greengrass Core.

> **Note**
>
> Vous pouvez également utiliser des AWS IoT Core objets et des certificats existants.

7. Créez des certificats, des paires de clés et des AWS IoT politiques pour chaque appareil connecté à votre Greengrass Core.

AWS IoT Greengrass principes de sécurité fondamentaux

Le noyau de Greengrass utilise les principes de sécurité suivants : AWS IoT client, serveur MQTT local et gestionnaire de secrets local. La configuration de ces mandataires est stockée dans l'objet `crypto` situé dans le fichier de configuration `config.json`. Pour de plus amples informations, veuillez consulter the section called “Fichier de configuration de AWS IoT Greengrass Core”.

Cette configuration inclut le chemin d'accès à la clé privée utilisée par le composant principal pour l'authentification et le chiffrement. AWS IoT Greengrass prend en charge deux modes de clé privée : le stockage basé sur le matériel ou le stockage basé sur le système de fichiers (par défaut). Pour plus d'informations sur le stockage des clés sur les modules de sécurité matérielle, consultez the section called “Intégration de sécurité matérielle”.

AWS IoT Client

Le AWS IoT client (client IoT) gère la communication sur Internet entre le cœur de Greengrass et. AWS IoT Core AWS IoT Greengrass utilise des certificats X.509 avec des clés publiques et privées pour l'authentification mutuelle lors de l'établissement de connexions TLS pour cette communication. Pour de plus amples informations, veuillez consulter Certificats X.509 et AWS IoT Core dans le Manuel du développeur AWS IoT Core .

Le client IoT prend en charge les certificats et les clés RSA et EC. Les chemins du certificat et de la clé privée sont spécifiés pour le mandataire `IoTCertificate` dans `config.json`.

Serveur MQTT

Le serveur MQTT local gère les communications sur le réseau local entre le cœur de Greengrass et les appareils clients du groupe. AWS IoT Greengrass utilise des certificats X.509 avec des clés publiques et privées pour l'authentification mutuelle lors de l'établissement de connexions TLS pour cette communication.

Par défaut, AWS IoT Greengrass génère une clé privée RSA pour vous. Pour configurer l’appareil principal (core) pour utiliser une autre clé privée, vous devez fournir le chemin de la clé du mandataire `MQTTServerCertificate` dans `config.json`. Vous êtes responsable de la rotation d'une clé fournie par le client.

Prise en charge des clés privées

	Clé RSA	Clé EC
Type de clé	Supported	Supported
Paramètres clés	Minimum 2048-bit length	NIST P-256 or NIST P-384 curve
Format de disque	PKCS#1, PKCS#8	SECG1, PKCS#8
Version GGC minimale	• Utilisez la clé RSA par défaut : 1.0	• Spécifiez une clé EC : 1.9

Clé RSA Clé EC

- Spécifiez une clé RSA : 1.7

C'est la configuration de la clé privée qui détermine les processus connexes. Pour obtenir la liste des suites de chiffrement que l'appareil principal (core) Greengrass prend en charge en tant que serveur, consultez the section called "Prise en charge des suites de chiffrement TLS".

Si aucune clé privée n'est spécifiée (par défaut)

- AWS IoT Greengrass fait pivoter la clé en fonction de vos paramètres de rotation.
- L'appareil principal (core) génère une clé RSA, qui est utilisée pour générer le certificat.
- Le certificat du serveur MQTT possède une clé publique RSA et une signature RSA SHA-256.

Si une clé privée RSA est spécifiée (nécessite GGC v1.7 ou version ultérieure)

- Vous êtes responsable de la rotation de la clé.
- L'appareil principal (core) utilise la clé spécifiée pour générer le certificat.
- La clé RSA doit avoir une longueur minimale de 2 048 bits.
- Le certificat du serveur MQTT possède une clé publique RSA et une signature RSA SHA-256.

Si une clé privée EC est spécifiée (nécessite GGC v1.9 ou version ultérieure)

- Vous êtes responsable de la rotation de la clé.
- L'appareil principal (core) utilise la clé spécifiée pour générer le certificat.
- La clé privée EC doit utiliser une courbe NIST P-256 ou NIST P-384.
- Le certificat du serveur MQTT possède une clé publique EC et une signature RSA SHA-256.

  Le certificat du serveur MQTT présenté par l'appareil principal (core) dispose d'une signature RSA SHA-256, quel que soit le type de clé. Pour cette raison, les clients doivent prendre en charge la validation de certificat RSA SHA-256 pour établir une connexion sécurisée avec l'appareil principal (core).

Secrets Manager

Le gestionnaire de secrets locaux gère en toute sécurité les copies locales des secrets que vous créez dans AWS Secrets Manager. Il utilise une clé privée pour sécuriser la clé de données qui est utilisée pour chiffrer les secrets. Pour de plus amples informations, veuillez consulter Déployer des secrets sur Core.

Par défaut, la clé privée du client IoT est utilisée, mais vous pouvez spécifier une autre clé privée pour le mandataire `SecretsManager` dans `config.json`. Seul le type de clé RSA est pris en charge. Pour de plus amples informations, veuillez consulter the section called “Spécifier la clé privée pour chiffrer un secret”.

> **Note**
>
> Actuellement, ne AWS IoT Greengrass prend en charge que le mécanisme de remplissage PKCS #1 v1.5 pour le chiffrement et le déchiffrement des secrets locaux lors de l'utilisation de clés privées matérielles. Si vous suivez les instructions fournies par le fournisseur pour générer manuellement des clés privées matérielles, assurez-vous de choisir PKCS #1 v1.5. AWS IoT Greengrass ne prend pas en charge le rembourrage asymétrique optimal (OAEP).

Prise en charge des clés privées

	Clé RSA	Clé EC
Type de clé	Supported	Not supported
Paramètres clés	Minimum 2048-bit length	Not applicable
Format de disque	PKCS#1, PKCS#8	Not applicable
Version GGC minimale	1.7	Not applicable

Abonnements gérés dans le flux de travail de messagerie MQTT

AWS IoT Greengrass utilise une table d'abonnement pour définir comment les messages MQTT peuvent être échangés entre les appareils clients, les fonctions et les connecteurs d'un groupe Greengrass, et AWS IoT Core avec ou avec le service fantôme local. Chaque abonnement spécifie une source, une cible et un sujet (ou sujet) MQTT sur lesquels les messages sont envoyés ou reçus. AWS IoT Greengrass autorise l'envoi de messages d'une source vers une cible uniquement si un abonnement correspondant est défini.

Un abonnement définit le flux de messages dans une seule direction, de la source vers la cible. Pour prendre en charge l’échange bidirectionnel de messages, vous devez créer deux abonnements, une pour chaque direction.

Prise en charge des suites de chiffrement TLS

AWS IoT Greengrass utilise le modèle de sécurité AWS IoT Core des transports pour chiffrer les communications avec le cloud à l'aide de suites de chiffrement TLS. De plus, AWS IoT Greengrass les données sont cryptées lorsqu'elles sont au repos (dans le cloud). Pour plus d'informations sur la sécurité du AWS IoT Core transport et les suites de chiffrement prises en charge, consultez la section Sécurité du transport dans le guide du AWS IoT Core développeur.

Suites de chiffrement prises en charge pour les communications réseau locales

En revanche AWS IoT Core, le AWS IoT Greengrass cœur prend en charge les suites de chiffrement TLS du réseau local suivantes pour les algorithmes de signature de certificats. Toutes ces suites de chiffrement sont prises en charge lorsque des clés privées sont stockées sur le système de fichiers. Un sous-ensemble est pris en charge lorsque l'appareil principal (core) est configuré pour utiliser des modules de sécurité matériels (HSM). Pour plus d'informations, consultez the section called "Mandataires de sécurité" et the section called "Intégration de sécurité matérielle". Le tableau indique également la version minimale du logiciel AWS IoT Greengrass Core requise pour le support.

	Chiffrement	Prise en charge de HSM	Version GGC minimale
TLSv1.2	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA	Supported	1.0
	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA	Supported	1.0
	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	Supported	1.0
	TLS_RSA_WITH_AES_128_CBC_SHA	Not supported	1.0

	Chiffrement	Prise en charge de HSM	Version GGC minimale
	TLS_RSA_WITH_AES_128_GCM_SHA256	Not supported	1.0
	TLS_RSA_WITH_AES_256_CBC_SHA	Not supported	1.0
	TLS_RSA_WITH_AES_256_GCM_SHA384	Not supported	1.0
	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	Supported	1.9
	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	Supported	1.9
TLSv1.1	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA	Supported	1.0
	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA	Supported	1.0
	TLS_RSA_WITH_AES_128_CBC_SHA	Not supported	1.0

	Chiffrement	Prise en charge de HSM	Version GGC minimale
	TLS_RSA_WITH_AES_256_CBC_SHA	Not supported	1.0
TLS v1.0	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA	Supported	1.0
	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA	Supported	1.0
	TLS_RSA_WITH_AES_128_CBC_SHA	Not supported	1.0
	TLS_RSA_WITH_AES_256_CBC_SHA	Not supported	1.0

Protection des données dans AWS IoT Greengrass

Le modèle de responsabilité partagée AWS s'applique à la protection des données dans AWS IoT Greengrass. Comme décrit dans ce modèle, AWS est responsable de la protection de l'infrastructure globale sur laquelle l'ensemble du AWS Cloud s'exécute. La gestion du contrôle de votre contenu hébergé sur cette infrastructure relève de votre responsabilité. Vous êtes également responsable des tâches de configuration et de gestion de la sécurité des Services AWS que vous utilisez. Pour en savoir plus sur la confidentialité des données, consultez Questions fréquentes (FAQ) sur la confidentialité des données. Pour en savoir plus sur la protection des données en Europe, consultez le billet de blog Modèle de responsabilité partagée AWSet RGPD (Règlement général sur la protection des données) sur le AWSBlog de sécurité.

À des fins de protection des données, nous vous recommandons de protéger les informations d'identification Compte AWS et de configurer les comptes utilisateur individuels avec AWS IAM Identity Center ou AWS Identity and Access Management (IAM). Ainsi, chaque utilisateur se

voit attribuer uniquement les autorisations nécessaires pour exécuter ses tâches. Nous vous recommandons également de sécuriser vos données comme indiqué ci-dessous :

- Utilisez l'authentification multifactorielle (MFA) avec chaque compte.
- Utilisez les certificats SSL/TLS pour communiquer avec les ressources AWS. Nous exigeons TLS 1.2 et recommandons TLS 1.3.
- Configurez une API (Interface de programmation) et le journal de l'activité des utilisateurs avec AWS CloudTrail.
- Utilisez des solutions de chiffrement AWS, ainsi que tous les contrôles de sécurité par défaut au sein des Services AWS.
- Utilisez des services de sécurité gérés avancés tels qu'Amazon Macie, qui contribuent à la découverte et à la sécurisation des données sensibles stockées dans Amazon S3.
- Si vous avez besoin de modules cryptographiques validés FIPS (Federal Information Processing Standard) 140-2 lorsque vous accédez à AWS via une CLI (Interface de ligne de commande) ou une API (Interface de programmation), utilisez un point de terminaison FIPS (Federal Information Processing Standard). Pour en savoir plus sur les points de terminaison FIPS (Federal Information Processing Standard) disponibles, consultez Federal Information Processing Standard (FIPS) 140-2 (Normes de traitement de l'information fédérale).

Nous vous recommandons fortement de ne jamais placer d'informations confidentielles ou sensibles, telles que les adresses e-mail de vos clients, dans des balises ou des champs de texte libre tels que le champ Name (Nom). Cela est également valable lorsque vous utilisez AWS IoT Greengrass ou d'autres Services AWS à l'aide de la console, de l'API, d'AWS CLI ou des kits SDK AWS. Toutes les données que vous saisissez dans des balises ou des champs de texte de forme libre utilisés pour les noms peuvent être utilisées à des fins de facturation ou dans les journaux de diagnostic. Si vous fournissez une adresse URL à un serveur externe, nous vous recommandons fortement de ne pas inclure d'informations d'identification dans l'adresse URL permettant de valider votre demande adressée à ce serveur.

Pour plus d'informations sur la protection des informations sensibles dansAWS IoT Greengrass, consultezthe section called "Ne journalisez pas les informations sensibles".

Pour en savoir plus sur la protection des données, consultez le billet de blog Modèle de responsabilité partagée AWS et RGPD sur le Blog sur la sécurité d'AWS.

Rubriques

- Chiffrement des données

- Intégration de sécurité matérielle

Chiffrement des données

AWS IoT Greengrassutilise le chiffrement pour protéger les données en transit (sur Internet ou sur un réseau local) et au repos (stockées dans leAWS Cloud).

Les appareils d'un environnement AWS IoT Greengrass recueillent souvent des données envoyées aux services AWS pour traitement ultérieur. Pour plus d'informations sur le chiffrement des données dans d'autres services AWS, consultez la documentation relative à la sécurité du service concerné.

Rubriques

- Chiffrement en transit
- Chiffrement au repos
- Gestion des clés pour l’appareil principal Greengrass (noyau)

Chiffrement en transit

AWS IoT Greengrass dispose de trois modes de communication où les données sont en transit :

- the section called “Données en transit sur Internet”. Communication entre un noyau Greengrass etAWS IoT Greengrasssur Internet est chiffrée.
- the section called “Données en transit sur le réseau local”. La communication entre un noyau Greengrass et les appareils clients sur un réseau local est chiffrée.
- the section called “Données sur l’appareil principal (noyau)”. La communication entre les composants de l’appareil noyau Greengrass n'est pas chiffrée.

Données en transit sur Internet

AWS IoT Greengrass utilise le protocole TLS (Transport Layer Security) pour chiffrer toutes les communications sur Internet. Toutes les données envoyées auAWS Cloudest envoyé via une connexion TLS à l'aide des protocoles MQTT ou HTTPS, de sorte qu'elles sont sécurisées par défaut.AWS IoT Greengrassutilise le kitAWS IoTmodèle de sécurité des transports. Pour de plus amples informations, veuillez consulter Sécurité du transport dans le Manuel du développeur AWS IoT Core.

Données en transit sur le réseau local

AWS IoT Greengrassutilise TLS pour chiffrer toutes les communications sur le réseau local entre le noyau Greengrass et les appareils clients. Pour de plus amples informations, veuillez consulter les informations relatives aux suites de chiffrement prises en charge pour la communication sur un réseau local.

Il est de votre responsabilité de protéger le réseau local et les clés privées.

Pour les appareils noyau Greengrass, il est de votre responsabilité de :

- Maintenir le noyau à jour avec les derniers correctifs de sécurité.
- Maintenir les bibliothèques système à jour avec les derniers correctifs de sécurité.
- Protéger les clés privées. Pour plus d'informations, consultez the section called “Gestion des clés”.

Pour les appareils clients, il est de votre responsabilité de :

- Maintenir la pile TLS à jour.
- Protéger les clés privées.

Données sur l’appareil principal (noyau)

AWS IoT Greengrass ne chiffre pas les données échangées localement sur l’appareil noyau Greengrass car les données ne quittent pas l’appareil. Cela inclut la communication entre les fonctions Lambda définies par l'utilisateur, les connecteurs, le kitAWS IoT GreengrassCore SDK et les composants système comme le gestionnaire de flux.

Chiffrement au repos

AWS IoT Greengrass stocke vos données :

- the section called “Données au repos dans leAWS Cloud”. Ces données sont chiffrées.
- the section called “Données au repos sur le noyau Greengrass”. Ces données ne sont pas chiffrées (sauf les copies locales de vos secrets).

Données au repos dans leAWS Cloud

AWS IoT Greengrasschiffre les données client stockées dans leAWS Cloud. Ces données sont protégées à l'aide de clés AWS KMS gérées par AWS IoT Greengrass.

Données au repos sur le noyau Greengrass

AWS IoT Greengrass s' appuie sur les autorisations de fichiers Unix et le chiffrement complet du disque (si activé) pour protéger les données au repos sur le noyau. Il est de votre responsabilité de sécuriser le système de fichiers et l'appareil.

Cependant, AWS IoT Greengrass ne chiffre pas les copies locales de vos secrets récupérés à partir de AWS Secrets Manager. Pour plus d'informations, consultez the section called “Chiffrement des secrets”.

Gestion des clés pour l’appareil principal Greengrass (noyau)

Il incombe au client de garantir le stockage sécurisé des clés cryptographiques (publiques et privées) sur l’appareil noyau Greengrass. AWS IoT Greengrass utilise des clés publiques et privées pour les scénarios suivants :

- La clé client IoT est utilisée avec le certificat IoT pour authentifier la liaison TLS (Transport Layer Security) lorsqu'un noyau Greengrass se connecte à AWS IoT Core. Pour plus d'informations, consultez the section called “Authentification et autorisation d'appareil”.

  > **Note**
  >
  > La clé et le certificat sont également appelés clé privée principale et certificat d’appareil noyau.

- La clé de serveur MQTT est utilisée par le certificat de serveur MQTT pour authentifier les connexions TLS entre l'appareil noyau et les appareils clients. Pour plus d'informations, consultez the section called “Authentification et autorisation d'appareil”.
- Le gestionnaire de secrets locaux utilise également la clé client IoT pour protéger la clé de données utilisée pour chiffrer les secrets locaux, mais vous pouvez fournir votre propre clé privée. Pour plus d'informations, consultez the section called “Chiffrement des secrets”.

Un noyau Greengrass prend en charge le stockage de clés privées via des autorisations de système de fichiers, des modules de sécurité matérielle, ou les deux. Si vous utilisez des clés privées basées sur le système de fichiers, vous êtes responsable de leur stockage sécurisé sur l’appareil noyau.

Sur un noyau Greengrass, l'emplacement de vos clés privées est spécifié dans la section `crypto` du fichier `config.json`. Si vous configurez le noyau pour qu'il utilise une clé fournie par le client pour

le certificat de serveur MQTT, il est de votre responsabilité de procéder à la rotation de la clé. Pour plus d'informations, consultez the section called “Mandataires de sécurité”.

Pour les appareils clients, il est de votre responsabilité de maintenir la pile TLS à jour et de protéger les clés privées. Les clés privées sont utilisées avec les certificats d’appareil pour authentifier les connexions TLS avec le service AWS IoT Greengrass.

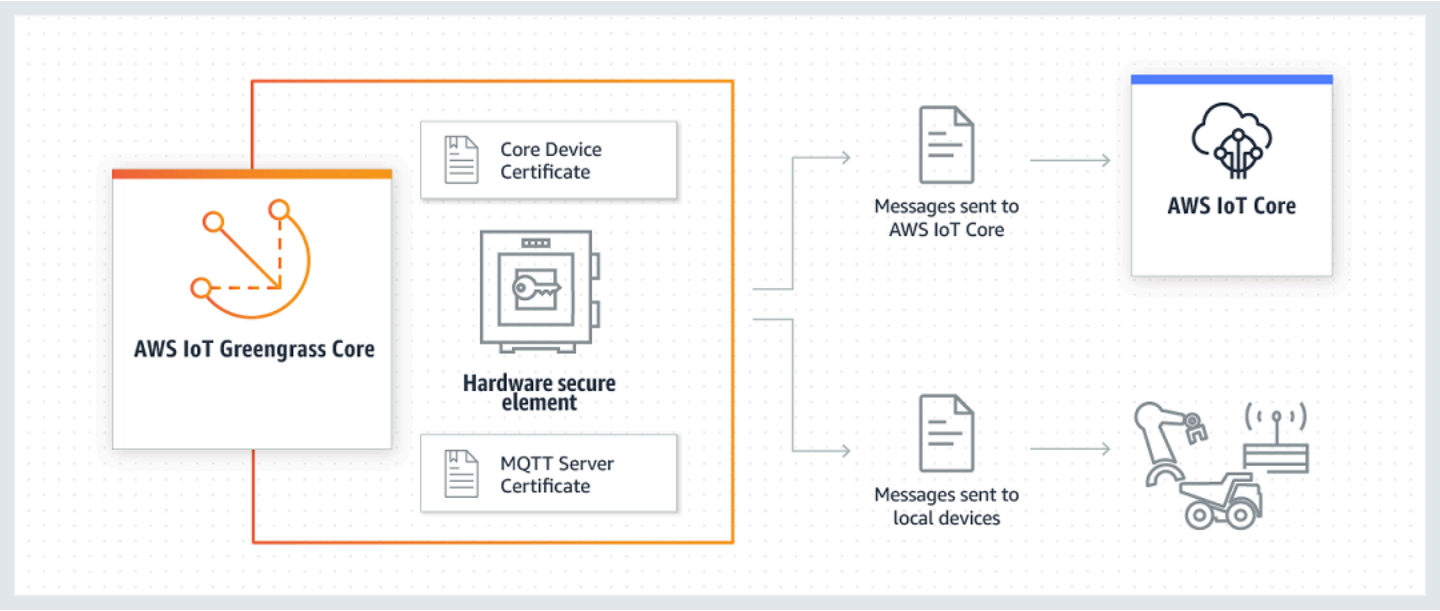
Intégration de sécurité matérielle

Cette fonction est disponible uniquement pourAWS IoT GreengrassCore version 1.7 et versions ultérieures.

AWS IoT Greengrass prend en charge l'utilisation de modules de sécurité matérielle (HSM) via l'interface PKCS#11 pour le stockage sécurisé et le déchargement des clés privées. Cela empêche les clés d'être exposées ou dupliquées dans les logiciels. Les clés privées peuvent être stockées de manière sécurisée sur des modules matériels, tels que les HSM, les modules de plateforme sécurisés (TPM) ou d'autres éléments de chiffrement.

Recherchez les appareils qualifiés pour cette fonctionnalité dans leAWS PartnerCatalogue d'appareils.

Le schéma suivant illustre l'architecture de sécurité matérielle d'unAWS IoT Greengrasscore.



Sur une installation standard, AWS IoT Greengrass utilise deux clés privées. Une clé est utilisée par le composant client AWS IoT (client IoT) lors de la négociation TLS (Transport Layer Security) lorsqu'un noyau Greengrass se connecte à AWS IoT Core. (Cette clé est également appelée la clé

privée principale.) L'autre clé est utilisée par le serveur MQTT local, ce qui permet aux appareils Greengrass de communiquer avec le noyau Greengrass. Si vous souhaitez utiliser la sécurité matérielle des deux composants, vous pouvez utiliser une clé privée partagée ou des clés privées séparées. Pour plus d'informations, consultez the section called " Pratiques d'allocation".

> **Note**
>
> Sur une installation standard, le Secrets Manager local utilise également la clé du client IoT pour ses processus de chiffrement, mais vous pouvez utiliser votre propre clé privée. Il doit s'agir d'une clé RSA avec une longueur minimale de 2 048 bits. Pour plus d'informations, consultez the section called "Spécifier la clé privée pour chiffrer un secret".

Prérequis

Pour pouvoir configurer la sécurité matérielle d'un noyau Greengrass, vous devez disposer des éléments suivants :

- Un module de sécurité matériel (HSM) qui prend en charge votre configuration de clé privée cible pour le client IoT, le serveur MQTT local et les composants Secrets Manager locaux. La configuration peut inclure une, deux ou trois clés privées basée sur le matériel, selon que vous configurez les composants pour partager les clés ou non. Pour plus d'informations sur la prise en charge des clés privées, consultez the section called "Mandataires de sécurité".
  - Pour les clés RSA : Une clé RSA de 2048 bits (ou plus) etPKCS #1 v1.5méthode de signature.
  - Pour les clés EC : Une courbe NIST P-256 ou NIST P-384.

  > **Note**
  >
  > Recherchez les appareils qualifiés pour cette fonctionnalité dans leAWS PartnerCatalogue d'appareils.

- Fournisseur de bibliothèque PKCS#11 qui est chargeable au moment de l'exécution (en utilisant libdl) et qui fournit des fonctions PKCS#11.
- Le module matériel doit être résolu par étiquette d'emplacement, tel que défini dans la spécification PKCS#11.
- La clé privée doit être générée et chargée sur le HSM en utilisant les outils de mise en service fournis par le fournisseur.

- La clé privée doit être résolue par étiquette d'objet.
- Le certificat de l'appareil noyau. Il s'agit d'un certificat client IoT qui correspond à la clé privée.
- Si vous utilisez l'agent de mise à jour Greengrass OTA,PKCS #11La bibliothèque wrapper doit être installée. Pour plus d'informations, consultez the section called “Configurer les mises à jour OTA”.

En outre, vérifiez que les conditions suivantes sont réunies :

- Les certificats de client IoT qui sont associés à la clé privée sont enregistrés dans AWS IoT et activés. Vous pouvez le vérifier dans laAWS IoTconsole sousGérer, développezTous les appareils, choisissezObjetset choisissez leCertificatsonglet pour le cœur.
- LeAWS IoT GreengrassLe logiciel version 1.7 ou plus est installé sur l'appareil principal, comme décrit dans la sectionModule 2du didacticiel de mise en route. La version 1.9 ou plus est nécessaire pour utiliser une clé EC pour le serveur MQTT.
- Les certificats sont attachés au noyau Greengrass. Vous pouvez le vérifier à l'aide de laGérerpour l'élément principal dans laAWS IoTconsole

> **Note**
>
> Actuellement, AWS IoT Greengrass ne prend pas en charge le chargement du certificat CA ou du certificat du client IoT directement depuis le HSM. Les certificats doivent être chargés en tant que fichiers de texte brut sur le système de fichiers dans un emplacement lisible par Greengrass.

Configuration de sécurité matérielle pour un appareil AWS IoT Greengrass Core

La sécurité matérielle est configurée dans le fichier de configuration de Greengrass. Il s'agit du fichier `config.json` qui se trouve dans le répertoire `/greengrass-root/config`.

> **Note**
>
> Pour passer en revue le processus de mise en place d'une configuration HSM à l'aide d'une implémentation purement logicielle, consultez the section called “Module 7 : Simulation d'intégration de sécurité matérielle”.

> **Important**
>
> La configuration simulée de l'exemple n'apporte aucun avantage de sécurité. Elle a pour objectif de vous permettre de découvrir la spécification PKCS#11 et d'effectuer des tests initiaux sur votre logiciel si vous prévoyez d'utiliser un HSM basé sur le matériel.

Pour configurer la sécurité matérielle dans AWS IoT Greengrass, vous devez modifier l'objet `crypto` dans `config.json`.

Lorsque vous utilisez la sécurité matérielle, l'objet `crypto` est utilisé pour spécifier les chemins d'accès aux certificats, clés privées et ressources de la bibliothèque du fournisseur PKCS # 11 sur le noyau, comme illustré dans l'exemple suivant.

```
"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'objet `crypto` contient les propriétés suivantes :

Champ	Description	Remarques
`Chemin CApath`	Chemin d'accès absolu au certificat d'autorité de certification racine AWS IoT.	Il doit s'agir de l'URI d'un fichier du formulaire : `file:///absolute/path/to/file` . **Note** Vérifiez que votre les points de terminaison correspondent à votre type de certificat.
`PKCS11`		
`Moteur OpenSSL`	Facultatif. Chemin d'accès absolu au fichier de moteur OpenSSL `.so` pour activer la prise en charge PKCS # 11 sur OpenSSL.	Doit être un chemin d'accès à un fichier sur le système de fichiers. Cette propriété est requise si vous utilisez l'agent de mise à jour Greengrass OTA avec la sécurité matérielle. Pour plus d'informations, consultez the section called “Configurer les mises à jour OTA”.
`Fournisseur P11`	Chemin absolu vers la bibliothèque libdl-loadable d'implémentation PKCS # 11.	Doit être un chemin d'accès à un fichier sur le système de fichiers.
`Étiquette de fente`	Étiquette d'emplacement qui est utilisée pour identifier le module matériel.	Doit se conformer aux spécifications d'étiquette PKCS # 11.

Champ	Description	Remarques
`slotUserPin`	PIN de l'utilisateur qui est utilisé pour authentifier le noyau Greengrass au module.	Vous devez disposer d'autorisations suffisantes pour exécuter C_Sign avec les clés privées configurées.
`principals`		
`Certificat IoT`	The certificate and private key that the core uses to make requests to AWS IoT.	
`Certificat IoT.privateKeyPath`	Chemin d'accès à la clé privée principale.	Pour le stockage de système de fichiers, il doit s'agir de l'URI d'un fichier du formulaire : `file:///`*`absolute/path/to/file`* . Pour le stockage HSM, il doit s'agir du chemin d'accès RFC WF-7512 PKCS # 11 qui spécifie l'objet étiquette.
`Certificat IoT. Chemin du certificat`	Chemin d'accès absolu au certificat de votre noyau.	Il doit s'agir de l'URI d'un fichier du formulaire : `file:///`*`absolute/path/to/file`* .
`MQTTServerCertificate`	Facultatif. Clé privée que le noyau utilise en combinaison avec le certificat pour agir en tant que passerelle ou serveur MQTT.	

Champ	Description	Remarques
`MQTTServerCertificate .privateKeyPath`	Chemin d'accès à la clé privée du serveur MQTT local.	Utilisez cette valeur pour spécifier votre propre clé privée pour le serveur MQTT local. Pour le stockage de système de fichiers, il doit s'agir de l'URI d'un fichier du formulaire : `file:///`*`absolute/path/to/file`* . Pour le stockage HSM, il doit s'agir du chemin d'accès RFC WF-7512 PKCS # 11 qui spécifie l'objet étiquette. Si cette propriété n'est pas spécifiée, AWS IoT Greengrass effectue une rotation de la clé basée sur vos paramètres de rotation. Si cette propriété est spécifiée, le client est responsable de la rotation de la clé.
`SecretsManager`	The private key that secures the data key used for encryption. For more information, see Déployer des secrets sur Core.	

Champ	Description	Remarques
`SecretsManager .privateKeyPath`	Chemin d'accès à la clé privée du Secrets Manager local.	Seule une clé RSA est prise en charge. Pour le stockage de système de fichiers, il doit s'agir de l'URI d'un fichier du formulaire : `file:///`*`absolute/path/to/file`* . Pour le stockage HSM, il doit s'agir du chemin d'accès RFC WF-7512 PKCS # 11 qui spécifie l'objet étiquette. La clé privée doit être générée à l'aide du mécanisme de remplissage PKCS#1 v1.5.

Champ	Description	Remarques
`Chemin CApath`	Chemin d'accès absolu au certificat d'autorité de certification racine AWS IoT.	Il doit s'agir de l'URI d'un fichier du formulaire : `file:///`*`absolute/path/to/file`* . **Note** Vérifiez que votre les points de terminaison correspondent à votre type de certificat.
`PKCS11`		

Champ	Description	Remarques
`Moteur OpenSSL`	Facultatif. Chemin d'accès absolu au fichier de moteur OpenSSL `.so` pour activer la prise en charge PKCS # 11 sur OpenSSL.	Doit être un chemin d'accès à un fichier sur le système de fichiers. Cette propriété est requise si vous utilisez l'agent de mise à jour Greengrass OTA avec la sécurité matérielle. Pour plus d'informations, consultez the section called “Configurer les mises à jour OTA”.
`Fournisseur P11`	Chemin absolu vers la bibliothèque libdl-loadable d'implémentation PKCS # 11.	Doit être un chemin d'accès à un fichier sur le système de fichiers.
`Étiquette de fente`	Étiquette d'emplacement qui est utilisée pour identifier le module matériel.	Doit se conformer aux spécifications d'étiquette PKCS # 11.
`slotUserPin`	PIN de l'utilisateur qui est utilisé pour authentifier le noyau Greengrass au module.	Vous devez disposer d'autorisations suffisantes pour exécuter C_Sign avec les clés privées configurées.
`principals`		
`Certificat IoT`	The certificate and private key that the core uses to make requests to AWS IoT.	

Champ	Description	Remarques
`Certificat IoT.privateKeyPath`	Chemin d'accès à la clé privée principale.	Pour le stockage de système de fichiers, il doit s'agir de l'URI d'un fichier du formulaire : `file:///`*`absolute/path/to/file`* . Pour le stockage HSM, il doit s'agir du chemin d'accès RFC WF-7512 PKCS # 11 qui spécifie l'objet étiquette.
`Certificat IoT. Chemin du certificat`	Chemin d'accès absolu au certificat de votre noyau.	Il doit s'agir de l'URI d'un fichier du formulaire : `file:///`*`absolute/path/to/file`* .
`MQTTServerCertificate`	Facultatif. Clé privée que le noyau utilise en combinaison avec le certificat pour agir en tant que passerelle ou serveur MQTT.	

Champ	Description	Remarques
`MQTTServerCertificate .privateKeyPath`	Chemin d'accès à la clé privée du serveur MQTT local.	Utilisez cette valeur pour spécifier votre propre clé privée pour le serveur MQTT local. Pour le stockage de système de fichiers, il doit s'agir de l'URI d'un fichier du formulaire : `file:///`*`absolute/path/to/file`* . Pour le stockage HSM, il doit s'agir du chemin d'accès RFC WF-7512 PKCS # 11 qui spécifie l'objet étiquette. Si cette propriété n'est pas spécifiée, AWS IoT Greengrass effectue une rotation de la clé basée sur vos paramètres de rotation. Si cette propriété est spécifiée, le client est responsable de la rotation de la clé.
`SecretsManager`	The private key that secures the data key used for encryption. For more information, see Déployer des secrets sur Core.	

Champ	Description	Remarques
`SecretsManager .privateKeyPath`	Chemin d'accès à la clé privée du Secrets Manager local.	Seule une clé RSA est prise en charge. Pour le stockage de système de fichiers, il doit s'agir de l'URI d'un fichier du formulaire : `file:///`*`absolute/path/to/file`* . Pour le stockage HSM, il doit s'agir du chemin d'accès RFC WF-7512 PKCS # 11 qui spécifie l'objet étiquette. La clé privée doit être générée à l'aide du mécanisme de remplissage PKCS#1 v1.5.

Champ	Description	Remarques
`Chemin CApath`	Chemin d'accès absolu au certificat d'autorité de certification racine AWS IoT.	Il doit s'agir de l'URI d'un fichier du formulaire : `file:///`*`absolute/path/to/file`* . **Note** Vérifiez que votre les points de terminaison correspondent à votre type de certificat.
`PKCS11`		

Champ	Description	Remarques
`Moteur OpenSSL`	Facultatif. Chemin d'accès absolu au fichier de moteur OpenSSL `.so` pour activer la prise en charge PKCS # 11 sur OpenSSL.	Doit être un chemin d'accès à un fichier sur le système de fichiers. Cette propriété est requise si vous utilisez l'agent de mise à jour Greengrass OTA avec la sécurité matérielle. Pour plus d'informations, consultez the section called “Configurer les mises à jour OTA”.
`Fournisseur P11`	Chemin absolu vers la bibliothèque libdl-loadable d'implémentation PKCS # 11.	Doit être un chemin d'accès à un fichier sur le système de fichiers.
`Étiquette de fente`	Étiquette d'emplacement qui est utilisée pour identifier le module matériel.	Doit se conformer aux spécifications d'étiquette PKCS # 11.
`slotUserPin`	PIN de l'utilisateur qui est utilisé pour authentifier le noyau Greengrass au module.	Vous devez disposer d'autorisations suffisantes pour exécuter C_Sign avec les clés privées configurées.
`principals`		
`Certificat IoT`	The certificate and private key that the core uses to make requests to AWS IoT.	

Champ	Description	Remarques
`Certificat IoT.privateKeyPath`	Chemin d'accès à la clé privée principale.	Pour le stockage de système de fichiers, il doit s'agir de l'URI d'un fichier du formulaire : `file:///`*`absolute/path/to/file`* . Pour le stockage HSM, il doit s'agir du chemin d'accès RFC WF-7512 PKCS # 11 qui spécifie l'objet étiquette.
`Certificat IoT. Chemin du certificat`	Chemin d'accès absolu au certificat de votre noyau.	Il doit s'agir de l'URI d'un fichier du formulaire : `file:///`*`absolute/path/to/file`* .
`MQTTServerCertificate`	Facultatif. Clé privée que le noyau utilise en combinaison avec le certificat pour agir en tant que passerelle ou serveur MQTT.	

Champ	Description	Remarques
`MQTTServerCertificate .privateKeyPath`	Chemin d'accès à la clé privée du serveur MQTT local.	Utilisez cette valeur pour spécifier votre propre clé privée pour le serveur MQTT local. Pour le stockage de système de fichiers, il doit s'agir de l'URI d'un fichier du formulaire : `file:///`*`absolute/path/to/file`* . Pour le stockage HSM, il doit s'agir du chemin d'accès RFC WF-7512 PKCS # 11 qui spécifie l'objet étiquette. Si cette propriété n'est pas spécifiée, AWS IoT Greengrass effectue une rotation de la clé basée sur vos paramètres de rotation. Si cette propriété est spécifiée, le client est responsable de la rotation de la clé.
`SecretsManager`	The private key that secures the data key used for encryption. For more information, see Déployer des secrets sur Core.	

Champ	Description	Remarques
`SecretsManager .privateKeyPath`	Chemin d'accès à la clé privée du Secrets Manager local.	Seule une clé RSA est prise en charge. Pour le stockage de système de fichiers, il doit s'agir de l'URI d'un fichier du formulaire : `file:///absolute/path/to/file` . Pour le stockage HSM, il doit s'agir du chemin d'accès RFC WF-7512 PKCS # 11 qui spécifie l'objet étiquette. La clé privée doit être générée à l'aide du mécanisme de remplissage PKCS#1 v1.5.

Pratiques d'allocation pour la sécurité matérielle AWS IoT Greengrass

Voici les pratiques de mise en service liées à la sécurité et aux performances.

Sécurité

- Générer des clés privées directement sur le HSM en utilisant le générateur de nombres aléatoires du matériel interne.

  > **Note**
  >
  > Si vous configurez des clés privées à utiliser avec cette fonction (en suivant les instructions indiquées par le fournisseur de matériel), sachez que AWS IoT Greengrass ne prend actuellement en charge que le mécanisme de remplissage PKCS1 v1.5 pour le chiffrement et le déchiffrement des secrets locaux. AWS IoT Greengrass ne prend pas en charge la fonctionnalité OAEP (Optimal Asymmetric Encryption Padding).

- Configurer les clés privées pour interdire l'exportation.

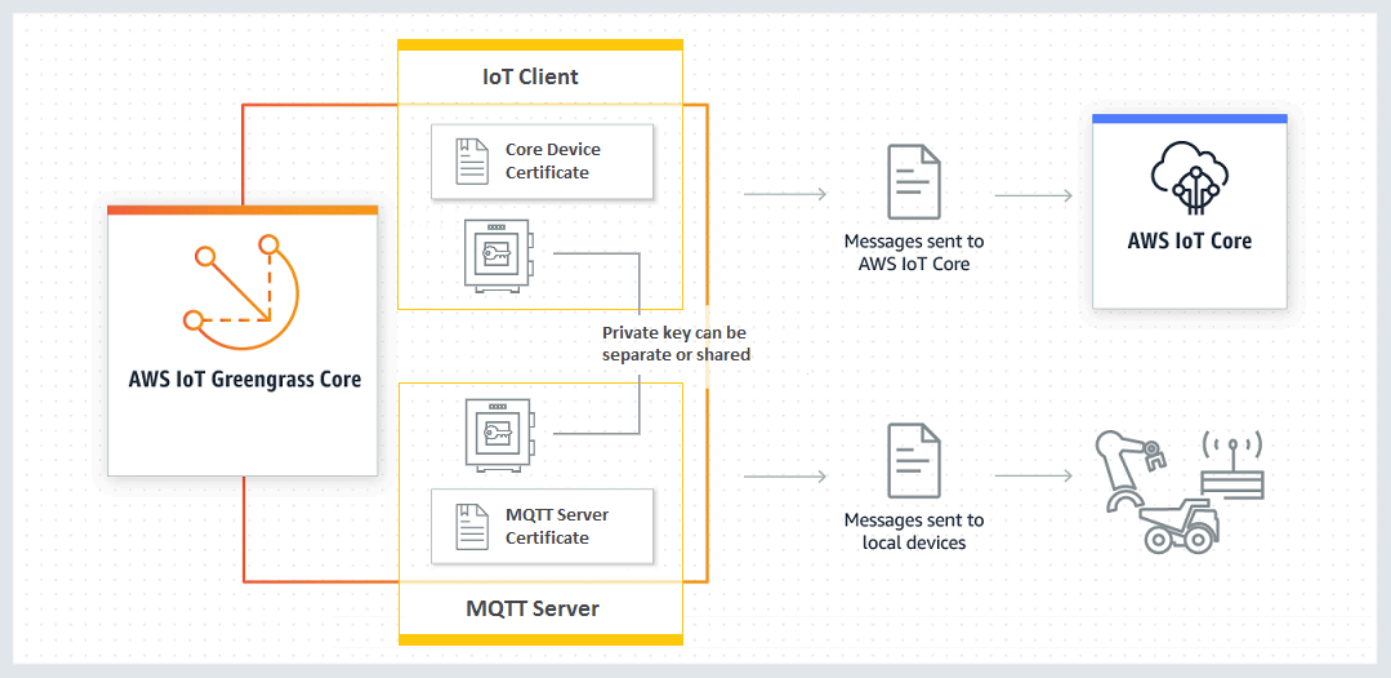
- Utilisez l'outil de mise en service par le fournisseur de matériel pour générer une demande de signature de certificat (CSR) à l'aide de la clé privée protégée par le matériel, puis utilisez l'AWS IoTpour générer un certificat client.

> **Note**
>
> La pratique de la rotation des clés ne s'applique pas lorsque les clés privées sont générées sur un HSM.

Performances

Le schéma suivant illustre le composant client IoT et le serveur MQTT local surAWS IoT Greengrasscore. Si vous souhaitez utiliser une configuration HSM pour les deux composants, vous pouvez utiliser la même clé privée partagée ou des clés privées distinctes. Si vous utilisez des clés distinctes, elles doivent être stockées dans le même emplacement.

> **Note**
>
> AWS IoT Greengrass n'impose aucune limite sur le nombre de clés que vous stockez sur le module HSM, afin que vous puissiez stocker les clés privées sur le client IoT, le serveur MQTT et les composants Secrets Manager. Toutefois, certains fournisseurs de modules HSM peuvent imposer des limites sur le nombre de clés que vous pouvez stocker dans un emplacement.



En général, la clé du client IoT n'est pas utilisée très fréquemment parce que le logiciel AWS IoT Greengrass Core maintient des connexions à longue durée de vie vers le cloud. Toutefois, la clé de serveur MQTT est utilisée chaque fois qu'un appareil Greengrass se connecte au noyau. Ces interactions affectent directement les performances.

Lorsque la clé de serveur MQTT est stockée sur le HSM, la vitesse à laquelle les appareils peuvent se connecter dépend du nombre d'opérations de signature RSA par seconde que le HSM peut effectuer. Par exemple, si le HSM prend 300 millisecondes pour effectuer une signature de RSASSA-PKCS1-v1.5 sur une clé privée RSA-2048, seuls trois appareils peuvent se connecter au noyau Greengrass par seconde. Une fois que les connexions sont effectuées, le HSM n'est plus utilisé et les quotas standard pour AWS IoT Greengrass s'appliquent.

Pour réduire les goulets d'étranglement de performances, vous pouvez stocker la clé privée du serveur MQTT sur le système de fichiers plutôt que sur le HSM. Avec cette configuration, le serveur MQTT se comporte comme si la sécurité matérielle n'était pas activée.

AWS IoT Greengrass prend en charge plusieurs configurations de stockage de clés pour le client IoT et les composants de serveur MQTT. Ainsi, vous pouvez bénéficier de conditions optimales pour vos besoins de sécurité et de performances. Le tableau suivant présente des exemples de configurations.

Configuration	Clé IoT	Clé MQTT	Performances
Clé partagée HSM	HSM : clé A	HSM : clé A	Limitée par le HSM ou l'UC
Clés distinctes pour le HSM	HSM : clé A	HSM : clé B	Limitée par le HSM ou l'UC
HSM pour IoT uniquement	HSM : clé A	Système de fichiers : clé B	Limitée par l'UC
Héritée	Système de fichiers : clé A	Système de fichiers : clé B	Limitée par l'UC

Pour configurer le noyau Greengrass pour utiliser les clés basées sur le système de fichiers pour le serveur MQTT, omettez la section `principals.MQTTServerCertificate` de `config.json` (ou spécifiez un chemin d'accès basé sur le fichiers vers la clé si vous n'utilisez pas la clé par défaut générée par AWS IoT Greengrass). L'objet `crypto` qui en résulte ressemble à ceci :

```
"crypto": {
  "PKCS11": {
    "OpenSSLEngine": "...",
    "P11Provider": "...",
    "slotLabel": "...",
    "slotUserPin": "..."
  },
  "principals": {
    "IoTCertificate": {
      "privateKeyPath": "...",
      "certificatePath": "..."
    },
    "SecretsManager": {
      "privateKeyPath": "..."
    }
  },
  "caPath" : "..."
}
```

Suites de chiffrement prises en charge pour l'intégration de sécurité matérielle

AWS IoT Greengrass prend en charge un ensemble de suites de chiffrement lorsque la sécurité matérielle est configurée sur l'appareil principal. Il s'agit d'un sous-ensemble de suites de chiffrement qui sont prises en charge lorsque l'appareil principal (core) est configuré pour utiliser une sécurité basée sur les fichiers. Pour plus d'informations, consultez the section called “Prise en charge des suites de chiffrement TLS”.

> **Note**
>
> En cas de connexion à l'appareil principal (core) Greengrass à partir d'appareils Greengrass sur le réseau local, veillez à utiliser l'une des suites de chiffrement prises en charge pour effectuer la connexion TLS.

Configurer le support pour over-the-air mises à jour

Pour activer over-the-air (OTA) de laAWS IoT GreengrassLogiciel principal : lorsque vous utilisez la sécurité matérielle, vous devez installer OpenSC libp11bibliothèque d'enveloppes PKCS #11et modifiez le fichier de configuration de Greengrass. Pour plus d'informations sur les mises à jour OTA, consultez Mises à jour OTA du logiciel AWS IoT Greengrass Core.

1. Arrêtez le démon Greengrass.

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

   > **Note**
   >
   > *greengrass-root* indique le chemin d'installation du logiciel AWS IoT Greengrass Core sur votre appareil. Généralement, il s'agit du répertoire `/greengrass`.

2. Installez le moteur OpenSSL. OpenSSL 1.0 ou 1.1 sont pris en charge.

   ```
   sudo apt-get install libengine-pkcs11-openssl
   ```

3. Trouvez le chemin d'accès au moteur OpenSSL (`libpkcs11.so`) sur votre système :

   a. Obtenez la liste des packages installés de la bibliothèque.

```
sudo dpkg -L libengine-pkcs11-openssl
```

Le fichier `libpkcs11.so` est situé dans le répertoire `engines`.

b. Copiez le chemin complet dans le fichier, par exemple, `/usr/lib/ssl/engines/libpkcs11.so`.

4. Ouvrez le fichier de configuration Greengrass. Il s'agit du fichier `config.json` dans le répertoire */greengrass-root*`/config`.
5. Pour la propriété `OpenSSLEngine`, entrez le chemin d'accès au fichier `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**
>
> Si la propriété `OpenSSLEngine` n'existe pas dans l'objet `PKCS11`, ajoutez-la.

6. Démarrez le démon Greengrass.

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

Rétrocompatibilité avec les versions antérieures du logiciel AWS IoT Greengrass Core

Le logiciel AWS IoT Greengrass Core avec la prise en charge de la sécurité matérielle est entièrement rétrocompatible avec les fichiers `config.json` qui sont générés pour la version v1.6 et antérieure. Si l'objet `crypto` n'est pas présent dans le fichier de configuration `config.json`,

AWS IoT Greengrass utilise les propriétés basées sur les fichiers `coreThing.certPath`, `coreThing.keyPath` et `coreThing.caPath`. Cette rétrocompatibilité s'applique aux mises à jour OTA de Greengrass, qui ne remplacent pas une configuration basée sur un fichier spécifié dans `config.json`.

Prise en charge du matériel sans PKCS # 11

La bibliothèque PKCS # 11 est généralement fournie par le fournisseur de matériel ou est en open source. Par exemple, avec du matériel compatible avec les normes (comme TPM1.2), il pourrait être possible d'utiliser un logiciel open source existant. Toutefois, si votre matériel ne dispose pas d'une implémentation de bibliothèque PKCS #11 correspondante ou si vous souhaitez écrire un fournisseur PKCS #11 personnalisé, vous devez contacter votreAWSUn représentant du support aux entreprises qui a des questions relatives à l'intégration

Consulter aussi

- PKCS #11 Cryptographic Token Interface Usage Guide Version 2.40. Révisé par John Leiseboer et Robert Griffin. 16 novembre 2014. OASIS Committee Remarque 02. http://docs.oasis-open.org/pkcs11/pkcs11-ug/v2.40/cn02/pkcs11-ug-v2.40-cn02.html. Dernière version : http://docs.oasis-open.org/pkcs11/pkcs11-ug/v2.40/pkcs11-ug-v2.40.html.
- RFC 7512
- PHOTOS #1 : RSA Encryption version 1.5

Authentification et autorisation d'appareil pour AWS IoT Greengrass

Dans les environnements AWS IoT Greengrass, les appareils utilisent des certificats X.509 pour l'authentification et des stratégies AWS IoT pour l'autorisation. Les certificats et les stratégies permettent aux appareils de se connecter en toute sécurité avec d’autres appareils, AWS IoT Core et AWS IoT Greengrass.

Les certificats X.509 sont des certificats numériques qui font appel à la norme d'infrastructure de clé publique X.509 pour associer une clé publique à l’identité contenue dans un certificat. Les certificats X.509 sont émis par une entité de confiance appelée autorité de certification (CA). Celle-ci gère un ou plusieurs certificats spéciaux appelés certificats d'autorité de certification, qu'elle utilise pour émettre des certificats X.509. Seule l'autorité du certificat a accès aux certificats d'autorité de certification.

Les stratégies AWS IoT définissent l'ensemble des opérations autorisées pour les appareils AWS IoT. Plus précisément, elles autorisent et refusent l'accès aux opérations de plan de données AWS IoT Core et AWS IoT Greengrass, telles que la publication de messages MQTT et la récupération de shadows d'appareil.

Tous les périphériques nécessitent une entrée dans le registre AWS IoT Core et un certificat X.509 activé avec une stratégie AWS IoT jointe. Les appareils se répartissent en deux catégories :

- Appareils noyau Greengrass. Les appareils noyau Greengrass utilisent des certificats et des stratégies AWS IoT pour se connecter en toute sécurité à AWS IoT Core. Les certificats et les politiques permettent également de AWS IoT Greengrass déployer des informations de configuration, des fonctions Lambda, des connecteurs et des abonnements gérés sur les appareils principaux.
- Appareils clients. Les appareils clients (également appelés appareils connectés, appareils Greengrass ou appareils) sont des appareils qui se connectent à un cœur Greengrass via MQTT. Ils utilisent des certificats et des politiques pour se connecter au AWS IoT Greengrass service AWS IoT Core et le utiliser. Cela permet aux appareils clients d'utiliser le service de AWS IoT Greengrass découverte pour rechercher un périphérique principal et s'y connecter. Un appareil client utilise le même certificat pour se connecter à la passerelle du AWS IoT Core périphérique et au périphérique principal. Les appareils clients utilisent également les informations de découverte pour l'authentification mutuelle avec le périphérique principal. Pour plus d'informations, consultez the section called “Flux de connexion des appareils” et the section called “Gérer l'authentification des appareils avec le noyau Greengrass”.

Certificats X.509

La communication entre les appareils principaux et clients et entre les appareils AWS IoT Core et/ou AWS IoT Greengrass doit être authentifiée. Cette authentification mutuelle est basée sur les certificats d'appareils X.509 enregistrés et sur des clés de chiffrement.

Dans un environnement AWS IoT Greengrass, les appareils utilisent des certificats avec des clés publiques et privées pour les connexions TLS (Transport Layer Security) suivantes :

- Le composant client AWS IoT sur le noyau Greengrass se connectant à AWS IoT Core et à AWS IoT Greengrass via Internet.
- Les appareils clients se connectent AWS IoT Greengrass à pour obtenir des informations de découverte de base sur Internet.

- Le composant du serveur MQTT situé sur le cœur de Greengrass se connecte aux appareils clients du groupe via le réseau local.

L'appareil AWS IoT Greengrass principal stocke les certificats à deux emplacements :

- Certificat de l'appareil noyau dans `/greengrass-root/certs`. En règle générale, le certificat de l'appareil noyau est nommé `hash.cert.pem` (par exemple, `86c84488a5.cert.pem`). Ce certificat est utilisé par le client AWS IoT pour l'authentification mutuelle lorsque le noyau se connecte aux services AWS IoT Core et AWS IoT Greengrass.
- Certificat de serveur MQTT dans `/greengrass-root/ggc/var/state/server`. Le certificat de serveur MQTT est nommé `server.crt`. Ce certificat est utilisé pour l'authentification mutuelle entre le serveur MQTT local (sur le noyau Greengrass) et les appareils Greengrass.

> **Note**
>
> *greengrass-root* indique le chemin d'installation du logiciel AWS IoT Greengrass Core sur votre appareil. Généralement, il s'agit du répertoire `/greengrass`.

Pour plus d'informations, consultez the section called “Mandataires de sécurité”.

Certificats d'autorité de certification (CA)

Les appareils principaux et les appareils clients téléchargent un certificat CA racine utilisé pour l'authentification AWS IoT Core et les AWS IoT Greengrass services. Nous vous recommandons d'utiliser un certificat d'autorité de certification racine Amazon Trust Services (ATS), tel que Amazon Root CA 1. Pour de plus amples informations, veuillez consulter Certificats d'autorité de certification pour l'authentification du serveur dans le Manuel du développeur AWS IoT Core.

> **Note**
>
> Votre type de certificat CA racine doit correspondre à votre point de terminaison. Utilisez un certificat d'autorité de certification racine ATS avec un point de terminaison ATS (de préférence) ou un certificat d'autorité de certification VeriSign racine avec un ancien point de terminaison. Seules certaines régions Amazon Web Services prennent en charge les anciens terminaux. Pour plus d'informations, consultez the section called “Les points de terminaison du service doivent correspondre au type de certificat”.

Les appareils clients téléchargent également le certificat CA du groupe Greengrass. Cela permet de valider le certificat du serveur MQTT sur le noyau Greengrass lors de l'authentification mutuelle. Pour plus d'informations, consultez the section called "Flux de connexion des appareils". L'expiration par défaut du certificat de serveur MQTT est de sept jours.

Rotation des certificats sur le serveur MQTT local

Les appareils clients utilisent le certificat du serveur MQTT local pour l'authentification mutuelle avec le périphérique principal de Greengrass. Par défaut, ce certificat expire au bout de 7 jours. Cette période limitée est basée sur les bonnes pratiques en matière de sécurité. Le certificat du serveur MQTT est signé par le certificat de l'autorité de certification de groupe, stocké dans le cloud.

Pour que la rotation des certificats ait lieu, votre appareil principal Greengrass doit être en ligne et pouvoir accéder directement au AWS IoT Greengrass service de façon régulière. Lorsque le certificat expire, le périphérique principal tente de se connecter au AWS IoT Greengrass service pour obtenir un nouveau certificat. Si la connexion réussit, l'appareil noyau télécharge un nouveau certificat de serveur principal MQTT et redémarre le service MQTT local. À ce stade, tous les appareils clients connectés au cœur sont déconnectés. Si le périphérique principal est hors ligne au moment de l'expiration, il ne reçoit pas le certificat de remplacement. Toute nouvelle tentative de connexion à l'appareil principal est rejetée. Les connexions existantes ne sont pas interrompues. Les appareils clients ne peuvent pas se connecter au périphérique principal tant que la connexion au AWS IoT Greengrass service n'est pas rétablie et qu'un nouveau certificat de serveur MQTT ne peut pas être téléchargé.

Vous pouvez définir le délai d'expiration sur une valeur comprise entre 7 et 30 jours, en fonction de vos besoins. Les rotations plus fréquentes nécessitent des connexions cloud plus fréquentes. Une rotation moins fréquente peut entraîner des problèmes de sécurité. Si vous souhaitez définir l'expiration du certificat sur une valeur supérieure à 30 jours, contactezAWS Support.

Dans la AWS IoT console, vous pouvez gérer le certificat sur la page Paramètres du groupe. Dans l'AWS IoT GreengrassAPI, vous pouvez utiliser l'UpdateGroupCertificateConfigurationaction.

Une fois le certificat de serveur MQTT expiré, toutes les tentatives effectuées pour sa validation échouent. Les appareils clients doivent être en mesure de détecter la panne et de mettre fin à la connexion.

Stratégies AWS IoT pour les opérations de plan de données

Utilisez des stratégies AWS IoT pour autoriser l'accès au plan de données AWS IoT Core et AWS IoT Greengrass. Le plan de données AWS IoT Core comprend des opérations pour les appareils,

les utilisateurs et les applications, telles que la connexion à AWS IoT Core et l'abonnement aux rubriques. Le plan de données AWS IoT Greengrass comprend des opérations pour les appareils Greengrass, telles que la récupération de déploiements et la mise à jour des informations de connectivité.

Une AWS IoT politique est un document JSON similaire à une politique IAM. Elle contient une ou plusieurs instructions de stratégie qui spécifient les propriétés suivantes :

- `Effect`. Le mode d'accès, qui peut être `Allow` ou`Deny`.
- `Action`. Liste des actions autorisées ou refusées par la politique.
- `Resource`. Liste des ressources sur lesquelles l'action est autorisée ou refusée.

AWS IoTles politiques * le prennent en charge en tant que caractère générique et traitent les caractères génériques MQTT (+et#) comme des chaînes littérales. Pour plus d'informations sur le * caractère générique, consultez la section Utilisation du caractère générique dans les ARN des ressources du Guide de l'AWS Identity and Access Managementutilisateur.

Pour de plus amples informations, veuillez consulter Stratégies AWS IoT et Actions de stratégie AWS IoT dans le Manuel du développeur AWS IoT Core.

> **Note**
>
> AWS IoT Corevous permet d'associer des AWS IoT politiques à des groupes d'objets afin de définir des autorisations pour des groupes d'appareils. Les politiques relatives aux groupes d'objets n'autorisent pas l'accès aux opérations du plan de AWS IoT Greengrass données. Pour autoriser un objet à accéder à une opération de plan de AWS IoT Greengrass données, ajoutez l'autorisation à une AWS IoT politique que vous associez au certificat de l'objet.

Actions de stratégie AWS IoT Greengrass

Actions de noyau Greengrass

AWS IoT Greengrass définit les actions de stratégie suivantes que les appareils noyau Greengrass peuvent utiliser dans les stratégies AWS IoT :

`greengrass:AssumeRoleForGroup`

Autorisation pour un appareil principal de Greengrass de récupérer des informations d'identification à l'aide de la fonction Lambda du système Token Exchange Service (TES). Les autorisations liées aux informations d'identification récupérées sont basées sur la stratégie attachée au rôle de groupe configuré.

Cette autorisation est vérifiée lorsqu'un appareil noyau Greengrass tente de récupérer des informations d'identification (en supposant que ces informations d'identification ne soient pas mises en cache localement).

`greengrass:CreateCertificate`

Autorisation pour un appareil noyau Greengrass de créer son propre certificat de serveur.

Cette autorisation est vérifiée lorsqu'un appareil noyau Greengrass crée un certificat. Les appareils noyau Greengrass tentent de créer un certificat de serveur lors de la première exécution, lorsque les informations de connectivité du noyau changent et lors des périodes de rotation désignées.

`greengrass:GetConnectivityInfo`

Autorisation pour un appareil noyau Greengrass de récupérer ses propres informations de connectivité.

Cette autorisation est vérifiée lorsqu'un appareil noyau Greengrass tente de récupérer ses informations de connectivité à partir de AWS IoT Core.

`greengrass:GetDeployment`

Autorisation pour un appareil noyau Greengrass de récupérer des déploiements.

Cette autorisation est vérifiée lorsqu'un appareil noyau Greengrass tente de récupérer des déploiements et des statuts de déploiement à partir du cloud.

`greengrass:GetDeploymentArtifacts`

Autorisation pour un périphérique principal de Greengrass de récupérer des artefacts de déploiement tels que des informations de groupe ou des fonctions Lambda.

Cette autorisation est vérifiée lorsqu'un appareil noyau Greengrass reçoit un déploiement, puis tente de récupérer des artefacts de déploiement.

`greengrass:UpdateConnectivityInfo`

Autorisation pour un appareil noyau Greengrass de mettre à jour ses propres informations de connectivité avec des informations d'adresse IP ou de nom d'hôte.

Cette autorisation est vérifiée lorsqu'un appareil noyau Greengrass tente de mettre à jour ses informations de connectivité dans le cloud.

`greengrass:UpdateCoreDeploymentStatus`

Autorisation pour un appareil noyau Greengrass de mettre à jour l'état d'un déploiement.

Cette autorisation est vérifiée lorsqu'un appareil noyau Greengrass reçoit un déploiement, puis tente de mettre à jour l'état du déploiement.

Actions des appareils Greengrass

AWS IoT Greengrassdéfinit les actions de stratégie suivantes que les appareils clients peuvent utiliser dans AWS IoT les politiques :

`greengrass:Discover`

Autorisation pour un appareil client d'utiliser l'API Discovery pour récupérer les informations de connectivité de base de son groupe et l'autorité de certification du groupe.

Cette autorisation est vérifiée lorsqu'un appareil client appelle l'API Discovery avec l'authentification mutuelle TLS.

Stratégie AWS IoT minimale pour l'appareil principal AWS IoT Greengrass (noyau)

L'exemple de stratégie suivant comprend l'ensemble d'actions minimum requis pour prendre en charge la fonctionnalité Greengrass de base pour l'appareil de votre noyau.

- La stratégie énumère les rubriques MQTT et les filtres de rubrique vers lesquels l'appareil du noyau peut publier des messages, auxquels il peut s'abonner et desquels il peut recevoir des messages, y compris les rubriques utilisées pour le shadow. Pour faciliter l'échange de messages entre les AWS IoT Core fonctions Lambda, les connecteurs et les appareils clients du groupe Greengrass, spécifiez les sujets et les filtres de sujets que vous souhaitez autoriser. Pour de plus amples

informations, veuillez consulter Exemples de stratégie de publication/abonnement dans le Manuel du développeur AWS IoT Core.

- La stratégie comprend une section qui permet à AWS IoT Core d'obtenir, de mettre à jour et de supprimer le shadow de l'appareil du noyau. Pour autoriser la synchronisation parallèle pour les appareils clients du groupe Greengrass, spécifiez les Amazon Resource Names (ARN) cibles dans la `Resource` liste (par exemple,). `arn:aws:iot:region:account-id:thing/device-name`
- L'utilisation de variables de stratégie d'objet (`iot:Connection.Thing.*`) dans la stratégie AWS IoT pour un appareil noyau n'est pas prise en charge. Le noyau utilise le même certificat d'appareil pour établir plusieurs connexions à AWS IoT Core, mais l'ID client dans une connexion peut ne pas correspondre exactement au nom de l'objet noyau.
- Pour l'autorisation `greengrass:UpdateCoreDeploymentStatus`, la segment final de l'`Resource`ARN est l'ARN codé en URL de l'appareil du noyau.

```
{
    "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 IoTles politiques relatives aux appareils clients nécessitent généralement des autorisations similaires pour `iot:Connect`iot:Publish,`iot:Receive`, et `iot:Subscribe` les actions.
Pour permettre à un appareil client de détecter automatiquement les informations de connectivité pour les cœurs des groupes Greengrass auxquels il appartient, la AWS IoT politique d'un appareil client doit inclure l'`greengrass:Discover`action. Dans la `Resource` section, spécifiez l'ARN de l'appareil client, et non l'ARN du périphérique principal Greengrass. Par exemple :

```
{
    "Effect": "Allow",
    "Action": [
        "greengrass:Discover"
    ],
    "Resource": [
```

```
            "arn:aws:iot:region:account-id:thing/device-name"
        ]
}
```

La AWS IoT politique relative aux appareils clients ne nécessite généralement pas d'autorisations ou d'`iot:DeleteThingShadow`actions `iot:GetThingShadow``iot:UpdateThingShadow`, car le noyau de Greengrass gère les opérations de synchronisation instantanée pour les appareils clients. Dans ce cas, assurez-vous que la `Resource` section relative aux actions parallèles de la AWS IoT politique principale inclut les ARN des appareils clients.

Dans la AWS IoT console, vous pouvez consulter et modifier la politique attachée au certificat de votre noyau.

1. Dans le volet de navigation, sous Gérer, développez Tous les appareils, puis choisissez Objets.
2. Choisissez votre cœur.
3. Sur la page de configuration de votre noyau, choisissez l'onglet Certificats.
4. Dans l'onglet Certificats, choisissez votre certificat.
5. Sur la page de configuration du certificat, choisissez Stratégies, puis la stratégie.

   Si vous souhaitez modifier la politique, choisissez Modifier la version active.
6. Passez en revue la politique et ajoutez, supprimez ou modifiez des autorisations selon vos besoins.
7. Pour définir une nouvelle version de politique comme version active, sous État de la version de politique, sélectionnez Définir la version modifiée comme version active pour cette politique.
8. Choisissez Enregistrer en tant que nouvelle version.

Gestion des identités et des accès pour AWS IoT Greengrass

AWS Identity and Access Management (IAM) est un Service AWS qui aide un administrateur à contrôler en toute sécurité l’accès aux ressources AWS. Des administrateurs IAM contrôlent les personnes qui s'authentifient (sont connectées) et sont autorisées (disposent d'autorisations) à utiliser

des ressources AWS IoT Greengrass. IAM est un Service AWS que vous pouvez utiliser sans frais supplémentaires.

> **Note**
>
> Cette rubrique décrit les concepts et fonctionnalités de l'IAM. Pour plus d'informations sur les fonctionnalités IAM prises en charge parAWS IoT Greengrass, consultezthe section called “Fonctionnement d’AWS IoT Greengrass avec IAM”.

Public ciblé

Votre utilisation d'AWS Identity and Access Management (IAM) diffère selon la tâche que vous accomplissez dans AWS IoT Greengrass.

Utilisateur du service – Si vous utilisez le service AWS IoT Greengrass pour effectuer votre tâche, votre administrateur vous fournit les informations d'identification et les autorisations dont vous avez besoin. Plus vous utiliserez de fonctions AWS IoT Greengrass pour effectuer votre travail, plus vous pourriez avoir besoin d'autorisations supplémentaires. En comprenant bien la gestion des accès, vous saurez demander les autorisations appropriées à votre administrateur. Si vous ne pouvez pas accéder à une fonctionnalité dans AWS IoT Greengrass, consultez Résolution des problèmes d’identité et d’accès pour AWS IoT Greengrass.

Administrateur du service – Si vous êtes le responsable des ressources AWS IoT Greengrass de votre entreprise, vous bénéficiez probablement d'un accès total à AWS IoT Greengrass. Votre responsabilité est de déterminer AWS IoT Greengrass les fonctionnalités ainsi que les ressources auxquelles les utilisateurs de votre service doivent accéder. Vous devez ensuite soumettre les demandes à votre administrateur IAM pour modifier les autorisations des utilisateurs de votre service. Consultez les informations sur cette page pour comprendre les concepts de base d’IAM. Pour en savoir plus sur la façon dont votre entreprise peut utiliser IAM avec AWS IoT Greengrass, veuillez consulter Fonctionnement d’AWS IoT Greengrass avec IAM.

Administrateur IAM : si vous êtes un administrateur IAM, vous souhaiterez peut-être en savoir plus sur la façon d'écrire des politiques pour gérer l'accès à AWS IoT Greengrass. Pour voir des exemples de politiques AWS IoT Greengrass basées sur l'identité que vous pouvez utiliser dans IAM, veuillez consulter Exemples de stratégies basées sur l'identité pour AWS IoT Greengrass.

Authentification par des identités

L'authentification correspond au processus par lequel vous vous connectez à AWS avec vos informations d'identification. Vous devez vous authentifier (être connecté à AWS) en tant qu'Utilisateur racine d'un compte AWS, en tant qu'utilisateur IAM ou en endossant un rôle IAM.

Vous pouvez vous connecter à AWS en tant qu'identité fédérée à l'aide des informations d'identification fournies par le biais d'une source d'identité. AWS IAM Identity Center Les utilisateurs (IAM Identity Center), l'authentification de connexion unique de votre entreprise et vos informations d'identification Google ou Facebook sont des exemples d'identités fédérées. Lorsque vous vous connectez avec une identité fédérée, votre administrateur aura précédemment configuré une fédération d'identités avec des rôles IAM. Lorsque vous accédez à AWS en utilisant la fédération, vous endossez indirectement un rôle.

Selon le type d'utilisateur que vous êtes, vous pouvez vous connecter à la AWS Management Console ou au portail d'accès AWS. Pour plus d'informations sur la connexion à AWS, consultez Connexion à votre Compte AWSdans le Guide de l'utilisateurConnexion à AWS.

Si vous accédez à AWS par programmation, AWS fournit un kit de développement logiciel (SDK) et une interface de ligne de commande (CLI) pour signer cryptographiquement vos demandes en utilisant vos informations d'identification. Si vous n'utilisez pas les outils AWS, vous devez signer les requêtes vous-même. Pour plus d'informations sur l'utilisation de la méthode recommandée pour signer des demandes vous-même, consultez Signature des demandes d'API AWS dans le Guide de l'utilisateur IAM.

Quelle que soit la méthode d'authentification que vous utilisez, vous devrez peut-être fournir des informations de sécurité supplémentaires. Par exemple, AWS vous recommande d'utiliser l'authentification multifactorielle (MFA) pour améliorer la sécurité de votre compte. Pour en savoir plus, veuillez consulter Authentification multifactorielle dans le Guide de l'utilisateur AWS IAM Identity Center et Utilisation de l'authentification multifactorielle (MFA) dans l'interface AWS dans le Guide de l'utilisateur IAM.

Utilisateur root Compte AWS

Lorsque vous créez un Compte AWS, vous commencez avec une seule identité de connexion disposant d'un accès complet à tous les Services AWS et ressources du compte. Cette identité est appelée utilisateur root du Compte AWS. Vous pouvez y accéder en vous connectant à l'aide de l'adresse électronique et du mot de passe que vous avez utilisés pour créer le compte. Il est vivement recommandé de ne pas utiliser l'utilisateur root pour vos tâches quotidiennes. Protégez

vos informations d'identification d'utilisateur root et utilisez-les pour effectuer les tâches que seul l'utilisateur root peut effectuer. Pour obtenir la liste complète des tâches qui vous imposent de vous connecter en tant qu'utilisateur root, consultez Tâches nécessitant les informations d'identification de l'utilisateur root dans le Guide de l'utilisateur IAM.

Utilisateurs et groupes IAM

Un utilisateur IAM est une identité dans votre Compte AWS qui dispose d'autorisations spécifiques pour une seule personne ou application. Dans la mesure du possible, nous vous recommandons de vous appuyer sur des informations d'identification temporaires plutôt que de créer des utilisateurs IAM ayant des informations d'identification à long terme tels que les clés d'accès. Toutefois, si certains cas d'utilisation spécifiques nécessitent des informations d'identification à long terme avec les utilisateurs IAM, nous vous recommandons de faire pivoter les clés d'accès. Pour plus d'informations, consultez Rotation régulière des clés d'accès pour les cas d'utilisation nécessitant des informations d'identification dans le Guide de l'utilisateur IAM.

Un groupe IAM est une identité qui concerne un ensemble d'utilisateurs IAM. Vous ne pouvez pas vous connecter en tant que groupe. Vous pouvez utiliser les groupes pour spécifier des autorisations pour plusieurs utilisateurs à la fois. Les groupes permettent de gérer plus facilement les autorisations pour de grands ensembles d'utilisateurs. Par exemple, vous pouvez avoir un groupe nommé IAMAdmins et accorder à ce groupe les autorisations d'administrer des ressources IAM.

Les utilisateurs sont différents des rôles. Un utilisateur est associé de manière unique à une personne ou une application, alors qu'un rôle est conçu pour être endossé par tout utilisateur qui en a besoin. Les utilisateurs disposent d'informations d'identification permanentes, mais les rôles fournissent des informations d'identification temporaires. Pour en savoir plus, consultez Quand créer un utilisateur IAM (au lieu d'un rôle) dans le Guide de l'utilisateur IAM.

Rôles IAM

Un rôle IAM est une entité au sein de votre Compte AWS qui dispose d'autorisations spécifiques. Le concept ressemble à celui d'utilisateur IAM, mais le rôle IAM n'est pas associé à une personne en particulier. Vous pouvez temporairement endosser un rôle IAM dans la AWS Management Console en changeant de rôle. Vous pouvez obtenir un rôle en appelant une opération d'API AWS CLI ou AWS à l'aide d'une URL personnalisée. Pour plus d'informations sur les méthodes d'utilisation des rôles, consultez Utilisation de rôles IAM dans le Guide de l'utilisateur IAM.

Les rôles IAM avec des informations d'identification temporaires sont utiles dans les cas suivants :

- Accès utilisateur fédéré – Pour attribuer des autorisations à une identité fédérée, vous créez un rôle et définissez des autorisations pour le rôle. Quand une identité externe s'authentifie, l'identité est associée au rôle et reçoit les autorisations qui sont définies par celui-ci. Pour obtenir des informations sur les rôles pour la fédération, consultez Création d'un rôle pour un fournisseur d'identité tiers (fédération) dans le Guide de l'utilisateur IAM. Si vous utilisez IAM Identity Center, vous configurez un jeu d'autorisations. IAM Identity Center met en corrélation le jeu d'autorisations avec un rôle dans IAM afin de contrôler à quoi vos identités peuvent accéder après leur authentification. Pour plus d'informations sur les jeux d'autorisations, consultez Jeux d'autorisations dans le Guide de l'utilisateur AWS IAM Identity Center.
- Autorisations d'utilisateur IAM temporaires : un rôle ou un utilisateur IAM peut endosser un rôle IAM pour profiter temporairement d'autorisations différentes pour une tâche spécifique.
- Accès intercompte : vous pouvez utiliser un rôle IAM pour permettre à un utilisateur (principal de confiance) d'un compte différent d'accéder aux ressources de votre compte. Les rôles constituent le principal moyen d'accorder l'accès intercompte. Toutefois, certains Services AWS vous permettent d'attacher une politique directement à une ressource (au lieu d'utiliser un rôle en tant que proxy). Pour en savoir plus sur la différence entre les rôles et les politiques basées sur les ressources pour l'accès intercompte, consultez Différence entre les rôles IAM et les politiques basées sur les ressources dans le Guide de l'utilisateur IAM.
- Accès interservices : certains Services AWS utilisent des fonctions dans d'autres Services AWS. Par exemple, lorsque vous effectuez un appel dans un service, il est courant que ce service exécute des applications dans Amazon EC2 ou stocke des objets dans Amazon S3. Un service peut le faire en utilisant les autorisations d'appel du principal, une fonction du service ou un rôle lié au service.
  - Forward access sessions (FAS) – Lorsque vous utilisez un utilisateur ou un rôle IAM pour effectuer des actions dans AWS, vous êtes considéré comme un principal. Lorsque vous utilisez certains services, vous pouvez effectuer une action qui déclenche une autre action dans un autre service. FAS utilise les autorisations du principal appelant Service AWS, combinées à la demande Service AWS pour effectuer des demandes aux services en aval. Les demandes de FAS ne sont effectuées que lorsqu'un service reçoit une demande qui nécessite des interactions avec d'autres Services AWS ou des ressources pour être traitée. Dans ce cas, vous devez disposer d'autorisations nécessaires pour effectuer les deux actions. Pour plus de détails sur la politique relative à la transmission de demandes FAS, consultez Sessions de transmission d'accès.
  - Fonction du service : il s'agit d'un rôle IAM attribué à un service afin de réaliser des actions en votre nom. Un administrateur IAM peut créer, modifier et supprimer une fonction du service

à partir d'IAM. Pour plus d'informations, consultez Création d'un rôle pour la délégation d'autorisations à un Service AWS dans le Guide de l'utilisateur IAM.

- Rôle lié au service – Un rôle lié au service est un type de fonction du service lié à un Service AWS. Le service peut endosser le rôle afin d'effectuer une action en votre nom. Les rôles liés à un service s'affichent dans votre Compte AWS et sont détenus par le service. Un administrateur IAM peut consulter, mais ne peut pas modifier, les autorisations concernant les rôles liés à un service.

- Applications s'exécutant sur Amazon EC2 : vous pouvez utiliser un rôle IAM pour gérer des informations d'identification temporaires pour les applications s'exécutant sur une instance EC2 et effectuant des demandes d'API AWS CLI ou AWS. Cette solution est préférable au stockage des clés d'accès au sein de l'instance EC2. Pour attribuer un rôle AWS à une instance EC2 et le rendre disponible à toutes les applications associées, vous pouvez créer un profil d'instance attaché à l'instance. Un profil d'instance contient le rôle et permet aux programmes qui s'exécutent sur l'instance EC2 d'obtenir des informations d'identification temporaires. Pour plus d'informations, consultez Utilisation d'un rôle IAM pour accorder des autorisations à des applications s'exécutant sur des instances Amazon EC2 dans le Guide de l'utilisateur IAM.

Pour savoir dans quel cas utiliser des rôles ou des utilisateurs IAM, consultez Quand créer un rôle IAM (au lieu d'un utilisateur) dans le Guide de l'utilisateur IAM.

Gestion des accès à l'aide de politiques

Vous contrôlez les accès dans AWS en créant des politiques et en les attachant à des identités AWS ou à des ressources. Une politique est un objet dans AWS qui, lorsqu'il est associé à une identité ou à une ressource, définit les autorisations de ces dernières. AWS évalue ces politiques lorsqu'un principal (utilisateur, utilisateur racine ou séance de rôle) envoie une demande. Les autorisations dans les politiques déterminent si la demande est autorisée ou refusée. La plupart des politiques sont stockées dans AWS en tant que documents JSON. Pour plus d'informations sur la structure et le contenu des documents de politique JSON, consultez Présentation des politiques JSON dans le Guide de l'utilisateur IAM.

Les administrateurs peuvent utiliser les politiques JSON AWS pour spécifier qui a accès à quoi. C'est-à-dire, quel principal peut effectuer des actions sur quelles ressources et dans quelles conditions.

Par défaut, les utilisateurs et les rôles ne disposent d'aucune autorisation. Pour octroyer aux utilisateurs des autorisations d'effectuer des actions sur les ressources dont ils ont besoin, un

administrateur IAM peut créer des politiques IAM. L'administrateur peut ensuite ajouter les politiques IAM aux rôles et les utilisateurs peuvent assumer les rôles.

Les politiques IAM définissent les autorisations d'une action, quelle que soit la méthode que vous utilisez pour exécuter l'opération. Par exemple, supposons que vous disposiez d'une politique qui autorise l'action `iam:GetRole`. Un utilisateur avec cette politique peut obtenir des informations utilisateur à partir de la AWS Management Console, de la AWS CLI ou de l'API AWS.

Politiques basées sur l'identité

Les politiques basées sur l'identité sont des documents de politique d'autorisations JSON que vous pouvez attacher à une identité telle qu'un utilisateur, un groupe d'utilisateurs ou un rôle IAM. Ces politiques contrôlent quel type d'actions des utilisateurs et des rôles peuvent exécuter, sur quelles ressources et dans quelles conditions. Pour découvrir comment créer une politique basée sur l'identité, consultez Création de politiques IAM dans le Guide de l'utilisateur IAM.

Les politiques basées sur l'identité peuvent être classées comme des politiques en ligne ou des politiques gérées. Les politiques en ligne sont intégrées directement à un utilisateur, groupe ou rôle. Les politiques gérées sont des politiques autonomes que vous pouvez attacher à plusieurs utilisateurs, groupes et rôles dans votre Compte AWS. Les politiques gérées incluent les politiques gérées par AWS et les politiques gérées par le client. Pour découvrir comment choisir entre une politique gérée et une politique en ligne, consultez Choix entre les politiques gérées et les politiques en ligne dans le Guide de l'utilisateur IAM.

politiques basées sur les ressources

Les politiques basées sur les ressources sont des documents de politique JSON que vous attachez à une ressource. Des politiques basées sur les ressources sont, par exemple, les politiques de confiance de rôle IAM et des politiques de compartiment Amazon S3. Dans les services qui sont compatibles avec les politiques basées sur les ressources, les administrateurs de service peuvent les utiliser pour contrôler l'accès à une ressource spécifique. Pour la ressource dans laquelle se trouve la politique, cette dernière définit quel type d'actions un principal spécifié peut effectuer sur cette ressource et dans quelles conditions. Vous devez spécifier un principal dans une politique basée sur les ressources. Les principaux peuvent inclure des comptes, des utilisateurs, des rôles, des utilisateurs fédérés ou des Services AWS.

Les politiques basées sur les ressources sont des politiques en ligne situées dans ce service. Vous ne pouvez pas utiliser les politiques gérées AWS depuis IAM dans une politique basée sur une ressource.

Listes de contrôle d’accès (ACL)

Les listes de contrôle d’accès (ACL) vérifie quels principaux (membres de compte, utilisateurs ou rôles) ont l’autorisation d’accéder à une ressource. Les listes de contrôle d’accès sont similaires aux politiques basées sur les ressources, bien qu’elles n’utilisent pas le format de document de politique JSON.

Amazon S3, AWS WAF et Amazon VPC sont des exemples de services prenant en charge les ACL. Pour en savoir plus sur les listes de contrôle d’accès, consultez Présentation des listes de contrôle d’accès (ACL) dans le Guide du développeur Amazon Simple Storage Service.

Autres types de politique

AWS prend en charge d’autres types de politiques moins courantes. Ces types de politiques peuvent définir le nombre maximum d’autorisations qui vous sont accordées par des types de politiques plus courants.

- Limite d’autorisations : une limite d’autorisations est une fonction avancée dans laquelle vous définissez le nombre maximal d’autorisations qu’une politique basée sur l’identité peut accorder à une entité IAM (utilisateur ou rôle IAM). Vous pouvez définir une limite d’autorisations pour une entité. Les autorisations qui en résultent représentent la combinaison des politiques basées sur l’identité d’une entité et de ses limites d’autorisation. Les politiques basées sur les ressources qui spécifient l’utilisateur ou le rôle dans le champ `Principal` ne sont pas limitées par les limites d’autorisations. Un refus explicite dans l’une de ces politiques remplace l’autorisation. Pour plus d’informations sur les limites d’autorisations, consultez Limites d’autorisations pour des entités IAM dans le Guide de l’utilisateur IAM.
- Politiques de contrôle des services (SCP) - les SCP sont des politiques JSON qui spécifient le nombre maximal d’autorisations pour une organisation ou une unité d’organisation (OU) dans AWS Organizations. AWS Organizations est un service qui vous permet de regrouper et de gérer de façon centralisée plusieurs Comptes AWS détenus par votre entreprise. Si vous activez toutes les fonctions d’une organisation, vous pouvez appliquer les politiques de contrôle des services (SCP) à l’un ou à l’ensemble de vos comptes. La SCP limite les autorisations pour les entités dans les comptes membres, y compris dans chaque Utilisateur racine d'un compte AWS. Pour plus d’informations sur les organisations et les SCP, consultez Fonctionnement des SCP dans le Guide de l’utilisateur AWS Organizations.
- politiques de séance : les politiques de séance sont des politiques avancées que vous utilisez en tant que paramètre lorsque vous créez par programmation une séance temporaire pour un rôle ou un utilisateur fédéré. Les autorisations de la séance obtenue sont une combinaison des politiques

basées sur l'identité de l'utilisateur ou du rôle et des politiques de séance. Les autorisations peuvent également provenir d'une politique basée sur les ressources. Un refus explicite dans l'une de ces politiques remplace l'autorisation. Pour plus d'informations, consultez Politiques de séance dans le Guide de l'utilisateur IAM.

Plusieurs types de politique

Lorsque plusieurs types de politiques s'appliquent à la requête, les autorisations obtenues sont plus compliquées à comprendre. Pour découvrir la façon dont AWS détermine s'il convient d'autoriser une demande en présence de plusieurs types de politiques, veuillez consulter Logique d'évaluation de politiques dans le Guide de l'utilisateur IAM.

Consultez aussi

- the section called "Fonctionnement d'AWS IoT Greengrass avec IAM"
- the section called "Exemples de politiques basées sur l'identité"
- the section called "Résolution des problèmes d'identité et d'accès"

Fonctionnement d'AWS IoT Greengrass avec IAM

Avant d'utiliser IAM pour gérer l'accèsAWS IoT Greengrass, vous devez comprendre les fonctionnalités IAM que vous pouvez utiliser avec. AWS IoT Greengrass

Fonctionnalité IAM	Prise en charge par Greengrass ?
Stratégies basées sur l'identité avec autorisations au niveau des ressources	Oui
Politiques basées sur les ressources	Non
Listes de contrôle d'accès (ACL)	Non
Autorisation basée sur les balises	Oui
Informations d'identification temporaires	Oui
Rôles liés à un service	Non

Fonctionnalité IAM	Prise en charge par Greengrass ?
Fonctions du service	Oui

Pour obtenir une vue d'ensemble de la façon dont les autres AWS services fonctionnent avec IAM, consultez la section AWSServices compatibles avec IAM dans le Guide de l'utilisateur d'IAM.

Politiques basées sur l’identité pour AWS IoT Greengrass

Avec les politiques basées sur l'identité IAM, vous pouvez spécifier les actions et les ressources autorisées ou refusées, ainsi que les conditions dans lesquelles les actions sont autorisées ou refusées. AWS IoT Greengrass prend en charge des actions, des ressources et des clés de condition spécifiques. Pour en savoir plus sur tous les éléments que vous utilisez dans une politique, consultez la référence des éléments de stratégie IAM JSON dans le guide de l'utilisateur IAM.

Actions

Les administrateurs peuvent utiliser les politiques JSON AWS pour spécifier qui a accès à quoi. C’est-à-dire, quel principal peut effectuer des actions sur quelles ressources et dans quelles conditions.

L’élément `Action` d’une politique JSON décrit les actions que vous pouvez utiliser pour autoriser ou refuser l’accès à une politique. Les actions de politique possèdent généralement le même nom que l’opération d’API AWS associée. Il existe quelques exceptions, telles que les actions avec autorisations uniquement qui n’ont pas d’opération API correspondante. Certaines opérations nécessitent également plusieurs actions dans une politique. Ces actions supplémentaires sont nommées actions dépendantes.

Intégration d’actions dans une stratégie afin d’accorder l’autorisation d’exécuter les opérations associées.

Les actions de stratégie pour AWS IoT Greengrass utilisent le préfixe `greengrass:` avant l'action. Par exemple, pour autoriser une personne à utiliser l'opération `ListGroups` API pour répertorier les groupes de son choixCompte AWS, vous devez inclure l'`greengrass:ListGroups`action dans sa politique. Les déclarations de politique doivent inclure un élément `Action` ou `NotAction`. AWS IoT Greengrass définit son propre ensemble d'actions qui décrivent les tâches que vous pouvez effectuer avec ce service.

Pour spécifier plusieurs actions dans une seule instruction, listez-les entre crochets (`[]`) et séparez-les par des virgules, comme suit :

```
"Action": [
    "greengrass:action1",
    "greengrass:action2",
    "greengrass:action3"
]
```

Vous pouvez utiliser des caractères génériques (*) pour spécifier plusieurs actions. Par exemple, pour spécifier toutes les actions qui commencent par le mot `List`, incluez l'action suivante :

```
"Action": "greengrass:List*"
```

> **Note**
>
> Nous vous recommandons d'éviter d'utiliser des caractères génériques pour spécifier toutes les actions disponibles pour un service. La bonne pratique consiste à applique le principe du moindre privilège et à accorder des autorisations de portée limitée dans une stratégie. Pour plus d'informations, consultez the section called "Accorder le moins d'autorisations possibles".

Pour obtenir la liste complète des AWS IoT Greengrass actions, consultez la section Actions définies par AWS IoT Greengrass dans le guide de l'utilisateur IAM.

Ressources

Les administrateurs peuvent utiliser les politiques JSON AWS pour spécifier qui a accès à quoi. C'est-à-dire, quel principal peut effectuer des actions sur quelles ressources et dans quelles conditions.

L'élément de politique JSON `Resource` indique le ou les objets pour lesquels l'action s'applique. Les instructions doivent inclure un élément `Resource` ou `NotResource`. Il est recommandé de définir une ressource à l'aide de son Amazon Resource Name (ARN). Vous pouvez le faire pour des actions qui prennent en charge un type de ressource spécifique, connu sous la dénomination autorisations de niveau ressource.

Pour les actions qui ne sont pas compatibles avec les autorisations de niveau ressource, telles que les opérations de liste, utilisez un caractère générique (*) afin d'indiquer que l'instruction s'applique à toutes les ressources.

```
"Resource": "*"
```

Le tableau suivant contient les ARN de ressource AWS IoT Greengrass qui peuvent être utilisés dans l'élément `Resource` d'une déclaration de stratégie. Pour un mappage des autorisations prises en charge au niveau des ressources pour les AWS IoT Greengrass actions, consultez la section Actions définies par AWS IoT Greengrass dans le guide de l'utilisateur IAM.

Ressource	ARN
Group	`arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/groups/${GroupId}`
GroupVersion	`arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/groups/${GroupId}/versions/${VersionId}`
CertificateAuthority	`arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/groups/${GroupId}/certificateauthorities/${CertificateAuthorityId}`
Deployment	`arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/groups/${GroupId}/deployments/${DeploymentId}`
BulkDeployment	`arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/bulk/deployments/${BulkDeploymentId}`
ConnectorDefinition	`arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/connectors/${ConnectorDefinitionId}`
ConnectorDefinitionVersion	`arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/connectors/${ConnectorDefinitionId}/versions/${VersionId}`
CoreDefinition	`arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/cores/${CoreDefinitionId}`

Ressource	ARN
CoreDefinitionVersion	arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/cores/${CoreDefinitionId}/versions/${VersionId}
DeviceDefinition	arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/devices/${DeviceDefinitionId}
DeviceDefinitionVersion	arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/devices/${DeviceDefinitionId}/versions/${VersionId}
FunctionDefinition	arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/functions/${FunctionDefinitionId}
FunctionDefinitionVersion	arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/functions/${FunctionDefinitionId}/versions/${VersionId}
LoggerDefinition	arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/loggers/${LoggerDefinitionId}
LoggerDefinitionVersion	arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/loggers/${LoggerDefinitionId}/versions/${VersionId}
ResourceDefinition	arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/resources/${ResourceDefinitionId}
ResourceDefinitionVersion	arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/resources/${ResourceDefinitionId}/versions/${VersionId}

Ressource	ARN
SubscriptionDefinition	`arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/subscriptions/${SubscriptionDefinitionId}`
SubscriptionDefinitionVersion	`arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/subscriptions/${SubscriptionDefinitionId}/versions/${VersionId}`
ConnectivityInfo	`arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/things/${ThingName}/connectivityInfo`

L'exemple d'`Resource`élément suivant indique l'ARN d'un groupe de la région de l'ouest des États-Unis (Oregon) dans le Compte AWS `123456789012` :

```
"Resource": "arn:aws:greengrass:us-west-2:123456789012:/greengrass/groups/
a1b2c3d4-5678-90ab-cdef-EXAMPLE11111
```

Ou, pour spécifier tous les groupes appartenant Compte AWS à un groupe spécifiqueRégion AWS, utilisez le caractère générique à la place de l'identifiant du groupe :

```
"Resource": "arn:aws:greengrass:us-west-2:123456789012:/greengrass/groups/*"
```

Certaines actions AWS IoT Greengrass (par exemple, certaines opérations de liste) ne peuvent pas être effectuées sur une ressource spécifique. Dans ce cas, vous devez utiliser le caractère générique seul.

```
"Resource": "*"
```

Pour spécifier plusieurs ARN de ressources dans une instruction, listez-les entre crochets (`[]`) et séparez-les par des virgules, comme suit :

```
"Resource": [
    "resource-arn1",
    "resource-arn2",
    "resource-arn3"
]
```

Pour de plus amples informations sur les formats d'ARN, veuillez consulter Noms ARN (Amazon Resource Name) et espaces de noms du service AWS dans le Référence générale d'Amazon Web Services.

Clés de condition

Les administrateurs peuvent utiliser les politiques JSON AWS pour spécifier qui a accès à quoi. C'est-à-dire, quel principal peut effectuer des actions sur quelles ressources et dans quelles conditions.

L'élément `Condition` (ou le bloc `Condition`) vous permet de spécifier des conditions lorsqu'une instruction est appliquée. L'élément `Condition` est facultatif. Vous pouvez créer des expressions conditionnelles qui utilisent des opérateurs de condition, tels que les signes égal ou inférieur à, pour faire correspondre la condition de la politique aux valeurs de la demande.

Si vous spécifiez plusieurs éléments `Condition` dans une instruction, ou plusieurs clés dans un seul élément `Condition`, AWS les évalue à l'aide d'une opération `AND` logique. Si vous spécifiez plusieurs valeurs pour une seule clé de condition, AWS évalue la condition à l'aide d'une opération `OR` logique. Toutes les conditions doivent être remplies avant que les autorisations associées à l'instruction ne soient accordées.

Vous pouvez aussi utiliser des variables d'espace réservé quand vous spécifiez des conditions. Par exemple, vous pouvez accorder à un utilisateur IAM l'autorisation d'accéder à une ressource uniquement si elle est balisée avec son nom d'utilisateur IAM. Pour plus d'informations, consultez Éléments d'une politique IAM : variables et identifications dans le Guide de l'utilisateur IAM.

AWS prend en charge les clés de condition globales et les clés de condition spécifiques à un service. Pour afficher toutes les clés de condition globales AWS, consultez Clés de contexte de condition globale AWS dans le Guide de l'utilisateur IAM.

AWS IoT Greengrass prend en charge les clés de condition globales suivantes.

Clé	Description
`aws:CurrentTime`	Filtre l'accès en vérifiant les conditions de date/heure pour la date et l'heure actuelles.
`aws:EpochTime`	Filtre l'accès en vérifiant les conditions de date/heure pour la date et l'heure actuelles au format epoch ou Unix.

Clé	Description
`aws:MultiFactorAuthAge`	Filtre l'accès en vérifiant depuis combien de temps (en secondes) les informations d'identification de sécurité validées par l'authentification MFA (Multi-Factor Authentication) dans la demande ont été émises via MFA.
`aws:MultiFactorAuthPresent`	Filtre l'accès en vérifiant si l'authentification MFA (Multi-Factor Authentication) a été utilisée pour valider les informations d'identification de sécurité temporaires qui ont effectué cette demande.
`aws:RequestTag/${TagKey}`	Filtre les demandes de création en fonction de l'ensemble de valeurs autorisé pour chacune des balises obligatoires.
`aws:ResourceTag/${TagKey}`	Filtre les actions en fonction de la valeur de balise associée à la ressource.
`aws:SecureTransport`	Filtre l'accès en vérifiant si la demande a été envoyée à l'aide de SSL.
`aws:TagKeys`	Filtre les demandes de création en fonction de la présence de balises obligatoires dans la demande.
`aws:UserAgent`	Filtre l'accès par l'application cliente du demandeur.

Pour plus d'informations, consultez Clés de contexte de condition globales AWS dans le Guide de l'utilisateur IAM.

Exemples

Pour voir des exemples de politiques AWS IoT Greengrass basées sur l'identité, consultez the section called “Exemples de politiques basées sur l'identité”.

Stratégies basées sur des ressources pour AWS IoT Greengrass

AWS IoT Greengrass ne prend pas en charge les stratégies basées sur les ressources.

Listes de contrôle d’accès (ACL)

AWS IoT Greengrass ne prend pas en charge les listes de contrôle d’accès.

Autorisation basée sur les balises AWS IoT Greengrass

Vous pouvez attacher des balises aux ressources AWS IoT Greengrass prise en charge, ou transmettre des balises dans une demande à AWS IoT Greengrass. Pour contrôler l'accès basé sur des balises, vous devez fournir les informations de balise dans l'élément Condition d'une stratégie utilisant les clés de condition `aws:ResourceTag/${TagKey}`, `aws:RequestTag/${TagKey}` ou `aws:TagKeys`. Pour plus d’informations, consultez Balisage de vos ressources Greengrass.

Rôles IAM pour AWS IoT Greengrass

Un rôle IAM est une entité au sein de votre Compte AWS qui dispose d'autorisations spécifiques.

Utilisation des informations d’identification temporaires avec AWS IoT Greengrass

Les informations d'identification temporaires sont utilisées pour se connecter à la fédération, pour assumer un rôle IAM ou pour assumer un rôle entre comptes. Vous obtenez des informations d'identification de sécurité temporaires en appelant des opérations d'AWS STSAPI telles que AssumeRoleou GetFederationToken.

Sur le noyau de Greengrass, les informations d'identification temporaires pour le rôle de groupe sont mises à la disposition des fonctions et connecteurs Lambda définis par l'utilisateur. Si vos fonctions Lambda utilisent le AWS SDK, vous n'avez pas besoin d'ajouter de logique pour obtenir les informations d'identification, car le AWS SDK le fait pour vous.

Rôles liés à un service

AWS IoT Greengrass ne prend pas en charge les rôles liés à un service.

Fonctions du service

Cette fonction permet à un service d’endosser une fonction du service en votre nom. Ce rôle autorise le service à accéder à des ressources d’autres services pour effectuer une action en votre nom. Les fonctions du service s’affichent dans votre compte IAM et sont la propriété du compte. Cela signifie qu’un administrateur IAM peut modifier les autorisations associées à ce rôle. Toutefois, une telle action peut perturber le bon fonctionnement du service.

AWS IoT Greengrass utilise un rôle de service pour accéder à certaines de vos ressources AWS en votre nom. Pour plus d'informations, consultez the section called "Rôle de service Greengrass".

Choix d'un rôle IAM dans la console AWS IoT Greengrass

Dans la AWS IoT Greengrass console, vous devrez peut-être choisir un rôle de service Greengrass ou un rôle de groupe Greengrass dans la liste des rôles IAM de votre compte.

- Le rôle de service Greengrass permet à AWS IoT Greengrass d'accéder à vos ressources AWS dans d'autres services en votre nom. En règle générale, vous n'avez pas besoin de choisir le rôle de service car la console peut le créer et le configurer pour vous. Pour plus d'informations, consultez the section called "Rôle de service Greengrass".
- Le rôle de groupe Greengrass est utilisé pour permettre aux fonctions et connecteurs Greengrass Lambda du groupe d'accéder à vos ressources. AWS Il peut également autoriser l'AWS IoT Greengrassexportation de flux vers des AWS services et la rédaction de CloudWatch journaux. Pour plus d'informations, consultez the section called "Rôle de groupe Greengrass".

Rôle de service Greengrass

Votre rôle de service Greengrass est unAWS Identity and Access ManagementRôle de service (IAM) qui autorise àAWS IoT Greengrasspour accéder aux ressources deAWSles services en votre nom. Ceci permet à AWS IoT Greengrass d'effectuer des tâches essentielles, telles que la récupération de vos fonctions AWS Lambda et la gestion des shadows AWS IoT.

Pour autoriser àAWS IoT Greengrasspour accéder à vos ressources, le rôle de service Greengrass doit être associé à votreCompte AWSet spécifiezAWS IoT Greengrassen tant qu'entité de confiance. Votre rôle doit inclure les AWSGreengrassResourceAccessRolePolicyune stratégie gérée ou une stratégie personnalisée qui définit des autorisations équivalentes pour leAWS IoT Greengrassles fonctionnalités que vous utilisez. Cette politique est maintenue parAWSet définit l'ensemble des autorisations quiAWS IoT Greengrasspour accéder à votreAWSAWS.

Vous pouvez réutiliser le même rôle de service Greengrass dans tous lesRégion AWS, mais vous devez l'associer à votre compte dans chaqueRégion AWSoù vous utilisezAWS IoT Greengrass. Le déploiement du groupe échoue si le rôle de service n'existe pas dans leCompte AWSet région.

Les sections suivantes décrivent comment créer et gérer le rôle de service Greengrass dans AWS Management Console ou l'AWS CLI.

- Gestion du rôle de service (console)

- Gestion du rôle de service (interface de ligne de commande)

> **Note**
>
> Outre le rôle de service qui autorise l'accès de niveau de service, vous pouvez attribuer unrôle de groupeà unAWS IoT Greengrass. Le rôle de groupe est un rôle IAM distinct qui contrôle la façon dont les fonctions Greengrass Lambda et les connecteurs du groupe peuvent accéder.AWSServices .

Gestion du rôle de service Greengrass (console)

La console AWS IoT facilite la gestion de votre rôle de service Greengrass. Par exemple, lorsque vous créez ou déployez un groupe Greengrass, la console vérifie si votreCompte AWSest associée à un rôle de service Greengrass dans leRégion AWSactuellement sélectionnée dans la console. Si ce n'est pas le cas, la console peut créer et configurer un rôle de service pour vous. Pour plus d'informations, consultez the section called “Créer le rôle de service Greengrass”.

Vous pouvez utiliser le pluginAWS IoTconsole pour les tâches de gestion de rôles suivantes :

- Rechercher votre rôle de service Greengrass
- Créer le rôle de service Greengrass
- Modifier le rôle de service Greengrass
- Détacher le rôle de service Greengrass

> **Note**
>
> L'utilisateur qui est connecté à la console doit disposer d'autorisations pour afficher, créer ou modifier le rôle de service.

Rechercher votre rôle de service Greengrass (console)

Procédez comme suit pour rechercher le rôle de service quiAWS IoT Greengrassutilise dans la version actuelleRégion AWS.

1. À partir desAWS IoTconsolevolet de navigation, choisissezParamètres.
2. Faites défiler l'écran jusqu'à la section Greengrass service role (Rôle de service Greengrass) pour afficher votre rôle de service et ses stratégies.

   Si vous ne voyez pas de rôle de service, vous pouvez laisser la console en créer ou en configurer un pour vous. Pour plus d'informations, consultez Créer le rôle de service Greengrass.

Créer le rôle de service Greengrass (console)

La console peut créer et configurer un rôle de service Greengrass par défaut pour vous. Ce rôle a les propriétés suivantes :

Propriété	Valeur
Nom	`Greengrass_ServiceRole`
Entité de confiance	`AWS service: greengrass`
Politique	AWSGreengrassResourceAccessRolePolicy

> **Note**
>
> Si le script de configuration de l'appareil Greengrass crée le rôle de service, le nom du rôle est `GreengrassServiceRole_`*`random-string`*.

Lorsque vous créez ou déployez un groupe Greengrass à partir duAWS IoT, la console vérifie si un rôle de service Greengrass est associé à votreCompte AWSdans leRégion AWSactuellement sélectionnée dans la console. Si ce n'est pas le cas, la console vous invite à autoriser àAWS IoT Greengrassà lire et à écrire dansAWSles services en votre nom.

Si vous accordez l'autorisation, la console vérifie si un rôle nomméGreengrass_ServiceRoleexiste dans votreCompte AWS.

- Si le rôle existe, la console attache le rôle de service à votreCompte AWSdans les actuelsRégion AWS.

- Si le rôle n'existe pas, la console crée un rôle de service Greengrass par défaut et l'attache à votreCompte AWSdans les actuelsRégion AWS.

> **Note**
>
> Si vous souhaitez créer un rôle de service avec des stratégies de rôle personnalisées, utilisez la console IAM pour créer ou modifier le rôle. Pour de plus amples informations, veuillez consulterCréation d'un rôle pour la délégation d'autorisations à unAWSserviceouModification d'un rôledans leIAM User Guide. Assurez-vous que le rôle accorde des autorisations équivalentes à la stratégie `AWSGreengrassResourceAccessRolePolicy` gérée pour les fonctions et les ressources que vous utilisez. Nous vous recommandons également d'inclure les`aws:SourceArn`et`aws:SourceAccount`clés de contexte de condition globale dans votre stratégie de confiance pour empêcher ledéputé confusproblème de sécurité. Les clés contextuelles de condition limitent l'accès afin d'autoriser uniquement les demandes provenant du compte spécifié et de l'espace de travail Greengrass. Pour plus d'informations sur le problème du député confus, veuillez consulterPrévention du problème de l'adjoint confus entre services.
> Si vous créez un rôle de service, revenez à laAWS IoTconsole et attachez le rôle au groupe. Vous pouvez effectuer cette opération sous Rôle de service Greengrass sur la page Paramètres du groupe.

Modifier le rôle de service Greengrass (console)

Utilisez la procédure suivante pour choisir un rôle de service Greengrass différent à attacher à votreCompte AWSdans leRégion AWSactuellement sélectionnée dans la console.

1. À partir desAWS IoTconsolevolet de navigation, choisissezParamètres.
2. UnderRôle de service Greengrass, choisissezChanger de rôle.

   LeRôle de service de mise à jours'ouvre et affiche les rôles IAM dans votreCompte AWSqui définissentAWS IoT Greengrassen tant qu'entité de confiance.
3. Choisissez le rôle de service Greengrass à associer.
4. ChoisissezAttachement d'.

> **Note**
>
> Pour permettre à la console de créer un rôle de service Greengrass par défaut pour vous, choisissez Créer un rôle pour moi au lieu de choisir un rôle dans la liste. LeCréer un rôle pour moile lien n'apparaît pas si un rôle nomméGreengrass_ServiceRoleest dans votreCompte AWS.

Détacher le rôle de service Greengrass (console)

Utilisez la procédure suivante pour détacher le rôle de service Greengrass de votreCompte AWSdans leRégion AWSactuellement sélectionnée dans la console. Cela révoque les autorisations pourAWS IoT Greengrasspour accéder àAWSservices dans le courantRégion AWS.

> **Important**
>
> Le détachement du rôle de service peut interrompre les opérations actives.

1. À partir desAWS IoTconsolevolet de navigation, choisissezParamètres.
2. UnderRôle de service Greengrass, choisissezDétachez un rôle.
3. Dans la boîte de dialogue de confirmation, choisissez Détacher.

> **Note**
>
> Si vous n'avez plus besoin du rôle, vous pouvez le supprimer dans la console IAM. Pour plus d'informations, consultez Suppression de rôles ou de profils d'instance dans le guide de l'utilisateur IAM.
> D'autres rôles peuvent permettre à AWS IoT Greengrass d'accéder à vos ressources. Pour trouver tous les rôles qui autorisentAWS IoT Greengrasspour assumer les autorisations en votre nom, dans la console IAM, sur leRôles, recherchez les rôles qui incluentAWSservice : herbe vertedans leEntités de confiancecolumn.

Gestion du rôle de service Greengrass (interface de ligne de commande)

Dans les procédures suivantes, nous supposons que lesAWS CLIest installé et configuré pour utiliser votreCompte AWSID. Pour de plus amples informations, veuillez consulterInstallation deAWSInterface de ligne de commandeetConfiguration deAWS CLIdans leAWS Command Line InterfaceGuide de l'utilisateur.

Vous pouvez utiliser l'AWS CLI pour les tâches de gestion de rôles suivantes :

- Obtenir votre rôle de service Greengrass
- Créer le rôle de service Greengrass
- Supprimer le rôle de service Greengrass

Obtenir le rôle de service Greengrass (interface de ligne de commande)

Utilisez la procédure suivante pour déterminer si un rôle de service Greengrass est associé à votreCompte AWSdans unRégion AWS.

- Obtenez le rôle de service. Remplacez*région*avec vos recettesRégion AWS(par exemple,`us-west-2`).

  ```
  aws Greengrass get-service-role-for-account --region region
  ```

  Si un rôle de service Greengrass est déjà associé à votre compte, les métadonnées de rôle suivantes sont renvoyées.

  ```
  {
    "AssociatedAt": "timestamp",
    "RoleArn": "arn:aws:iam::account-id:role/path/role-name"
  }
  ```

  Si aucune métadonnée de rôle n'est renvoyée, vous devez créer le rôle de service (s'il n'existe pas) et l'associer à votre compte dans leRégion AWS.

Créer le rôle de service Greengrass (interface de ligne de commande)

Procédez comme suit pour créer un rôle et l'associer à votreCompte AWS.

Pour créer le rôle de service à l'aide d'IAM

1. Créez le rôle avec une stratégie d'approbation permettant à AWS IoT Greengrass d'assumer le rôle. Cet exemple crée un rôle nommé `Greengrass_ServiceRole`, mais vous pouvez utiliser un autre nom. Nous vous recommandons également d'inclure les`aws:SourceArn`et`aws:SourceAccount`clés de contexte de condition globale dans votre stratégie de confiance pour empêcher ledéputé confusproblème de sécurité. Les clés contextuelles de condition limitent l'accès afin d'autoriser uniquement les demandes provenant du compte spécifié et de l'espace de travail Greengrass. Pour plus d'informations sur le problème du député confus, veuillez consulterPrévention du problème de l'adjoint confus entre services.

   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. Copiez l'ARN de rôle du rôle des métadonnées dans la sortie. Vous utilisez l'ARN pour associer le rôle à votre compte.
3. Attachez la stratégie `AWSGreengrassResourceAccessRolePolicy` au rôle.

```
aws iam attach-role-policy --role-name Greengrass_ServiceRole --policy-arn
 arn:aws:iam::aws:policy/service-role/AWSGreengrassResourceAccessRolePolicy
```

Pour associer le rôle de service à votreCompte AWS

- Associez le rôle à votre compte. Remplacez*role-arn*avec le nom ARN du rôle de service et*région*avec vos recettesRégion AWS(par exemple,`us-west-2`).

```
aws greengrass associate-service-role-to-account --role-arn role-arn --
region region
```

En cas de réussite, la réponse suivante est renvoyée.

```
{
  "AssociatedAt": "timestamp"
}
```

Supprimer le rôle de service Greengrass (interface de ligne de commande)

Procédez comme suit pour dissocier le rôle de service Greengrass de votreCompte AWS.

- Dissociez le rôle de service de votre compte. Remplacez*région*avec vos recettesRégion AWS(par exemple,`us-west-2`).

```
aws greengrass disassociate-service-role-from-account --region region
```

En cas de réussite, la réponse suivante est renvoyée.

```
{
  "DisassociatedAt": "timestamp"
}
```

> **Note**
>
> Vous devez supprimer le rôle de service si vous ne l'utilisez dans aucunRégion AWS. Utilisez d'abord delete-role-policy pour détacher la stratégie gérée du rôle, puis utilisez `AWSGreengrassResourceAccessRolePolicy`delete-role pour supprimer le rôle. Pour plus d'informations, consultez Suppression de rôles ou de profils d'instance dans le guide de l'utilisateur IAM.

Consulter aussi

- Création d'un rôle pour la délégation d'autorisations à unAWSservicedans leIAM User Guide
- Modification d'un rôledans leIAM User Guide
- Suppression de rôles ou de profils d'instancedans leIAM User Guide
- AWS IoT GreengrassCommandes dans leAWS CLIRéférence des commandes
  - associate-service-role-to-compte
  - disassociate-service-role-from-compte
  - get-service-role-for-compte
- Commandes IAM dans leAWS CLIRéférence des commandes
  - attach-role-policy
  - create-role
  - delete-role
  - delete-role-policy

Rôle de groupe Greengrass

Le rôle de groupe Greengrass est un rôle IAM qui autorise le code s'exécutant sur un noyau Greengrass à accéder à votreAWSAWS. Vous créez le rôle et gérez les autorisations dansAWS Identity and Access Management(IAM) et attachez le rôle à votre groupe Greengrass. Un groupe Greengrass possède un rôle de groupe. Pour ajouter ou modifier des autorisations, vous pouvez attacher un rôle différent ou modifier les stratégies IAM associées au rôle.

Le rôle doit définir AWS IoT Greengrass en tant qu'entité approuvée. Selon votre analyse de rentabilité, le rôle de groupe peut contenir des stratégies IAM qui définissent :

- Des autorisations pour les utilisateursFonctions Lambdapour accéder àAWSServices .
- Des autorisations permettant aux connecteurs d'accéder aux services AWS.
- Autorisations d'gestionnaire de fluxpour exporter des flux versAWS IoT Analyticset Kinesis Data Streams.
- Des autorisations permettant la journalisation CloudWatch .

Les sections suivantes décrivent comment attacher ou détacher un rôle de groupe Greengrass dans l'AWS Management Console ou l'AWS CLI.

- Gérer le rôle de groupe (console)
- Gérer le rôle de groupe (interface de ligne de commande)

> **Note**
>
> En plus du rôle de groupe qui autorise l'accès à partir du noyau Greengrass, vous pouvez affecter un rôle de service Greengrass qui autorise AWS IoT Greengrass à accéder aux ressources AWS en votre nom.

Gestion du rôle de groupe Greengrass (console)

Vous pouvez utiliser le pluginAWS IoTconsole pour les tâches de gestion de rôles suivantes :

- Rechercher votre rôle de groupe Greengrass
- Ajouter ou modifier le rôle de groupe Greengrass
- Retirer le rôle de groupe Greengrass

> **Note**
>
> L'utilisateur connecté à la console doit disposer des autorisations nécessaires pour gérer le rôle.

Rechercher votre rôle de groupe Greengrass (console)

Suivez ces étapes pour rechercher le rôle attribué à un groupe Greengrass.

1. DansAWS IoTVolet de navigation de la consoleGérer, développezAppareils Greengrasset puisGroupes (V1).
2. Choisissez le groupe cible.
3. Sur la page de configuration de groupe, choisissezAffichage des paramètres.

Si un rôle est attaché au groupe, il apparaît sousRôle de groupe.

Ajouter ou modifier le rôle de groupe Greengrass (console)

Procédez comme suit pour choisir un rôle IAM dans votreCompte AWSpour ajouter à un groupe Greengrass.

Un rôle de groupe est soumis aux exigences suivantes :

- AWS IoT Greengrass doit être défini en tant qu'entité de confiance.
- Les stratégies d'autorisation associées au rôle doivent accorder les autorisations à votreAWSressources requises par les fonctions Lambda et les connecteurs du groupe, ainsi que par les composants système Greengrass.

> **Note**
>
> Nous vous recommandons également d'inclure le`aws:SourceArn`et`aws:SourceAccount`Clés de contexte de condition globale dans votre stratégie d'approbation pour empêcher ledéputé confusproblème de sécurité. Les clés contextuelles de condition limitent l'accès afin d'autoriser uniquement les demandes

provenant du compte spécifié et de l'espace de travail Greengrass. Pour plus d'informations sur le problème du député confus, consultezPrévention du problème de l'adjoint confus entre services.

Utilisez la console IAM pour créer et configurer le rôle et ses autorisations. Pour plus d'informations sur les étapes de création d'un exemple de rôle permettant d'accéder à une table Amazon DynamoDB, veuillez consulterthe section called “Configuration du rôle de groupe”. Pour les étapes générales, consultezCréation d'un rôle pour unAWSservice (console)dans leIAM User Guide.

Une fois le rôle configuré, utilisez leAWS IoTpour ajouter le rôle au groupe.

> **Note**
>
> Cette procédure est requise uniquement pour choisir un rôle pour le groupe. Elle n'est pas requise après la modification des autorisations du rôle de groupe actuellement sélectionné.

1. DansAWS IoTVolet de navigation de la consoleGérer, développezAppareils Greengrasset puisGroupes (V1).
2. Choisissez le groupe cible.
3. Sur la page de configuration de groupe, choisissezAffichage des paramètres.
4. UnderRôle de groupe, choisissez d'ajouter ou de modifier le rôle :
   - Pour ajouter le rôle, choisissezRôle d'associépuis sélectionnez votre rôle dans votre liste de rôles. Voici les rôles dans votreCompte AWSqui définissentAWS IoT Greengrassen tant qu'entité de confiance.
   - Pour choisir un autre rôle, choisissezModifier le rôlepuis sélectionnez votre rôle dans votre liste de rôles.
5. Choisissez Save (Enregistrer).

Retirer le rôle de groupe Greengrass (console)

Procédez comme suit pour détacher le rôle d'un groupe Greengrass.

1. DansAWS IoTVolet de navigation de la consoleGérer, développezAppareils Greengrasset puisGroupes (V1).
2. Choisissez le groupe cible.
3. Sur la page de configuration de groupe, choisissezAffichage des paramètres.
4. UnderRôle de groupe, choisissezDissocier un rôle.
5. Dans la boîte de dialogue de confirmation, choisissezDissocier un rôle. Cette étape retire le rôle du groupe mais ne le supprime pas. Si vous souhaitez supprimer le rôle, utilisez la console IAM.

Gestion du rôle de groupe Greengrass (interface de ligne de commande)

Vous pouvez utiliser l'AWS CLI pour les tâches de gestion de rôles suivantes :

- Obtenir votre rôle de groupe Greengrass
- Créer le rôle de groupe Greengrass
- Retirer le rôle de groupe Greengrass

Obtenir le rôle de groupe Greengrass (CLI)

Suivez ces étapes pour savoir si un groupe Greengrass a un rôle associé.

1. Obtenez l'ID du groupe cible dans la liste de vos groupes.

   ```
   aws greengrass list-groups
   ```

   Voici un exemple de réponse `list-groups` : Chaque groupe de la réponse inclut une propriété `Id` qui contient l'ID de groupe.

   ```
   {
       "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"
        },
        ...
    ]
}
```

Pour de plus amples informations, y compris des exemples qui utilisent l'option `query`pour filtrer les résultats, veuillez consulter the section called “Obtention de l'ID de groupe”.

2. Copiez l'`Id` du groupe cible à partir de la sortie.
3. Obtenez le rôle de groupe. Remplacez *`group-id`* par l'ID du groupe cible.

```
aws greengrass get-associated-role --group-id group-id
```

Si un rôle est associé à votre groupe Greengrass, les métadonnées de rôle suivantes sont renvoyées.

```
{
  "AssociatedAt": "timestamp",
  "RoleArn": "arn:aws:iam::account-id:role/path/role-name"
}
```

Si votre groupe n'a pas de rôle associé, l'erreur suivante est renvoyée.

```
An error occurred (404) when calling the GetAssociatedRole operation: You need to
 attach an IAM role to this deployment group.
```

Créer le rôle de groupe Greengrass (CLI)

Procédez comme suit pour créer un rôle et l'associer à un groupe Greengrass.

Pour créer le rôle de groupe à l'aide d'IAM

1. Créez le rôle avec une stratégie d'approbation permettant à AWS IoT Greengrass d'assumer le rôle. Cet exemple crée un rôle nommé `MyGreengrassGroupRole`, mais vous pouvez utiliser un autre nom. Nous vous recommandons également d'inclure le`aws:SourceArn`et`aws:SourceAccount`Clés de contexte de condition globale dans votre stratégie d'approbation pour empêcher ledéputé confusproblème de sécurité. Les clés contextuelles de condition limitent l'accès afin d'autoriser uniquement les demandes provenant du compte spécifié et de l'espace de travail Greengrass. Pour plus d'informations sur le problème du député confus, consultezPrévention du problème de l'adjoint confus entre services.

   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. Copiez l'ARN de rôle du rôle des métadonnées dans la sortie. Vous utilisez l'ARN pour associer le rôle à votre groupe.
3. Associez des stratégies gérées ou intégrées au rôle pour prendre en charge votre analyse de rentabilité. Par exemple, si une fonction Lambda définie par l'utilisateur lit à partir d'Amazon S3, vous pouvez attacher le`AmazonS3ReadOnlyAccess`stratégie gérée au rôle.

```
aws iam attach-role-policy --role-name MyGreengrassGroupRole --policy-arn
 arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess
```

En cas de succès, aucune réponse n'est renvoyée.

Pour associer le rôle à votre groupe Greengrass

1. Obtenez l'ID du groupe cible dans la liste de vos groupes.

```
aws greengrass list-groups
```

Voici un exemple de réponse `list-groups` : Chaque groupe de la réponse inclut une propriété `Id` qui contient l'ID de groupe.

```
{
    "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"
        },
        ...
    ]
}
```

Pour de plus amples informations, y compris des exemples qui utilisent l'option `query`pour filtrer les résultats, veuillez consulter the section called “Obtention de l'ID de groupe”.

2. Copiez l'`Id` du groupe cible à partir de la sortie.
3. Associez le rôle à votre groupe. Remplacez *`group-id`* par l'ID du groupe cible et *`role-arn`* par l'ARN du rôle de groupe.

```
aws greengrass associate-role-to-group --group-id group-id --role-arn role-arn
```

En cas de réussite, la réponse suivante est renvoyée.

```
{
  "AssociatedAt": "timestamp"
}
```

Retirer le rôle de groupe Greengrass (CLI)

Procédez comme suit pour dissocier le rôle de groupe de votre groupe Greengrass.

1. Obtenez l'ID du groupe cible dans la liste de vos groupes.

   ```
   aws greengrass list-groups
   ```

   Voici un exemple de réponse `list-groups` : Chaque groupe de la réponse inclut une propriété `Id` qui contient l'ID de groupe.

   ```
   {
       "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"
           },
           ...
       ]
   }
   ```

Pour de plus amples informations, y compris des exemples qui utilisent l'option `query`pour filtrer les résultats, veuillez consulter the section called “Obtention de l'ID de groupe”.

2. Copiez l'`Id` du groupe cible à partir de la sortie.
3. Dissociez le rôle de votre groupe. Remplacez *`group-id`* par l'ID du groupe cible.

```
aws greengrass disassociate-role-from-group --group-id group-id
```

En cas de réussite, la réponse suivante est renvoyée.

```
{
  "DisassociatedAt": "timestamp"
}
```

> **Note**
>
> Vous pouvez supprimer le rôle de groupe si vous ne l'utilisez pas. Utilisez d'abord delete-role-policy pour détacher chaque stratégie gérée du rôle, puis delete-role pour supprimer le rôle. Pour plus d'informations, consultez Suppression de rôles ou de profils d'instance dans le guide de l'utilisateur IAM.

Consulter aussi

- Sujets connexes dans leIAM User Guide
  - Création d'un rôle pour la délégation d'autorisations à unAWSservice
  - Modification d'un rôle
  - Ajout et suppression d'autorisations basées sur l'identité IAM
  - Suppression de rôles ou de profils d'instance
- AWS IoT GreengrassCommandes dans leAWS CLIRéférence des commandes
  - list-groups
  - associate-role-to-group
  - disassociate-role-from-group
  - get-associated-role
- Commandes IAM dans leAWS CLIRéférence des commandes

- attach-role-policy
- create-role
- delete-role
- delete-role-policy

Prévention du problème de l'adjoint confus entre services

Le problème de l'adjoint confus est un problème de sécurité dans lequel une entité qui n'a pas l'autorisation d'effectuer une action peut contraindre une entité plus privilégiée à effectuer cette action. Dans AWS, l'emprunt d'identité entre services peut entraîner le problème de député confus. L'usurpation d'identité entre services peut se produire lorsqu'un service (le service appelant) appelle un autre service (le service appelé). Le service appelant peut être manipulé et ses autorisations utilisées pour agir sur les ressources d'un autre client auxquelles on ne serait pas autorisé d'accéder autrement. Pour éviter cela, AWS fournit des outils qui vous aident à protéger vos données pour tous les services avec des principaux de service qui ont eu accès aux ressources de votre compte.

Nous vous recommandons d'utiliser les clés de contexte de condition globale `aws:SourceArn` et `aws:SourceAccount` dans les politiques de ressources afin de limiter les autorisations à la ressource octroyées par AWS IoT Greengrass à un autre service. Si vous utilisez les deux clés de contexte de condition globale, la valeur `aws:SourceAccount` et le compte de la valeur `aws:SourceArn` doit utiliser le même ID de compte lorsqu'il est utilisé dans la même déclaration de politique.

Pour`aws:SourceArn`doit être la ressource client Greengrass associée au`sts:AssumeRole`de la demande.

Le moyen le plus efficace de se protéger contre le problème de député confus consiste à utiliser la clé de contexte de condition globale `aws:SourceArn` avec l'ARN complet de la ressource. Si vous ne connaissez pas l'ARN complet de la ressource ou si vous spécifiez plusieurs ressources, utilisez la clé de contexte de condition globale `aws:SourceArn` avec des caractères génériques (*) pour les parties inconnues de l'ARN. Par exemple, `arn:aws:greengrass:`*`region`*`:`*`account-id`*`:*`.

Pour obtenir des exemples de stratégies qui utilisent le`aws:SourceArn`et`aws:SourceAccount`clés de contexte de condition globale, consultez les rubriques suivantes :

- Créer le rôle de service Greengrass
- Créer le rôle de groupe Greengrass

- Créer et configurer un rôle d'exécution IAM pour les déploiements en bloc

Exemples de stratégies basées sur l'identité pour AWS IoT Greengrass

Par défaut, les utilisateurs et les rôles IAM ne sont pas autorisés à créer ou modifier les ressources AWS IoT Greengrass. Ils ne peuvent pas non plus exécuter des tâches à l'aide de AWS Management Console, AWS CLI ou de l'API AWS. Un administrateur IAM doit créer des politiques IAM autorisant les utilisateurs et les rôles à exécuter des opérations d'API spécifiques sur les ressources spécifiées dont ils ont besoin. Il doit ensuite attacher ces politiques aux utilisateurs ou aux groupes IAM ayant besoin de ces autorisations.

Bonnes pratiques en matière de politiques

Les stratégies basées sur l'identité déterminent si une personne peut créer, consulter ou supprimer des ressources AWS IoT Greengrass dans votre compte. Ces actions peuvent entraîner des frais pour votre Compte AWS. Lorsque vous créez ou modifiez des politiques basées sur l'identité, suivez ces instructions et recommandations :

- Démarrer avec AWS gérées et évoluez vers les autorisations de moindre privilège - Pour commencer à accorder des autorisations à vos utilisateurs et charges de travail, utilisez les politiques gérées AWS qui accordent des autorisations dans de nombreux cas d'utilisation courants. Ils sont disponibles dans votre Compte AWS. Nous vous recommandons de réduire encore les autorisations en définissant des Politiques gérées par le client AWS qui sont spécifiques à vos cas d'utilisation. Pour de plus amples informations, veuillez consulter Politiques gérées AWS ou Politiques gérées AWS pour les activités professionnelles dans le Guide de l'utilisateur IAM.
- Accorder les autorisations de moindre privilège - Lorsque vous définissez des autorisations avec des politiques IAM, accordez uniquement les autorisations nécessaires à l'exécution d'une seule tâche. Pour ce faire, vous définissez les actions qui peuvent être entreprises sur des ressources spécifiques dans des conditions spécifiques, également appelées autorisations de moindre privilège. Pour plus d'informations sur l'utilisation de IAM pour appliquer des autorisations, consultez Politiques et autorisations dans IAM dans le Guide de l'utilisateur IAM.
- Utiliser des conditions dans les politiques IAM pour restreindre davantage l'accès - Vous pouvez ajouter une condition à vos politiques afin de limiter l'accès aux actions et aux ressources. Par exemple, vous pouvez écrire une condition de politique pour spécifier que toutes les demandes doivent être envoyées via SSL. Vous pouvez également utiliser des conditions pour accorder l'accès aux actions de service si elles sont utilisées via un Service AWS spécifique, comme

AWS CloudFormation. Pour plus d'informations, consultez Conditions pour éléments de politique JSON IAM dans le Guide de l'utilisateur IAM.

- Utilisez IAM Access Analyzer pour valider vos politiques IAM afin de garantir des autorisations sécurisées et fonctionnelles - IAM Access Analyzer valide les politiques nouvelles et existantes de manière à ce que les politiques IAM respectent le langage de politique IAM (JSON) et les bonnes pratiques IAM. IAM Access Analyzer fournit plus de 100 vérifications de politiques et des recommandations exploitables pour vous aider à créer des politiques sécurisées et fonctionnelles. Pour de plus amples informations, veuillez consulter Validation de politique IAM Access Analyzer dans le Guide de l'utilisateur IAM.
- Authentification multifactorielle (MFA) nécessaire : si vous avez un scénario qui nécessite des utilisateurs IAM ou un utilisateur root dans votre Compte AWS, activez l'authentification multifactorielle pour une sécurité renforcée. Pour exiger le MFA lorsque des opérations d'API sont appelées, ajoutez des conditions MFA à vos politiques. Pour de plus amples informations, veuillez consulter Configuration de l'accès aux API protégé par MFA dans le Guide de l'utilisateur IAM.

Pour plus d'informations sur les bonnes pratiques dans IAM, veuillez consulter Bonnes pratiques de sécurité dans IAM dans le Guide de l'utilisateur IAM.

Politiques AWS gérées pour AWS IoT Greengrass

AWS IoT Greengrassgère les politiquesAWS gérées suivantes que vous pouvez utiliser pour accorder des autorisations aux utilisateurs et aux rôles IAM.

Politique	Description
AWSGreengrassFullAccess	Autorise toutes les actions AWS IoT Greengras s pour toutes vos ressources AWS. Cette politique est recommandée aux administrateurs deAWS IoT Greengrass services ou à des fins de test.
AWSGreengrassReadOnlyAccess	Autorise les actions `List` et `Get` AWS IoT Greengrass pour toutes vos ressources AWS.
AWSGreengrassResourceAccessRolePolicy	Permet d'accéder aux ressources provenant deAWS services telsAWS Lambda queAWS IoT Device Shadow. Il s'agit de la stratégie

Politique	Description
	par défaut utilisée pour le rôle de service Greengrass. Cette stratégie est conçue pour offrir une facilité d'accès générale. Vous pouvez définir une stratégie personnalisée plus restricti ve.
Grassota verteUpdateArtifactAccess	Permet d'accéder en lecture seule aux artefacts de mise à jour over-the-air (OTA) du logicielA WS IoT Greengrass Core dans tous lesRégion AWS s.

Exemples de politiques

Les exemples de stratégie définie par le client suivants accordent des autorisations pour des scénarios courants.

Exemples

- Autorisation accordée aux utilisateurs pour afficher leurs propres autorisations

Pour apprendre à créer une politique basée sur l'identité IAM à l'aide de ces exemples de documents de politique JSON, veuillez consulter Création de politiques dans l'onglet JSON dans le Guide de l'utilisateur IAM.

Autorisation accordée aux utilisateurs pour afficher leurs propres autorisations

Cet exemple montre comment créer une politique qui permet aux utilisateurs IAM d'afficher les politiques en ligne et gérées attachées à leur identité d'utilisateur. Cette politique inclut les autorisations nécessaires pour réaliser cette action sur la console ou par programmation à l'aide de l'AWS CLI ou de l'API AWS.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
```

```
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

Résolution des problèmes d’identité et d’accès pour AWS IoT Greengrass

Utilisez les informations suivantes pour identifier et résoudre les problèmes courants que vous pouvez rencontrer lorsque vous travaillez avec AWS IoT Greengrass et IAM.

Problèmes

- Je ne suis pas autorisé à effectuer une action dans AWS IoT Greengrass
- Erreur : Greengrass n'est pas autorisée à assumer le rôle de service associé à ce compte, ou à l'erreur : Échec : Le rôle de service TES n'est pas associé à ce compte.
- Erreur : Autorisation refusée lors de la tentative d'utilisation du rôle arn:aws:iam : :role/ <account-id>pour <role-name>accéder à l'URL s3 https ://-greengrass-updates.s3<region>. <region>.amazonaws.com/core/ <architecture>/greengrass-core- <distribution-version>.tar.gz.
- Le shadow de l’appareil n'est pas synchronisé avec le cloud.
- Je ne suis pas autorisé à exécuter iam :PassRole

- Je suis un administrateur et je veux autoriser d'autres utilisateurs à accéder à AWS IoT Greengrass
- Je veux autoriser des personnes extérieures à mon Compte AWS à accéder à mes ressources AWS IoT Greengrass

Pour obtenir de l'aide relative à la résolution des problèmes généraux, veuillez consulter Résolution des problèmes.

Je ne suis pas autorisé à effectuer une action dans AWS IoT Greengrass

Si vous recevez un message d'erreur indiquant que vous n'êtes pas autorisé à exécuter une action, vous devez contacter votre administrateur pour obtenir de l'aide. Votre administrateur est la personne qui vous a fourni votre nom d'utilisateur et votre mot de passe.

L'exemple d'erreur suivant se produit lorsque`mateojackson`L'utilisateur IAM essaie d'afficher les détails d'une version de définition principale, mais n'a pas`greengrass:GetCoreDefinitionVersion`autorisations.

```
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
```

Dans ce cas, Mateo demande à son administrateur de mettre à jour ses politiques pour lui permettre d'accéder à la ressource `arn:aws:greengrass:us-west-2:123456789012:/greengrass/definition/cores/78cd17f3-bc68-ee18-47bd-5bda5EXAMPLE/versions/368e9ffa-4939-6c75-859c-0bd4cEXAMPLE` à l'aide de l'action `greengrass:GetCoreDefinitionVersion`.

Erreur : Greengrass n'est pas autorisée à assumer le rôle de service associé à ce compte, ou à l'erreur : Échec : Le rôle de service TES n'est pas associé à ce compte.

Solution : Cette erreur peut se produire lorsque le déploiement échoue. Vérifiez qu'un rôle de service Greengrass est associé à votreCompte AWSdans les actuelsRégion AWS. Pour plus d'informations, consultez the section called “Gestion du rôle de service (interface de ligne de commande)” ou the section called “Gestion du rôle de service (console)”.

Erreur : Autorisation refusée lors de la tentative d'utilisation du rôle arn:aws:iam : :role/ <account-id>pour <role-name>accéder à l'URL s3 https ://-greengrass-updates.s3<region>. <region>.amazonaws.com/core/ <architecture>/greengrass-core-<distribution-version>.tar.gz.

Solution : Cette erreur peut se produire lorsqu' over-the-air La mise à jour (OTA) échoue. Dans la politique relative au rôle du signataire, ajoutez la cibleRégion AWSen tant que`Resource`. Le rôle signataire est utilisé pour pré-signer l'URL S3 pour la mise à jour du logiciel AWS IoT Greengrass. Pour plus d'informations, consultez Rôle de signataire d'URL S3.

Le shadow de l’appareil n'est pas synchronisé avec le cloud.

Solution : Assurez-vous queAWS IoT Greengrasspossède des autorisations pour`iot:UpdateThingShadow`et`iot:GetThingShadow`Actions dansRôle de service Greengrass. Si le rôle de service utilise la stratégie gérée `AWSGreengrassResourceAccessRolePolicy`, ces autorisations sont incluses par défaut.

Consultez Dépannage des problèmes de temporisation de la synchronisation shadow.

Vous pouvez rencontrer les problèmes IAM généraux que vous êtes susceptible de rencontrer lors de l'utilisation desAWS IoT Greengrass.

Je ne suis pas autorisé à exécuter iam :PassRole

Si vous recevez un message d'erreur selon lequel vous n'êtes pas autorisé à exécuter`iam:PassRole`action, vos stratégies doivent être mises à jour pour vous permettre de transmettre un rôle àAWS IoT Greengrass.

Certains Services AWS vous permettent de transmettre un rôle existant à ce service, au lieu de créer un nouveau rôle de service ou rôle lié à un service. Pour ce faire, un utilisateur doit disposer des autorisations nécessaires pour transmettre le rôle au service.

L'exemple d'erreur suivant se produit lorsqu'un utilisateur IAM nommé `marymajor` essaie d'utiliser la console pour exécuter une action dans AWS IoT Greengrass. Toutefois, l'action nécessite que le service ait des autorisations accordées par une fonction du service. Mary ne dispose pas des autorisations nécessaires pour transférer le rôle au service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

Dans ce cas, les politiques de Mary doivent être mises à jour pour lui permettre d'exécuter l'action `iam:PassRole`.

Si vous avez encore besoin d'aide, contactez votre administrateur AWS. Votre administrateur est la personne qui vous a fourni vos informations de connexion.

Je suis un administrateur et je veux autoriser d'autres utilisateurs à accéder à AWS IoT Greengrass

Pour permettre à d'autres utilisateurs d'accéder à AWS IoT Greengrass vous devez créer une entité IAM (utilisateur ou rôle) pour la personne ou l'application qui a besoin de l'accès. Ils utiliseront les informations d'identification de cette entité pour accéder à AWS. Vous devez ensuite associer une politique à l'entité qui leur accorde les autorisations appropriées dans AWS IoT Greengrass.

Pour démarrer immédiatement, veuillez consulter Création de votre premier groupe et utilisateur délégué IAM dans le Guide de l'utilisateur IAM.

Je veux autoriser des personnes extérieures à mon Compte AWS à accéder à mes ressources AWS IoT Greengrass

Vous pouvez créer un rôle IAM que les utilisateurs provenant d'autres comptes ou les personnes extérieures à votre organisation peuvent utiliser pour accéder àAWSRessources. Vous pouvez spécifier qui est approuvé pour assumer le rôle. Pour plus d'informations, veuillez consulter la rubriqueOctroi d'un accès à un utilisateur IAM dans un autreCompte AWSque vous possédezetOctroi d'un accès à des comptes Amazon Web Services appartenant à des tiersdans leIAM User Guide.

AWS IoT Greengrass ne prend pas en charge l'accès inter-comptes basé sur des stratégies basées sur des ressources ou des listes de contrôle d'accès (ACL).

Validation de la conformité pour AWS IoT Greengrass

Pour savoir si un programme Services AWS de conformité Service AWS s'inscrit dans le champ d'application de programmes de conformité spécifiques, consultez Services AWS la section de conformité et sélectionnez le programme de conformité qui vous intéresse. Pour des informations générales, voir Programmes de AWS conformité Programmes AWS de .

Vous pouvez télécharger des rapports d'audit tiers à l'aide de AWS Artifact. Pour plus d'informations, voir Téléchargement de rapports dans AWS Artifact .

Votre responsabilité en matière de conformité lors de l'utilisation Services AWS est déterminée par la sensibilité de vos données, les objectifs de conformité de votre entreprise et les lois et réglementations applicables. AWS fournit les ressources suivantes pour faciliter la mise en conformité :

- Guides de démarrage rapide sur la sécurité et la conformité : ces guides de déploiement abordent les considérations architecturales et indiquent les étapes à suivre pour déployer des environnements de base axés sur AWS la sécurité et la conformité.
- Architecture axée sur la sécurité et la conformité HIPAA sur Amazon Web Services : ce livre blanc décrit comment les entreprises peuvent créer des applications AWS conformes à la loi HIPAA.

> **Note**
>
> Tous ne Services AWS sont pas éligibles à la loi HIPAA. Pour plus d'informations, consultez le HIPAA Eligible Services Reference.

- AWS Ressources de https://aws.amazon.com/compliance/resources/ de conformité — Cette collection de classeurs et de guides peut s'appliquer à votre secteur d'activité et à votre région.
- AWS Guides de conformité destinés aux clients — Comprenez le modèle de responsabilité partagée sous l'angle de la conformité. Les guides résument les meilleures pratiques en matière de sécurisation Services AWS et décrivent les directives relatives aux contrôles de sécurité dans de nombreux cadres (notamment le National Institute of Standards and Technology (NIST), le Payment Card Industry Security Standards Council (PCI) et l'Organisation internationale de normalisation (ISO)).
- Évaluation des ressources à l'aide des règles du guide du AWS Config développeur : le AWS Config service évalue dans quelle mesure les configurations de vos ressources sont conformes aux pratiques internes, aux directives du secteur et aux réglementations.
- AWS Security Hub— Cela Service AWS fournit une vue complète de votre état de sécurité interne AWS. Security Hub utilise des contrôles de sécurité pour évaluer vos ressources AWS et vérifier votre conformité par rapport aux normes et aux bonnes pratiques du secteur de la sécurité. Pour obtenir la liste des services et des contrôles pris en charge, consultez Référence des contrôles Security Hub.
- Amazon GuardDuty — Cela Service AWS détecte les menaces potentielles qui pèsent sur vos charges de travail Comptes AWS, vos conteneurs et vos données en surveillant votre environnement pour détecter toute activité suspecte et malveillante. GuardDuty peut vous aider

à répondre à diverses exigences de conformité, telles que la norme PCI DSS, en répondant aux exigences de détection des intrusions imposées par certains cadres de conformité.

- AWS Audit Manager— Cela vous Service AWS permet d'auditer en permanence votre AWS utilisation afin de simplifier la gestion des risques et la conformité aux réglementations et aux normes du secteur.

Résilience dans AWS IoT Greengrass

LeAWSL'infrastructure mondiale d'repose sur des régions et des zones de disponibilité Amazon Web Services. EachRégion AWSLes fournit plusieurs zones de disponibilité physiquement séparées et isolées, reliées par un réseau à latence faible, à débit élevé et à forte redondance. Avec les zones de disponibilité, vous pouvez concevoir et exploiter des applications et des bases de données qui basculent automatiquement d'une zone à l'autre sans interruption. Les zones de disponibilité sont plus hautement disponibles, tolérantes aux pannes et évolutives que les infrastructures traditionnelles à un ou plusieurs centres de données.

Pour plus d'informations sur les régions et les zones de disponibilité Amazon Web Services, consultezAWSInfrastructure mondiale.

Outre l'infrastructure globale AWS, AWS IoT Greengrass propose plusieurs fonctionnalités qui contribuent à la prise en charge des vos besoins en matière de résilience et de sauvegarde de données.

- Si le noyau perd la connectivité Internet, les appareils clients peuvent continuer à communiquer sur le réseau local.
- Vous pouvez configurer le noyau pour stocker les messages non traités destinés àAWS Cloudcibles dans un cache de stockage local plutôt que dans un stockage en mémoire. Le cache de stockage local peut être conservé malgré les redémarrages du noyau (par exemple, après le déploiement d'un groupe ou un redémarrage de l'appareil), ce qui permet à AWS IoT Greengrass de continuer à traiter les messages destinés à AWS IoT Core. Pour plus d'informations, consultez the section called "File d'attente de messages MQTT".
- Vous pouvez configurer le noyau pour établir une session permanente avec le courtier de messages AWS IoT Core. Cela permet au noyau de recevoir les messages envoyés lorsqu'il est hors ligne. Pour plus d'informations, consultez the section called "Sessions persistantes MQTT avec AWS IoT Core".
- Vous pouvez configurer un groupe Greengrass pour écrire des journaux dans le système de fichiers local et sur CloudWatch Bûches. Si le noyau perd la connectivité, la journalisation locale

peut continuer,mais CloudWatch les journaux sont envoyés avec un nombre limité de tentatives. Une fois le nombre de tentatives atteint, l'événement est supprimé. YTu devraiségalementsoyez conscientLimites de la.

- Vous pouvez créer des fonctions Lambda qui lisentgestionnaire de fluxflux et envoie les données vers des destinations de stockage local.

Sécurité de l'infrastructure dans AWS IoT Greengrass

En tant que service géré, AWS IoT Greengrass est protégé par les procédures de sécurité du réseau mondial AWS. Pour plus d'informations sur les services de sécurité AWS et la manière dont AWS protège l'infrastructure, consultez la section Sécurité du cloud AWS. Pour concevoir votre environnement AWS en utilisant les meilleures pratiques en matière de sécurité de l'infrastructure, consultez la section Protection de l'infrastructure dans le Security Pillar AWS Well-Architected Framework (Pilier de sécurité de l'infrastructure Well-Architected Framework).

Vous utilisez les appels d'API publiés AWS pour accéder à AWS IoT Greengrass via le réseau. Les clients doivent prendre en charge les éléments suivants :

- Protocole TLS (Transport Layer Security). Nous exigeons TLS 1.2 et nous recommandons TLS 1.3.
- Ses suites de chiffrement PFS (Perfect Forward Secrecy) comme DHE (Ephemeral Diffie-Hellman) ou ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). La plupart des systèmes modernes tels que Java 7 et les versions ultérieures prennent en charge ces modes.

En outre, les demandes doivent être signées à l'aide d'un ID de clé d'accès et d'une clé d'accès secrète associée à un principal IAM. Vous pouvez également utiliser AWS Security Token Service (AWS STS) pour générer des informations d'identification de sécurité temporaires et signer les demandes.

Dans unAWS IoT Greengrassenvironnement, les appareils utilisent des certificats X.509 et des clés cryptographiques pour se connecter et s'authentifier auprès duAWS Cloud. Pour plus d'informations, veuillez consulter the section called “Authentification et autorisation d'appareil”.

Configuration et analyse des vulnérabilités dans AWS IoT Greengrass

Les environnements IoT sont composés d'un grand nombre d'appareils disposant de capacités diverses, d'une durée de vie longue et qui sont répartis géographiquement. Ces caractéristiques peuvent rendre la configuration des appareils complexe et source d'erreurs. Les appareils étant souvent limités en puissance de calcul, de mémoire et de capacités de stockage, l'utilisation du chiffrement et d'autres formes de sécurité sur les appareils eux-mêmes s'en trouve limitée. En outre, les appareils utilisent souvent des logiciels aux vulnérabilités connues. Ces facteurs font des appareils IoT une cible attractive pour les pirates informatiques et rend difficile leur sécurisation sur une base permanente.

AWS IoT Device Defender résout ces problèmes en fournissant des outils pour identifier les problèmes de sécurité, ainsi que les écarts par rapport aux bonnes pratiques. Vous pouvez utiliser AWS IoT Device Defender pour analyser, auditer et surveiller les appareils connectés afin de détecter les comportements anormaux et d'atténuer les risques liés à la sécurité. AWS IoT Device Defender peut auditer les appareils pour s'assurer qu'ils respectent les bonnes pratiques en matière de sécurité et détecter les comportements anormaux sur les appareils. Cela vous permet d'appliquer des stratégies de sécurité cohérentes sur l'ensemble de vos appareils et de réagir rapidement lorsque des appareils sont menacés. Dans les connexions avec AWS IoT Core, AWS IoT Greengrass génère des ID client prévisibles que vous pouvez utiliser avec des fonctionnalités AWS IoT Device Defender. Pour plus d'informations, consultez AWS IoT Device Defender dans le Guide du développeur AWS IoT Core.

Dans les environnements AWS IoT Greengrass, vous devez prendre en compte les considérations suivantes :

- Il est de votre responsabilité de sécuriser vos appareils physiques, le système de fichiers de vos appareils et le réseau local.
- AWS IoT Greengrassn'impose pas l'isolation du réseau pour les fonctions Lambda définies par l'utilisateur, qu'elles soient exécutées ou non dans un conteneur Greengrass. Il est donc possible pour les fonctions Lambda de communiquer avec n'importe quel autre processus exécuté dans le système ou à l'extérieur via le réseau.

Si vous perdez le contrôle d'un appareil principal de Greengrass et que vous souhaitez empêcher les appareils clients de transmettre des données au cœur, procédez comme suit :

1. Retirez le noyau Greengrass du groupe Greengrass.
2. Effectuez une rotation du certificat de CA de groupe. Dans la AWS IoT console, vous pouvez faire pivoter le certificat CA sur la page Paramètres du groupe. Dans l'AWS IoT GreengrassAPI, vous pouvez utiliser l'CreateGroupCertificateAuthorityaction.

   Nous vous recommandons également d'utiliser le chiffrement complet du disque si le disque dur de votre appareil principal est susceptible d'être volé.

AWS IoT Greengrass et interface des points de terminaison VPC (AWS PrivateLink)

Vous pouvez établir une connexion privée entre votre VPC et leAWS IoT Greengrassplan de contrôle en créant unPoint de terminaison d'un VPC d'interface. Vous pouvez utiliser ce point de terminaison pour gérer des groupes, des fonctions Lambda, des déploiements et d'autres ressources dans leAWS IoT Greengrassservice. Les points de terminaison d'interface sont alimentés parAWS PrivateLink, une technologie qui vous permet d'accéderAWS IoT GreengrassAPI privées sans passerelle Internet, périphérique NAT, connexion VPN ouAWSConnexion Direct Connect. Les instances de votre VPC ne requièrent pas d'adresses IP publiques pour communiquer avec les API AWS IoT Greengrass. Le trafic entre votre VPC et AWS IoT Greengrass ne quitte pas le réseau Amazon.

> **Note**
>
> Actuellement, vous ne pouvez pas configurer les périphériques principaux de Greengrass pour qu'ils fonctionnent complètement au sein de votre VPC.

Chaque point de terminaison d'interface est représenté par une ou plusieurs interfaces réseau Elastic dans vos sous-réseaux.

Pour de plus amples informations, consultez Points de terminaison VPC (AWS PrivateLink) dans le Guide de l'utilisateur Amazon VPC.

Rubriques

- Considérations relatives aux points de terminaison de VPC AWS IoT Greengrass
- Créer un point de terminaison de VPC d'interface pourAWS IoT Greengrassopérations de plan de contrôle

- Création d'une stratégie de point de terminaison d'un VPC pour AWS IoT Greengrass

Considérations relatives aux points de terminaison de VPC AWS IoT Greengrass

Avant de configurer un point de terminaison de VPC d'interface pourAWS IoT Greengrass, revuePropriétés et limites des points de terminaison d'interfacedans leAmazon VPC User Guide. De plus, soyez conscient des points suivants :

- AWS IoT Greengrassprend en charge l'exécution d'appels en direction de toutes ses actions d'API de plan de contrôle depuis votre VPC. Le plan de contrôle inclut des opérations telles queCreateDeploymentetDémarrer le déploiement en vrac. Le plan de contrôle ne fait paspasinclure des opérations telles queGetDeploymentetDécouvrez, qui sont des opérations de plan de données.
- Points de terminaison VPC pourAWS IoT Greengrassne sont pas pris en charge parAWSRégions chinoises.

Créer un point de terminaison de VPC d'interface pourAWS IoT Greengrassopérations de plan de contrôle

Vous pouvez créer un point de terminaison de VPC pour leAWS IoT Greengrassplan de contrôle à l'aide de la console Amazon VPC ou de la consoleAWS Command Line Interface(AWS CLI). Pour de plus amples informations, veuillez consulter Création d'un point de terminaison d'interface dans le Guide de l'utilisateur Amazon VPC.

Pour créer un point de terminaison de VPC pour AWS IoT Greengrass, utilisez le nom de service suivant :

- com.amazonaws.*region*.greengrass

Si vous activez le DNS privé pour le point de terminaison, vous pouvez faire des demandes d'API à AWS IoT Greengrass en utilisant son nom DNS par défaut pour la région, par exemple `greengrass.us-east-1.amazonaws.com`. Le DNS privé est activé par défaut.

Pour plus d'informations, consultez Accès à un service via un point de terminaison d'interface dans le Guide de l'utilisateur Amazon VPC.

Création d'une stratégie de point de terminaison d'un VPC pour AWS IoT Greengrass

Vous pouvez attacher une stratégie de point de terminaison à votre point de terminaison de VPC qui contrôle l'accès àAWS IoT Greengrassopérations du plan de contrôle. La politique spécifie les informations suivantes :

- Le principal qui peut exécuter des actions.
- Les actions que le principal peut effectuer.
- Ressources sur lesquelles le mandataire peut effectuer des actions.

Pour plus d'informations, consultez Contrôle de l'accès aux services avec points de terminaison d'un VPC dans le Guide de l'utilisateur Amazon VPC.

Example Exemple : Politique de point de terminaison de VPC pourAWS IoT Greengrassactions

Voici un exemple de stratégie de point de terminaison pour AWS IoT Greengrass. Lorsqu'elle est attachée à un point de terminaison, cette politique accorde l'accès aux actions AWS IoT Greengrass répertoriées pour tous les principaux sur toutes les ressources.

```
{
    "Statement": [
        {
            "Principal": "*",
            "Effect": "Allow",
            "Action": [
                "greengrass:CreateDeployment",
                "greengrass:StartBulkDeployment"
            ],
            "Resource": "*"
        }
    ]
}
```

Bonnes pratiques de sécurité pour AWS IoT Greengrass

Cette rubrique contient les bonnes pratiques en matière de sécurité pour AWS IoT Greengrass.

Accorder le moins d’autorisations possibles

Respectez le principe du moindre privilège en utilisant l'ensemble minimal d'autorisations dans les rôles IAM. Limitez l'utilisation du `*` caractère générique pour les `Resource` propriétés `Action` et dans vos politiques IAM. Au lieu de cela, déclarez un ensemble fini d'actions et de ressources lorsque cela est possible. Pour plus d'informations sur le moindre privilège et les autres bonnes pratiques en matière de stratégie, veuillez consulter the section called “Bonnes pratiques en matière de politiques”.

La meilleure pratique du moindre privilège s'applique également aux AWS IoT politiques que vous attachez à votre cœur de Greengrass et à vos appareils clients.

Ne codez pas en dur les informations d'identification dans les fonctions Lambda

Ne codez pas en dur les informations d'identification dans vos fonctions Lambda définies par l'utilisateur. Pour mieux protéger vos informations d'identification :

- Pour interagir avec les services AWS, définissez des autorisations pour des actions et des ressources spécifiques dans le rôle de groupe Greengrass.
- Utilisez les secrets locaux pour stocker vos informations d'identification. Ou, si la fonction utilise le AWS SDK, utilisez les informations d'identification de la chaîne de fournisseurs d'informations d'identification par défaut.

Ne journalisez pas les informations sensibles

Vous devez empêcher la journalisation des informations d'identification et d'autres informations personnelles identifiables (PII). Nous vous recommandons de mettre en œuvre les mesures de protection suivantes, même si l'accès aux journaux locaux sur un appareil principal nécessite des privilèges root et que l'accès aux CloudWatch journaux nécessite des autorisations IAM.

- N'utilisez pas d'informations sensibles dans les chemins de rubrique MQTT.
- N'utilisez pas d'informations sensibles dans les noms, les types et les attributs d’appareil (objet) dans le registre AWS IoT Core.
- N'enregistrez pas d'informations sensibles dans vos fonctions Lambda définies par l'utilisateur.
- N'utilisez pas d'informations sensibles dans les noms et identifiants de ressource Greengrass :
  - Connecteurs

- Cœurs
- Appareils
- Fonctions
- Groups
- Loggers
- Ressources (local, Machine Learning ou secret)
- Abonnements

Créer des abonnements ciblés

Les abonnements contrôlent le flux d'informations dans un groupe Greengrass en définissant la manière dont les messages sont échangés entre les services, les appareils et les fonctions Lambda. Pour vous assurer qu'une application ne peut faire que ce qu'elle est censée faire, vos abonnements doivent permettre aux éditeurs d'envoyer des messages à des rubriques spécifiques uniquement, et limiter les abonnés à recevoir des messages de rubriques nécessaires à leur fonctionnalité.

Veiller à la synchronisation de l'horloge de votre appareil

Il est important que l'heure soit exacte sur votre appareil. Les certificats X.509 ont une date et une heure d'expiration. L'horloge de votre appareil est utilisée pour vérifier qu'un certificat de serveur est toujours valide. Les horloges de l'appareil peuvent se décaler au fil du temps ou les batteries peuvent se décharger.

Pour de plus amples informations, veuillez consulter les bonnes pratiques décrites dans la section Veiller à la synchronisation de l'horloge de votre appareil dans le Manuel du développeur AWS IoT Core.

Gérer l'authentification des appareils avec le noyau Greengrass

Les appareils clients peuvent exécuter FreeRTOS ou utiliser AWS IoTle Device SDK AWS IoT Greengrass ou l'API Discovery pour obtenir les informations de découverte utilisées pour se connecter et s'authentifier auprès du noyau d'un même groupe Greengrass. Les informations de découverte comprennent :

- Informations de connectivité pour le cœur de Greengrass qui se trouve dans le même groupe Greengrass que l'appareil client. Ces informations comprennent l'adresse de l'hôte et le numéro de port de chaque point de terminaison pour l’appareil noyau.

- Le certificat d'autorité de certification de groupe utilisé pour signer le certificat de serveur MQTT local. Les appareils clients utilisent le certificat CA du groupe pour valider le certificat du serveur MQTT présenté par le noyau.

Voici les meilleures pratiques permettant aux appareils clients de gérer l'authentification mutuelle avec un noyau Greengrass. Ces pratiques peuvent vous aider à atténuer vos risques si votre appareil noyau est endommagé.

Valider le certificat de serveur MQTT local pour chaque connexion.

Les appareils clients doivent valider le certificat de serveur MQTT présenté par le cœur chaque fois qu'ils établissent une connexion avec le cœur. Cette validation concerne le dispositif client dans le cadre de l'authentification mutuelle entre un dispositif principal et des appareils clients. Les appareils clients doivent être en mesure de détecter une panne et de mettre fin à la connexion.

Ne pas coder en dur les informations de découverte.

Les appareils clients doivent s'appuyer sur des opérations de découverte pour obtenir les informations de connectivité de base et le certificat de l'autorité de certification du groupe, même si le cœur utilise une adresse IP statique. Les appareils clients ne doivent pas coder en dur ces informations de découverte.

Mettre à jour périodiquement les informations de découverte.

Les appareils clients doivent régulièrement exécuter la détection pour mettre à jour les informations de connectivité de base et le certificat de l'autorité de certification du groupe. Nous recommandons aux appareils clients de mettre à jour ces informations avant d'établir une connexion avec le cœur. Étant donné que des durées plus courtes entre les opérations de découverte peuvent réduire le temps d'exposition potentiel, nous recommandons que les appareils clients se déconnectent et se reconnectent régulièrement pour déclencher la mise à jour.

Si vous perdez le contrôle d'un appareil principal de Greengrass et que vous souhaitez empêcher les appareils clients de transmettre des données au cœur, procédez comme suit :

1. Retirez le noyau Greengrass du groupe Greengrass.

2. Effectuez une rotation du certificat de CA de groupe. Dans la AWS IoT console, vous pouvez faire pivoter le certificat CA sur la page Paramètres du groupe. Dans l'AWS IoT GreengrassAPI, vous pouvez utiliser l'CreateGroupCertificateAuthorityaction.

   Nous vous recommandons également d'utiliser le chiffrement complet du disque si le disque dur de votre appareil principal est susceptible d’être volé.

Pour plus d’informations, consultez the section called “Authentification et autorisation d'appareil”.

Consultez aussi

- Les meilleures pratiques en matière de sécurité sont AWS IoT Core décrites dans le guide du AWS IoT développeur
- Dix règles d'or en matière de sécurité pour les solutions IoT industrielles sur l'Internet des objets sur le blog AWS officiel

Journalisation et surveillance dans AWS IoT Greengrass

La surveillance est un enjeu important pour assurer la fiabilité, la disponibilité et les performances d'AWS IoT Greengrass et de vos solutions AWS. Vous devez recueillir les données de surveillance de toutes les parties de votre solution AWS de façon à pouvoir déboguer plus facilement un éventuel échec multipoint. Avant de commencer la surveillance de AWS IoT Greengrass, vous devez créer un plan de surveillance qui contient les réponses aux questions suivantes :

- Quels sont les objectifs de la surveillance ?
- Quelles sont les ressources à surveiller ?
- A quelle fréquence les ressources doivent-elles être surveillées ?
- Quels outils de surveillance utiliser ?
- Qui exécute les tâches de supervision ?
- Qui doit être informé en cas de problème ?

Outils de surveillance

AWS fournit des outils que vous pouvez utiliser pour surveiller AWS IoT Greengrass. Vous pouvez configurer certains de ces outils afin qu'ils effectuent la surveillance à votre place. Une intervention manuelle est nécessaire pour certains outils. Nous vous recommandons d'automatiser le plus possible les tâches de supervision.

Vous pouvez utiliser les outils de surveillance automatique pour surveillerAWS IoT Greengrasset signalez les problèmes :

- Amazon CloudWatch Journaux— Contrôler vos fichiers journaux, stockez-les et accédez-y à partir deAWS CloudTrailou d'autres sources. Pour de plus amples informations, veuillez consulterSurveillance des fichiers journauxdans leAmazon CloudWatch Guide de l'utilisateur.
- AWS CloudTrailSurveillance des journaux— Partager des fichiers journaux entre comptes, surveillez CloudTrail les fichiers journaux en temps réel en les envoyant à CloudWatch Enregistre des journaux, écrivez des applications de traitement des journaux en Java et vérifiez que vos fichiers journaux n'ont pas changé après leur livraison par CloudTrail. Pour de plus amples informations, veuillez consulterUtilisation d' CloudTrail Fichiers journauxdans leAWS CloudTrailGuide de l'utilisateur.

- Amazon EventBridge— Utilisation de EventBridge pour obtenir des notifications sur les changements d'état pour vos déploiements de groupe Greengrass ou les appels d'API enregistrés avec CloudTrail. Pour de plus amples informations, veuillez consulterthe section called “Obtention des notifications de déploiement”ouQu'est-ce qu'Amazon EventBridge ?dans leAmazon EventBridge Guide de l'utilisateur.
- Télémétrie sanitaire du système Greengrass— Abonnez-vous pour recevoir des données de télémétrie envoyées par le noyau de Greengrass. Pour plus d'informations, consultez the section called “Collecte de données de télémétrie sur l'état du système”.
- Vérification de l'état— Utilisez les API d'état pour obtenir un instantané de l'état de l'état localAWS IoT Greengrassprocessus sur l'appareil principal. Pour plus d'informations, consultez the section called “Appel de l'API de contrôle de santé locale”.

Consulter aussi

- the section called “Surveillance avec les journaux AWS IoT Greengrass”
- the section called “Journalisation des appels d'API AWS IoT Greengrass avec AWS CloudTrail”
- the section called “Obtention des notifications de déploiement”

Surveillance avec les journaux AWS IoT Greengrass

AWS IoT Greengrass se compose du service cloud et du logiciel AWS IoT Greengrass Core. Le logiciel AWS IoT Greengrass Core peut écrire des journaux sur Amazon CloudWatch et dans le système de fichiers local de votre appareil principal. Les fonctions Lambda et les connecteurs exécutés sur le noyau peuvent également écrire des journaux dans Logs et dans le système de fichiers local. CloudWatch Vous pouvez utiliser les journaux pour surveiller les événements et résoudre les problèmes. Toutes les entrées de journal AWS IoT Greengrass comportent un horodatage, un niveau de journalisation, ainsi que des informations sur l'événement. Les modifications apportées aux paramètres de journalisation prennent effet après le déploiement du groupe.

La journalisation est configurée au niveau du groupe. Pour savoir comment configurer la journalisation pour un groupe Greengrass, veuillez consulter the section called “Configurer la journalisation pour AWS IoT Greengrass”.

Accès aux CloudWatch journaux

Si vous configurez la CloudWatch journalisation, vous pouvez consulter les journaux sur la page Logs de la CloudWatch console Amazon. Les groupes de journaux pour les journaux AWS IoT Greengrass utilisent les conventions de dénomination suivantes :

```
/aws/greengrass/GreengrassSystem/greengrass-system-component-name
/aws/greengrass/Lambda/aws-region/account-id/lambda-function-name
```

Chaque groupe de journaux contient des flux de journaux qui utilisent la convention de dénomination suivante :

```
date/account-id/greengrass-group-id/name-of-core-that-generated-log
```

Les considérations suivantes s'appliquent lorsque vous utilisez CloudWatch des journaux :

- Les journaux sont envoyés à CloudWatch Logs avec un nombre limité de tentatives en cas d'absence de connexion Internet. Une fois le nombre de tentatives atteint, l'événement est supprimé.
- Transaction, mémoire et d'autres limitations s'appliquent. Pour plus d'informations, consultez the section called “Limites de la journalisation”.
- Votre rôle dans le groupe Greengrass doit vous permettre d'AWS IoT Greengrassécrire dans Logs. CloudWatch Pour accorder des autorisations, intégrez la stratégie en ligne suivante à votre rôle de groupe.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents",
                "logs:DescribeLogStreams"
            ],
            "Resource": [
                "arn:aws:logs:*:*:*"
            ]
```

```
        }
    ]
}
```

> **Note**
>
> Vous pouvez accorder un accès plus granulaire à votre journal de ressources. Pour plus d'informations, consultez la section Utilisation de politiques basées sur l'identité (politiques IAM) pour les CloudWatch journaux dans le guide de l'utilisateur Amazon CloudWatch .

Le rôle de groupe est un rôle IAM que vous créez et associez à votre groupe Greengrass. Vous pouvez utiliser la console ou l'API AWS IoT Greengrass pour gérer le rôle de groupe.

Utilisation de la console

1. Dans le volet de navigation de la AWS IoT console, sous Gérer, développez les appareils Greengrass, puis choisissez Groups (V1).
2. Choisissez le groupe cible.
3. Choisissez Afficher les paramètres. Sous Rôle de groupe, vous pouvez afficher, associer ou dissocier le rôle de groupe.

   Pour connaître la procédure permettant d'attacher le rôle de groupe, veuillez consulter rôle de groupe.

Utilisation de la CLI

- Pour trouver le rôle du groupe, utilisez la get-associated-rolecommande.
- Pour associer le rôle de groupe, utilisez la associate-role-to-groupcommande.
- Pour supprimer le rôle de groupe, utilisez la disassociate-role-from-groupcommande.

Pour savoir comment obtenir l'ID de groupe à utiliser avec ces commandes, reportez-vous à the section called “Obtention de l'ID de groupe”.

Accès aux journaux du système de fichiers

Si vous configurez la journalisation de système de fichiers, les fichiers journaux sont stockés sous *greengrass-root*/ggc/var/log sur l'appareil principal. Voici la structure de répertoires de haut niveau :

```
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**
>
> Par défaut, *greengrass_root* est le répertoire /greengrass. Si un répertoire en écriture est configuré, les journaux se trouvent dans ce répertoire.

Les considérations suivantes s'appliquent lorsque vous utilisez les journaux du système de fichiers :

- La lecture des journaux AWS IoT Greengrass sur le système de fichiers requiert des autorisations racine.
- AWS IoT Greengrass prend en charge la rotation basée sur la taille et le nettoyage automatique lorsque le volume des données de journalisation est proche de la limite configurée.
- Le fichier crash.log est disponible dans les journaux du système de fichiers uniquement. Ce journal n'est pas écrit dans CloudWatch Logs.
- Des limitations d'utilisation du disque s'appliquent. Pour plus d’informations, consultez the section called “Limites de la journalisation”.

> **Note**
>
> Les journaux relatifs aux logiciels AWS IoT Greengrass Core v1.0 sont stockés sous le répertoire *greengrass-root*/var/log.

Configuration de journalisation par défaut

Si les paramètres de journalisation ne sont pas configurés explicitement, AWS IoT Greengrass utilise la configuration de journalisation par défaut suivante après le déploiement du premier groupe.

AWS IoT Greengrass Composants système

- Type - `FileSystem`
- Composant - `GreengrassSystem`
- Niveau - `INFO`
- Espace - `128 KB`

Fonctions Lambda définies par l'utilisateur

- Type - `FileSystem`
- Composant - `Lambda`
- Niveau - `INFO`
- Espace - `128 KB`

> **Note**
>
> Avant le premier déploiement, seuls les composants du système écrivent des journaux dans le système de fichiers, car aucune fonction Lambda définie par l'utilisateur n'est déployée.

Configurer la journalisation pour AWS IoT Greengrass

Vous pouvez utiliser la AWS IoT console ou les AWS IoT GreengrassAPI pour configurer la AWS IoT Greengrass journalisation.

> **Note**
>
> AWS IoT GreengrassPour autoriser l'écriture de journaux dans CloudWatch Logs, votre rôle de groupe doit autoriser les actions CloudWatch Logs requises.

Configurer la journalisation (console)

Vous pouvez configurer la journalisation sur la page Settings (Paramètres) du groupe.

1. Dans le volet de navigation de la AWS IoT console, sous Gérer, développez les appareils Greengrass, puis choisissez Groups (V1).
2. Choisissez le groupe dans lequel vous souhaitez configurer la journalisation.
3. Sur la page de configuration du groupe, choisissez l'onglet Logs.
4. Choisissez l'emplacement de journalisation, comme suit :

   - Pour configurer la CloudWatch journalisation, pour la configuration CloudWatch des journaux, choisissez Modifier.
   - Pour configurer la journalisation du système de fichiers, pour la configuration des journaux locaux, choisissez Edit (Modifier).

   Vous pouvez configurer la journalisation pour un emplacement ou les deux.
5. Dans le mode de configuration des journaux d'édition, sélectionnez le niveau de journal du système Greengrass ou le niveau de journal des fonctions User Lambda. Vous pouvez choisir un composant ou les deux.
6. Choisissez le plus bas niveau d'événements que vous souhaitez consigner. Les évènements en dessous de ce seuil sont filtrés, mais ne sont pas stockés.
7. Choisissez Enregistrer. Les modifications prennent effet après le déploiement du groupe.

Configurer la journalisation (API)

Vous pouvez utiliser les API de journalisation AWS IoT Greengrass pour configurer la journalisation par programmation. Par exemple, utilisez l'action `CreateLoggerDefinition` pour créer une définition d'enregistreur d'événements en fonction d'une charge utile `LoggerDefinitionVersion`, qui utilise la syntaxe suivante :

```
{
  "Loggers": [
    {
      "Id": "string",
      "Type": "FileSystem|AWSCloudWatch",
      "Component": "GreengrassSystem|Lambda",
      "Level": "DEBUG|INFO|WARN|ERROR|FATAL",
      "Space": "integer"
    },
    {
      "Id": "string",
      ...
    }
  ]
}
```

`LoggerDefinitionVersion` est un tableau d'un ou de plusieurs objets `Logger` dont les propriétés sont les suivantes :

`Id`

Identifiant unique de l'enregistreur d'événements.

`Type`

Mécanisme de stockage pour les événements de journaux. Lorsqu'il `AWSCloudWatch` est utilisé, les événements du journal sont envoyés à CloudWatch Logs. Lorsque `FileSystem` est utilisé, les journaux d'événements sont stockés dans le système de fichiers local.

Valeurs valides : `AWSCloudWatch`, `FileSystem`

`Component`

Source de l'événement de journal. Lorsque `GreengrassSystem` est utilisé, les événements provenant des composants système Greengrass sont consignés. Quand `Lambda` est utilisé, les événements provenant des fonctions Lambda définies par l'utilisateur sont consignés.

Valeurs valides : `GreengrassSystem`, `Lambda`

`Level`

Seuil du niveau de journalisation. Les journaux d'évènements en dessous de ce seuil sont filtrés, mais ne sont pas stockés.

Valeurs valides : `DEBUG`, `INFO` (recommandée), `WARN`, `ERROR`, `FATAL`

`Space`

Volume maximum de stockage local, en Ko, à utiliser pour stocker les journaux. Ce champ s'applique uniquement lorsque `Type` est défini sur `FileSystem`.

Exemple de configuration

L'exemple `LoggerDefinitionVersion` suivant spécifie un fichier de configuration qui :

- Active la journalisation du système de fichiers `ERROR` et au-delà pour les composants AWS IoT Greengrass du système.
- Active la journalisation du système de fichiers `INFO` (et versions ultérieures) pour les fonctions Lambda définies par l'utilisateur.
- Active CloudWatch `INFO` (et au-delà) la journalisation pour les fonctions Lambda définies par l'utilisateur.

```
{
  "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"
```

```
      }
    ]
  }
}
```

Lorsque vous avez créé une version de la définition de l'enregistreur d'événements, vous pouvez utiliser son ARN de version pour créer une version de groupe avant de déployer le groupe.

Limites de la journalisation

AWS IoT Greengrass possède les limitations de journalisation suivantes.

Transactions par seconde

Lorsque la connexion à CloudWatch est activée, le composant de journalisation regroupe les événements du journal localement avant de les envoyer CloudWatch, de sorte que vous pouvez enregistrer à un rythme supérieur à cinq demandes par seconde et par flux de journal.

Mémoire

S'il AWS IoT Greengrass est configuré pour envoyer des journaux CloudWatch et qu'une fonction Lambda enregistre plus de 5 Mo/seconde pendant une période prolongée, le pipeline de traitement interne finit par se remplir. Le pire scénario théorique est de 6 Mo par fonction Lambda.

Décalage d'horloge

Lorsque la connexion à CloudWatch est activée, le composant de journalisation signe les demandes en CloudWatch utilisant le processus de signature normal de Signature Version 4. Si l'heure du système sur le périphérique AWS IoT Greengrass principal est désynchronisée de plus de 15 minutes, les demandes sont rejetées.

Utilisation du disque

Utilisez la formule suivante pour calculer le volume total maximal d'espace disque affecté à la journalisation.

```
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
```

Où :

`greengrass-system-component-space`

Volume maximal de stockage local pour les journaux des composants du système AWS IoT Greengrass.

`lambda-space`

La quantité maximale de stockage local pour les journaux des fonctions Lambda.

`lambda-count`

Le nombre de fonctions Lambda déployées.

Perte de journaux

Si votre appareil AWS IoT Greengrass principal est configuré pour se connecter uniquement à Internet CloudWatch et qu'il n'y a aucune connexion Internet, vous n'avez aucun moyen de récupérer les journaux actuellement en mémoire.

Lorsque les fonctions Lambda sont interrompues (par exemple, pendant le déploiement), les journaux de quelques secondes ne sont pas écrits. CloudWatch

CloudTrail journaux

AWS IoT Greengrassfonctionne avecAWS CloudTrail, un service qui fournit un enregistrement des actions entreprises par un utilisateur, un rôle ou un AWS service dansAWS IoT Greengrass. Pour plus d'informations, consultez the section called “Journalisation des appels d'API AWS IoT Greengrass avec AWS CloudTrail”.

Journalisation des appels d'API AWS IoT Greengrass avec AWS CloudTrail

AWS IoT Greengrassest intégré à AWS CloudTrail un service qui fournit un enregistrement des actions entreprises par un utilisateur, un rôle ou un AWS service dansAWS IoT Greengrass. CloudTrail capture tous les appels d'API AWS IoT Greengrass sous forme d'événements. Les appels

capturés incluent des appels de la console AWS IoT Greengrass et les appels de code vers les opérations d'API AWS IoT Greengrass. Si vous créez un suivi, vous pouvez activer la diffusion continue d' CloudTrail événements vers un compartiment Amazon S3, y compris les événements pourAWS IoT Greengrass. Si vous ne configurez pas de suivi, vous pouvez toujours consulter les événements les plus récents dans la CloudTrail console dans Historique des événements. À l'aide des informations collectées par CloudTrail, vous pouvez déterminer la demande qui a été faiteAWS IoT Greengrass, l'adresse IP à partir de laquelle la demande a été faite, qui a fait la demande, quand elle a été faite et des détails supplémentaires.

Pour en savoir plus CloudTrail, consultez le guide de AWS CloudTrail l'utilisateur.

AWS IoT Greengrassinformations dans CloudTrail

CloudTrail est activé sur votre compte Compte AWS lorsque vous créez le compte. Lorsqu'une activité se produit dansAWS IoT Greengrass, cette activité est enregistrée dans un CloudTrail événement avec d'autres événements de AWS service dans l'historique des événements. Vous pouvez afficher, rechercher et télécharger les événements récents dans votre Compte AWS. Pour plus d'informations, consultez la section Affichage des événements avec l'historique des CloudTrail événements.

Pour un enregistrement continu des événements dans votre Compte AWS, y compris les événements pour AWS IoT Greengrass, créez un journal d'activité. Un suivi permet CloudTrail de fournir des fichiers journaux à un compartiment Amazon S3. Par défaut, lorsque vous créez un journal d'activité dans la console, il s'applique à toutes les régions Région AWS. Le journal d’activité consigne les événements de toutes les Régions dans la partition AWS et livre les fichiers journaux dans le compartiment Amazon S3 de votre choix. En outre, vous pouvez configurer d'autres AWS services pour analyser plus en détail les données d'événements collectées dans les CloudTrail journaux et agir en conséquence. Pour en savoir plus, consultez les ressources suivantes :

- Présentation de la création d’un journal de suivi
- CloudTrail services et intégrations pris en charge
- Configuration des notifications Amazon SNS pour CloudTrail
- Réception de fichiers CloudTrail journaux de plusieurs régions et réception de fichiers CloudTrail journaux de plusieurs comptes

Toutes les AWS IoT Greengrass actions sont enregistrées CloudTrail et documentées dans la référence de l'AWS IoT GreengrassAPI. Par

exemple, les appels aux `CreateFunctionDefinition` actions `AssociateServiceRoleToAccount``GetGroupVersion`,`GetConnectivityInfo`, et génèrent des entrées dans les fichiers CloudTrail journaux.

Chaque événement ou entrée de journal contient des informations sur la personne ayant initié la demande. Les informations relatives à l'identité permettent de déterminer les éléments suivants :

- Si la demande a été effectuée avec les informations d'identification utilisateur racine ou AWS Identity and Access Management (IAM).
- Si la demande a été effectuée avec les informations d'identification de sécurité temporaires d'un rôle ou d'un utilisateur fédéré.
- Si la demande a été effectuée par un autre service AWS.

Pour de plus amples informations, veuillez consulter l'élément userIdentity CloudTrail .

Présentation des entrées des fichiers journaux AWS IoT Greengrass

Un suivi est une configuration qui permet de transmettre des événements sous forme de fichiers journaux à un compartiment Amazon S3 que vous spécifiez. CloudTrail les fichiers journaux contiennent une ou plusieurs entrées de journal. Un événement représente une demande unique provenant de n'importe quelle source et inclut des informations sur l'action demandée, la date et l'heure de l'action, les paramètres de la demande, etc. CloudTrail les fichiers journaux ne constituent pas une trace ordonnée des appels d'API publics, ils n'apparaissent donc pas dans un ordre spécifique.

L'exemple suivant montre une entrée de CloudTrail journal illustrant l'`AssociateServiceRoleToAccount`action.

```
{
    "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'exemple suivant montre une entrée de CloudTrail journal illustrant l'`GetGroupVersion`action.

```
{
    "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'exemple suivant montre une entrée de CloudTrail journal illustrant l'`GetConnectivityInfo`action.

```
{
    "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'exemple suivant montre une entrée de CloudTrail journal illustrant l'`CreateFunctionDefinition`action.

```
{
    "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"
}
```


Consultez aussi

- Qu'est-ce que AWS CloudTrail ? dans le Guide de l'utilisateur AWS CloudTrail
- Création d'une EventBridge règle qui se déclenche lors d'un appel d'AWSAPI CloudTrail à l'aide du guide de EventBridge l'utilisateur Amazon

- Référence d'API AWS IoT Greengrass

Collecte de données de télémétrie sur l'état du système à partir des appareilsAWS IoT Greengrass principaux

Les données de télémétrie sur l'état du système sont des données de diagnostic qui peuvent vous aider à surveiller les performances des opérations critiques sur vos appareils principaux Greengrass. L'agent de télémétrie du noyau Greengrass collecte des données de télémétrie locales et les publie sur Amazon EventBridge sans nécessiter d'interaction avec le client. Les appareils principaux publient des données de télémétrie dans EventBridge la mesure du possible. Par exemple, les appareils principaux peuvent ne pas fournir de données de télémétrie lorsqu'ils sont hors ligne.

> **Note**
>
> Amazon EventBridge est un service de bus d'événements que vous pouvez utiliser pour connecter vos applications à des données provenant de diverses sources, telles que les appareils principaux de Greengrass et les notifications de déploiement. Pour plus d'informations, consultez Qu'est-ce qu'Amazon EventBridge ? dans le guide de EventBridge l'utilisateur Amazon.

Vous pouvez créer des projets et des applications pour récupérer, analyser, transformer et générer des rapports de télémétrie à partir de vos appareils de périphérie. Les experts du domaine, tels que les ingénieurs de procédés, peuvent utiliser ces applications pour mieux comprendre l'état de santé de la flotte.

Pour garantir le bon fonctionnement des composants de pointe de Greengrass,AWS IoT Greengrass utilise les données à des fins de développement et d'amélioration de la qualité. Cette fonctionnalité permet également de définir des fonctionnalités de pointe nouvelles et améliorées. AWS IoT Greengrassne conserve les données de télémétrie que pendant sept jours au maximum.

Cette fonctionnalité est disponible dans le logicielAWS IoT Greengrass Core v1.11.0 et est activée par défaut pour tous les cœurs Greengrass, y compris les cœurs existants. Vous commencez automatiquement à recevoir des données dès que vous passez au logicielAWS IoT Greengrass Core v1.11.0 ou version ultérieure.

Pour plus d'informations sur l'accès ou la gestion des données de télémétrie publiées ou sur la gestion de données de télémétrie publiées, consultezthe section called “S'abonner pour recevoir des données de télémétrie”.

L'agent de télémétrie collecte et publie les mesures système suivantes.

Métriques de télémétrie

Name (Nom)	Description	Source
`SystemMemUsage`	La quantité de mémoire actuellement utilisée par toutes les applications du périphérique principal Greengrass, y compris le système d'exploitation.	Système
`CpuUsage`	La quantité de processeu r actuellement utilisée par toutes les applications du périphérique principal Greengrass, y compris le système d'exploitation.	Système
`TotalNumberOfFDs`	Nombre de descripteurs de fichiers stockés par le système d'exploitation du périphéri que principal Greengrass. Un descripteur de fichier identifie de manière unique un fichier ouvert.	Système
`LambdaOutOfMemory`	Nombre d'exécutions entraînant l'épuisement de la mémoire de la fonction Lambda.	Système

Name (Nom)	Description	Source
`DroppedMessageCount`	Nombre de messages supprimés qui sont destinés àAWS IoT Core.	`GGCloudSpooler` composant du système
`LambdaTimeout`	Nombre de délais d'expiration pour exécuter la fonction Lambda définie par l'utilisateur.	Fonction Lambda et système définis par l'utilisateurAWS Cloud
`LambdaUngracefullyKilled`	Nombre d'exécutions que la fonction Lambda définie par l'utilisateur ne parvient pas à effectuer.	Fonction Lambda et système définis par l'utilisateurAWS Cloud
`LambdaError`	Nombre d'exécutions qui entraînent l'écriture de journaux d'erreurs par la fonction Lambda définie par l'utilisateur.	Fonction Lambda et système définis par l'utilisateurAWS Cloud
`BytesAppended`	Nombre d'octets de données ajoutés au gestionnaire de flux.	`GGStreamManager` composant du système
`BytesUploadedToIoTAnalytics`	Nombre d'octets de données que le gestionnaire de flux exporte vers les canaux dans lesquelsAWS IoT Analytics.	`GGStreamManager` composant du système
`BytesUploadedToKinesis`	Nombre d'octets de données que le gestionnaire de flux exporte vers des flux dans Amazon Kinesis Data Streams.	`GGStreamManager` composant du système

Name (Nom)	Description	Source
`BytesUploadedToIoTSiteWise`	Nombre d'octets de données dans lesquels le gestionnaire de flux exporte vers des propriétés de ressource dans, dansAWS IoT SiteWise,	`GGStreamManager` composant du système
`BytesUploadedToS3ExportTaskExecutor`	Nombre d'octets de données que le gestionnaire de flux exporte vers des objets dans Amazon S3.	`GGStreamManager` composant du système
`BytesUploadedToHTTP`	Nombre d'octets de données que le gestionnaire de flux exporte vers HTTP.	`GGStreamManager` composant du système

Configuration des paramètres de télémétrie

La télémétrie Greengrass utilise les paramètres suivants :

- L'agent de télémétrie agrège les données de télémétrie toutes les heures.
- L'agent de télémétrie publie un message de télémétrie toutes les 24 heures.

> **Note**
>
> Les paramètres ne peuvent pas être modifiés.

Vous pouvez activer ou désactiver la fonction de télémétrie pour un périphérique Greengrass Core. AWS IoT Greengrassutilise des ombres pour gérer la configuration de la télémétrie. Vos modifications prennent effet immédiatement lorsque le noyau est connecté àAWS IoT Core.

L'agent de télémétrie publie des données à l'aide du protocole MQTT avec un niveau de qualité de service (QoS) de 0. Cela signifie qu'il ne confirme pas la livraison et ne tente pas à nouveau de publier. Les messages de télémétrie partagent une connexion MQTT avec d'autres messages pour les abonnements auxquels ils sont destinésAWS IoT Core.

Outre les coûts de liaison de données, le transfert de données du noyau vers leAWS IoT Core serveur est gratuit. Cela est dû au fait que l'agent publie dans une rubriqueAWS réservée. Toutefois, selon votre cas d'utilisation, vous pouvez encourir des frais lorsque vous recevez ou traitez les données.

Prérequis

Les exigences suivantes s'appliquent lorsque vous configurez les paramètres de télémétrie :

- Vous devez utiliser le logicielAWS IoT Greengrass Core v1.11.0 ou ultérieure.

  > **Note**
  >
  > Si vous utilisez une version antérieure et si vous ne voulez pas utiliser la télémétrie, vous n'avez rien à faire.

- Vous devez fournir des autorisations IAM pour mettre à jour l'ombre principale (objet) et pour appeler les API de configuration avant de mettre à jour les paramètres de télémétrie.

  L'exemple de politique IAM suivant vous permet de gérer la configuration de l'ombre et de l'exécution d'un cœur spécifique :

```
{
    "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"
            ]
        }
    ]
}
```

Vous pouvez accorder un accès granulaire ou conditionnel aux ressources, par exemple en utilisant un schéma de* dénomination générique. Pour plus d'informations, consultez la section Ajout et suppression de politiques IAM dans le Guide de l'utilisateur IAM.

Configuration des paramètres de télémétrie (console)

Ce qui suit montre comment mettre à jour les paramètres de télémétrie d'un noyau Greengrass dans laAWS IoT Greengrass console.

1. Dans le volet de navigation de laAWS IoT console, sous Gérer, développez les appareils Greengrass, puis choisissez Groupes (V1).
2. Sous Groupes Greengrass, choisissez votre groupe cible.
3. Sur la page de configuration du groupe, dans la section Présentation, choisissez votre noyau Greengrass.
4. Sur la page de configuration du noyau, choisissez l'onglet Télémétrie.
5. Dans la section Télémétrie de l'état du système, choisissez Configurer.
6. Dans Configurer la télémétrie, sélectionnez Télémétrie pour activer ou désactiver l'état de télémétrie.

> **Important**
>
> Par défaut, la fonctionnalité de télémétrie est activée pour le logicielAWS IoT Greengrass Core v1.11.0 ou version ultérieure.

Les modifications prennent effet au moment de l'exécution. Vous n'avez pas besoin de déployer le groupe.

Configurer les paramètres de télémétrie (CLI)

Dans l'AWS IoT GreengrassAPI, l'`TelemetryConfiguration`objet représente les paramètres de télémétrie d'un noyau Greengrass. Cet objet fait partie de l'`RuntimeConfiguration`objet associé au noyau. Vous pouvez utiliser l'AWS IoT GreengrassAPI ouAWS CLI leAWS SDK pour gérer la télémétrie Greengrass. Les exemples de cette section utilisent leAWS CLI.

Pour vérifier les paramètres de télémétrie

La commande suivante permet d'obtenir les paramètres de télémétrie d'un noyau Greengrass.

- Remplacez *`core-thing-name`*par le nom du noyau cible.

  Pour obtenir le nom de l'objet, utilisez la get-core-definition-versioncommande. La commande renvoie l'ARN de l'objet qui contient le nom de l'objet.

```
aws greengrass get-thing-runtime-configuration --thing-name core-thing-name
```

La commande renvoie un`GetCoreRuntimeConfigurationResponse` objet dans la réponse JSON. Par exemple :

```
{
    "RuntimeConfiguration": {
        "TelemetryConfiguration": {
            "ConfigurationSyncStatus": "OutOfSync",
            "Telemetry": "On"
        }
    }
}
```

Pour configurer les paramètres de télémétrie

La commande suivante met à jour les paramètres de télémétrie d'un noyau Greengrass.

- Remplacez *`core-thing-name`*par le nom du noyau cible.

  Pour obtenir le nom de l'objet, utilisez la get-core-definition-versioncommande. La commande renvoie l'ARN de l'objet qui contient le nom de l'objet.

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\"}}"
```

Les modifications apportées aux paramètres de télémétrie ont été appliquées si`ConfigurationSyncStatus` c'est le cas`InSync`. Les modifications prennent effet au moment de l'exécution. Vous n'avez pas besoin de déployer le groupe.

TelemetryConfiguration objet

L'`TelemetryConfiguration`objet a les propriétés suivantes :

`ConfigurationSyncStatus`

Vérifie si les paramètres de télémétrie sont synchronisés. Vous n'apporterez peut-être aucune modification à cette propriété.

Type : chaîne

Valeurs valides : `InSync` ou `OutOfSync`

`Telemetry`

Active ou désactive la télémétrie. La valeur par défaut est `On`.

Type : chaîne

Valeurs valides : `On` ou `Off`

S'abonner pour recevoir des données de télémétrie

Vous pouvez créer des règles dans Amazon EventBridge qui définissent la manière de traiter les données de télémétrie publiées à partir de l'appareil principal Greengrass. Lorsqu'il EventBridge reçoit les données, il appelle les actions cibles définies dans vos règles. Vous pouvez, par exemple, créer des règles d'événement pour envoyer des notifications, stocker des informations sur les événements, prendre des mesures correctives ou appeler d'autres événements.

Événement de télémétrie

L'événement de changement d'état de déploiement incluant les données de télémétrie utilise le format suivant :

```
{
    "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"
                    }
                ]
            }
        ]
    }
}
```

LeADP tableau contient une liste de points de données agrégés qui possèdent les propriétés suivantes :

`TS`

Obligatoire. Date et heure de regroupement des données.

`NS`

Obligatoire. L'espace de noms du système.

`M`

Obligatoire. La liste des indicateurs. Une métrique contient les propriétés suivantes :

`N`

Nom de la métrique.

`Sum`

La valeur de la métrique agrégée. L'agent de télémétrie ajoute de nouvelles valeurs au total précédent, de sorte que la somme est une valeur toujours croissante. Vous pouvez utiliser l'horodatage pour trouver la valeur d'une agrégation spécifique. Par exemple, pour trouver la dernière valeur agrégée, soustrayez la valeur horodatée précédente de la dernière valeur horodatée.

U

Unité de la valeur métrique.

`ThingName`

Obligatoire. Nom de l'appareil que vous ciblez.

Conditions préalables à la création de EventBridge règles

Avant de créer une EventBridge règle pourAWS IoT Greengrass, vous devez procéder comme suit :

- Familiarisez-vous avec les événements, les règles et les cibles dans EventBridge.
- Créez et configurez les cibles appelées par vos EventBridge règles. Les règles peuvent invoquer de nombreux types de cibles, tels que les flux Amazon Kinesis,AWS Lambda les fonctions, les rubriques Amazon SNS et les files d'attente Amazon SQS.

  Votre EventBridge règle et les cibles associées doivent se trouverRégion AWS là où vous avez créé vos ressources Greengrass. Pour plus d'informations, consultez la section Points de terminaison et quotas de service dans le Références générales AWS.

Pour plus d'informations, consultez Qu'est-ce qu'Amazon EventBridge ? et Démarrez avec Amazon EventBridge dans le Guide de EventBridge l'utilisateur Amazon.

Création d'une règle d'événement pour obtenir des données de télémétrie (console)

Suivez les étapes suivantes pourAWS Management Console créer une EventBridge règle qui reçoit les données de télémétrie publiées par le noyau Greengrass. Cela permet aux serveurs web, aux adresses e-mail et aux autres abonnés à la rubrique de répondre à l'événement. Pour plus d'informations, consultez la section Création d'une EventBridge règle qui se déclenche lors d'un événement à partir d'uneAWS ressource du Guide de EventBridge l'utilisateur Amazon.

1. Ouvrez la EventBridgeconsole Amazon et choisissez Créer une règle.
2. Sous Name and description (Nom et description), entrez un nom et une description pour la règle.
3. Choisissez Bus d'événements et activez la règle sur le bus d'événements sélectionné.
4. Sélectionnez le type de règle et choisissez Règle avec un modèle d'événement.
5. Choisissez Suivant.

6. Dans Event source, sélectionnez AWSévénements ou événements EventBridge partenaires.
7. Dans Exemple d'événement, choisissez AWSÉvénements, puis sélectionnez Greengrass Telemetry Data.
8. Dans Modèle d'événements, effectuez les sélections suivantes :

   a. Pour Event source (Origine de l'événement), choisissez AWSservices (Services).

   b. Pour le AWSservice, choisissez Greengrass.

   c. Pour Type d'événement, choisissez Greengrass Telemetry Data.
9. Choisissez Suivant.
10. Pour Target 1, choisissez AWSservice.
11. Pour Sélectionner une cible, choisissez SQS Queue.
12. Dans Queue, choisissez votre fonction.

Création d'une règle d'événement pour obtenir des données de télémétrie (CLI)

Suivez les étapes suivantes pourAWS CLI créer une EventBridge règle qui reçoit les données de télémétrie publiées par le noyau Greengrass. Cela permet aux serveurs web, aux adresses e-mail et aux autres abonnés à la rubrique de répondre à l'événement.

1. Créez la règle .

   - Remplacez *thing-name* par le nom d'objet du noyau.

     Pour obtenir le nom de l'objet, utilisez la get-core-definition-versioncommande. La commande renvoie l'ARN de l'objet qui contient le nom de l'objet.

   ```
   aws events put-rule \
     --name TestRule \
     --event-pattern "{\"source\": [\"aws.greengrass\"], \"detail\": {\"ThingName\":
    [\"thing-name\"]}}"
   ```

   Les propriétés qui sont omises dans le modèle sont ignorées.
2. Ajoutez la rubrique en tant que cible de règle. L'exemple suivant utilise Amazon SQS mais vous pouvez configurer d'autres types de cibles.

   - Remplacez *queue-arn* par l'ARN de votre file d'attente Amazon SQS.

```
aws events put-targets \
  --rule TestRule \
  --targets "Id"="1","Arn"="queue-arn"
```

> **Note**
>
> Pour autoriser Amazon EventBridge à appeler votre file d'attente cible, vous devez ajouter une stratégie basée sur les ressources à votre rubrique. Pour plus d'informations, consultez les autorisations Amazon SQS dans le guide de EventBridge l'utilisateur Amazon.

Pour plus d'informations, consultez la section Événements et modèles d'événements EventBridge dans le Guide de EventBridge l'utilisateur Amazon.

Résolution des problèmesAWS IoT Greengrass de télémétrie

Consultez les informations suivantes pour essayer de résoudre les problèmes liés à la configuration de laAWS IoT Greengrass télémétrie.

Erreur : la réponse contient "ConfigurationStatus« :"OutOfSync "après avoir exécuté la get-thing-runtime-configuration commande

Des solutions :

- Le serviceAWS IoT Device Shadow met du temps à traiter les mises à jour de configuration d'exécution et à fournir les mises à jour au périphérique principal Greengrass. Vous pouvez attendre et vérifier ultérieurement si les paramètres de télémétrie sont synchronisés.
- Vérifiez que votre appareil principal est en ligne.
- Activez Amazon CloudWatch Logs inAWS IoT Core pour surveiller l'ombre.
- Utilisez AWS IoTdes indicateurs pour surveiller vos activités.

Appel de l'API de contrôle de santé locale

AWS IoT Greengrasscontient une API HTTP locale qui fournit un instantané de l'état actuel des processus de travail locaux démarrés parAWS IoT Greengrass. Cet instantané inclut des fonctions

Lambda définies par l'utilisateur et des fonctions Lambda système. Les fonctions Lambda du système font partie de laAWS IoT GreengrassLogiciel Core. Ils s'exécutent comme des processus de travail local sur l'appareil principal et gèrent les opérations telles que le routage des messages, la synchronisation du shadow local et la détection automatique d'adresse IP automatique.

L'API de contrôle de santé prend en charge les demandes suivantes :

- Envoyez une demandeGETde la demandeobtenir des informations de santé pour tous les travailleurs.
- Envoyez une demandePOSTde la demandeobtenir des informations sur la santé de certains travailleurs.

Les demandes sont envoyées localement sur l'appareil et ne nécessitent pas de connexion Internet.

Obtenir des informations de santé pour tous les travailleurs

Envoyez une demandeGETdemande pour obtenir des informations de santé sur tous les travailleurs en cours d'exécution.

- Remplacez*port*avec le numéro de port de l'IPC.

```
GET http://localhost:port/2016-11-01/health/workers
```

`port`

Numéro de port de l'IPC.

La valeur peut varier entre 1 024 et 65 535. La valeur par défaut est 8 000.

Pour modifier ce numéro de port, vous pouvez mettre à jour leggDaemonPortpropriété dans leconfig.jsondans le fichier. Pour plus d'informations, consultez Fichier de configuration de AWS IoT Greengrass Core.

Exemple de demande

L'exemple suivantcurlrequest obtient des informations de santé pour tous les travailleurs.

```
curl http://localhost:8000/2016-11-01/health/workers
```

Réponse JSON

Cette demande renvoie un tableau deRenseignements personnels sur la santéobjets.

Exemple de réponse

L'exemple de réponse suivant répertorie les objets d'informations d'intégrité pour tous les processus de travail qui ont été démarrés parAWS 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"
    },
    ...
]
```

Obtenir des informations sur la santé de certains travailleurs

Envoyez une demandePOSTdemande d'obtenir des informations sur la santé de certains travailleurs. Remplacez*port*avec le numéro de port de l'IPC. La valeur par défaut est 8 000.

```
POST http://localhost:port/2016-11-01/health/workers
```

Exemple de demande

L'exemple suivantcurlrequest obtient des informations de santé pour des travailleurs spécifiques.

```
curl --data "@body.json" http://localhost:8000/2016-11-01/health/workers
```

Voici un exemple`body.json`corps de la demande :

```
{
    "FuncArns": [
        "arn:aws:lambda:::function:GGShadowService:1",
        "arn:aws:lambda:us-west-2:123456789012:function:my-lambda-function:3"
    ]
}
```

Le corps de la demande contient un`FuncArns`tableau.

`FuncArns`

Liste d'ARN (Amazon Resource Name) pour les fonctions Lambda qui représentent les travailleurs cibles.

- Pour les fonctions Lambda définies par l'utilisateur, spécifiez l'ARN de la version actuellement déployée. Si vous avez ajouté des fonctions Lambda au groupe à l'aide d'un alias ARN, vous pouvez utiliser la requête GET pour obtenir tous les travailleurs, puis choisir les ARN que vous souhaitez interroger.
- Pour les fonctions Lambda du système, spécifiez l'ARN de la fonction Lambda correspondante. Pour plus d'informations, consultez the section called “Fonctions Lambda du système”.

Type : tableau de chaînes

Longueur minimale : 1

Longueur maximale : Nombre total de travailleurs démarrés par AWS IoT Greengrass sur l'appareil principal.

Réponse JSON

Cette demande renvoie un`Workers`tableau et un`InvalidArns`tableau.

`Workers`

Une liste d'objets d'informations de santé pour les travailleurs spécifiés.

Type : tableauobjets sur la santé

`InvalidArns`

Liste des ARN de fonction non valides, y compris les ARN de fonction auxquels aucun programme de travail n'est associé.

Type : tableau de chaînes

Exemple de réponse

Les exemples de listes de réponses suivantsobjets sur la santépour les travailleurs spécifiés.

```
{
    "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"
    ]
}
```

Cette demande renvoie les erreurs suivantes :

400 Demande non valide

Le corps de la demande est incorrect. Pour résoudre ce problème, utilisez le format suivant et renvoyez la demande :

```
{"FuncArns":["function-1-arn","function-2-arn"]}
```

400 demandes dépassant le nombre maximum de travailleurs

Le nombre d'ARN spécifiés dans le`FuncArns`du tableau dépasse le nombre d'employés.

Renseignements personnels sur la santé

Un objet d'information sur la santé contient les propriétés suivantes :

`FuncArn`

ARN de la fonction Lambda du système qui représente l'application de travail.

Type: `string`

`WorkerId`

ID du travail. Cette propriété peut être utile pour le débogage. L`eruntime.log`et les journaux de la fonction Lambda contiennent l'ID du worker. Cette propriété peut donc être particulièrement utile pour déboguer une fonction Lambda à la demande qui fait tourner plusieurs instances.

Type: `string`

`ProcessId`

ID de processus (PID) du processus de travail.

Type: `int`

`WorkerState`

État du travail.

Type: `string`

Les états de travail possibles sont les suivants :

`Working`

Traitement d'un message.

`Waiting`

J'attends un message. S'applique aux fonctions Lambda de longue durée exécutées en tant que démon ou processus autonome.

`Starting`

Spun up, pour commencer.

`FailedInitialization`

Impossible d'initialiser.

`Terminated`

Arrêté par le démon Greengrass

`NotStarted`

Impossible de démarrer, nouvelle tentative de démarrage.

`Initialized`

Initialisation réussie.

Fonctions Lambda du système

Vous pouvez demander des informations d'intégrité pour les fonctions Lambda système suivantes :

`GGCloudSpooler`

Gère la file d'attente pour les messages MQTT qui ontAWS IoT Corecomme source ou cible.

ARN : `arn:aws:lambda:::function:GGCloudSpooler:1`

`GGConnManager`

Acheminement des messages MQTT entre le noyau Greengrass et les appareils clients.

ARN : `arn:aws:lambda:::function:GGConnManager`

`GGDeviceCertificateManager`

Écoute leAWS IoTshadow pour les modifications apportées aux points de terminaison IP du cœur et génère le certificat côté serveur utilisé par GGConnManager pour l'authentification mutuelle.

ARN : `arn:aws:lambda:::function:GGDeviceCertificateManager`

`GGIPDetector`

Gère la détection automatique d'adresse IP qui permet aux appareils du groupe Greengrass de détecter votre appareil Greengrass principal. Ce service n'est pas applicable lorsque vous fournissez des adresses IP manuellement.

ARN : `arn:aws:lambda:::function:GGIPDetector:1`

`GGSecretManager`

Gère le stockage sécurisé des secrets locaux et l'accès par Lambda et connecteurs définis par l'utilisateur.

ARN : `arn:aws:lambda:::function:GGSecretManager:1`

`GGShadowService`

Gère les ombres locales pour les appareils clients.

ARN : `arn:aws:lambda:::function:GGShadowService`

`GGShadowSyncManager`

Synchronise les ombres locales avec leAWS Cloudpour l'appareil principal et les appareils clients, si`syncShadow`est définie sur`true`.

ARN : `arn:aws:lambda:::function:GGShadowSyncManager`

`GGStreamManager`

Traite les flux de données localement et effectue des exportations automatiques versAWS Cloud.

ARN : `arn:aws:lambda:::function:GGStreamManager:1`

`GGTES`

Service local d'échange de jetons qui récupère les informations d'identification IAM définies dans le rôle de groupe Greengrass utilisé par le code local pour accéderAWSServices .

ARN : `arn:aws:lambda:::function:GGTES`

Balisage de vos ressources AWS IoT Greengrass

Les balises peuvent vous aider à organiser et à gérer vos groupes AWS IoT Greengrass. Vous pouvez utiliser des balises pour attribuer des métadonnées à des groupes, des déploiements en masse, et des noyaux, des appareils et d'autres ressources qui sont ajoutés aux groupes. Les tags peuvent également être utilisés dans les politiques IAM pour définir un accès conditionnel à vos ressources Greengrass.

> **Note**
>
> Actuellement, les balises de ressource Greengrass ne sont pas prises en charge pour les groupes de facturation ou les rapports de répartition des coûts AWS IoT.

Principes de base des étiquettes

Les balises vous permettent de classer vos ressources AWS IoT Greengrass, par exemple, par objectif, par propriétaire et par environnement. Lorsque vous avez de nombreuses ressources de même type, vous pouvez identifier rapidement une ressource spécifique en fonction des balises qui lui sont associées. Une balise est constituée d'une clé et d'une valeur facultative que vous définissez. Nous vous recommandons de concevoir un ensemble de clés de balise pour chaque type de ressource. L'utilisation d'un ensemble de clés de balise cohérent facilite la gestion de vos ressources. Par exemple, vous pouvez définir un ensemble de balises pour vos groupes qui vous permet de suivre l'emplacement usine de vos principaux appareils. Pour plus d'informations, consultez Stratégies d'étiquette AWS.

Prise en charge du balisage dans la console AWS IoT

Vous pouvez créer, afficher et gérer des balises pour vos ressources `Group` dans la console AWS IoT. Avant de créer des balises, tenez compte des restrictions liées aux balises. Pour de plus amples informations, veuillez consulter Conventions de dénomination et d'utilisation des balises dans le Référence générale d'Amazon Web Services.

Pour attribuer des balises lorsque vous créez un groupe

Vous pouvez attribuer des balises à un groupe lorsque vous créez le groupe. Choisissez Ajouter une nouvelle balise dans la section Balises pour afficher les champs de saisie des balises.

Pour afficher et gérer les balises à partir de la page de configuration du groupe

Vous pouvez afficher et gérer les balises depuis la page de configuration du groupe en choisissant Afficher les paramètres. Dans la section Balises du groupe, choisissez Gérer les balises pour ajouter, modifier ou supprimer des balises de groupe.

Prise en charge du balisage dans l'API AWS IoT Greengrass

Vous devez utiliser l'API AWS IoT Greengrass afin de créer et gérer des balises pour les ressources AWS IoT Greengrass qui prennent en charge le balisage. Avant de créer des balises, tenez compte des restrictions liées aux balises. Pour de plus amples informations, veuillez consulter Conventions de dénomination et d'utilisation des balises dans le Référence générale d'Amazon Web Services.

- Pour ajouter des balises lors de la création de ressources, définissez-les dans la propriété `tags` de la ressource.
- Pour ajouter des balises après la création d'une ressource ou pour mettre à jour des valeurs de balise, utilisez l'action `TagResource`.
- Pour supprimer des balises d'une ressource, utilisez l'action `UntagResource`.
- Pour récupérer les balises qui sont associées à une ressource, utilisez l'action `ListTagsForResource` ou obtenez la ressource et inspectez sa propriété `tags`.

Le tableau suivant répertorie les ressources que vous pouvez baliser dans l'API AWS IoT Greengrass et ses actions `Create` et `Get` correspondantes.

Ressource	Création	Get
`Group`	`CreateGroup`	`GetGroup`
`ConnectorDefinition`	`CreateConnectorDefinition`	`GetConnectorDefinition`
`CoreDefinition`	`CreateCoreDefinition`	`GetCoreDefinition`
`DeviceDefinition`	`CreateDeviceDefinition`	`GetDeviceDefinition`

Ressource	Création	Get
FunctionDefinition	CreateFunctionDefinition	GetFunctionDefinition
LoggerDefinition	CreateLoggerDefinition	GetLoggerDefinition
ResourceDefinition	CreateResourceDefinition	GetResourceDefinition
SubscriptionDefinition	CreateSubscriptionDefinition	GetSubscriptionDefinition
BulkDeployment	StartBulkDeployment	GetBulkDeploymentStatus

Utilisez les actions suivantes afin de répertorier et de gérer des balises pour les ressources qui prennent en charge le balisage :

- TagResource. Ajoute des balises à une ressource. Egalement utilisé pour modifier la valeur de la paire clé-valeur de la balise.
- ListTagsForResource. Répertorie les balises d'une ressource.
- UntagResource. Supprime les balises d'une ressource.

Vous pouvez ajouter ou supprimer des balises sur une ressource à tout moment. Pour modifier la valeur d'une clé de balise, ajoutez une balise à la ressource qui définit la même clé et la nouvelle valeur. La nouvelle valeur remplace l'ancienne valeur. Vous pouvez définir la valeur d'une balise sur une chaîne vide, mais pas sur null.

Lorsque vous supprimez une ressource, les balises associées à celle-ci sont également supprimées.

> **Note**
>
> Ne confondez pas les balises de ressources avec les attributs que vous pouvez attribuer aux objets AWS IoT. Bien que les noyaux Greengrass soient des objets AWS IoT, les balises de

ressource qui sont décrites dans cette rubrique sont attachées à une `CoreDefinition`, et non à l'objet principal.

Utilisation des balises avec des politiques IAM

Dans vos politiques IAM, vous pouvez utiliser des balises de ressources pour contrôler l'accès et les autorisations des utilisateurs. Par exemple, des stratégies peuvent permettre aux utilisateurs de créer uniquement les ressources qui comportent une balise spécifique. Les stratégies peuvent également empêcher les utilisateurs de créer ou de modifier des ressources qui ont des balises spécifiques. Vous pouvez baliser des ressources au cours de la création (baliser lors de la création) afin de ne pas avoir à exécuter des scripts de balisage personnalisés plus tard. Lorsque de nouveaux environnements sont lancés avec des balises, les autorisations IAM correspondantes sont appliquées automatiquement.

Les clés de contexte de condition et les valeurs suivantes peuvent être utilisées dans l'élément `Condition` (également nommé bloc de `Condition`) de la stratégie.

`greengrass:ResourceTag/`*`tag-key`*`: `*`tag-value`*

Accorder ou refuser aux utilisateurs des actions sur des ressources ayant des balises spécifiques.

`aws:RequestTag/`*`tag-key`*`: `*`tag-value`*

Exiger qu'une balise spécifique soit utilisée (ou ne soit pas utilisée) lorsque vous effectuez des demandes d'API pour créer ou modifier des balises sur une ressource balisable.

`aws:TagKeys: [`*`tag-key`*`, ...]`

Exiger qu'un ensemble spécifique de clés de balise soit utilisé (ou ne soit pas utilisé) lorsque vous effectuez une demande d'API pour créer ou modifier une ressource balisable.

Les clés de contexte de condition et les valeurs peuvent être utilisées uniquement sur les actions AWS IoT Greengrass qui agissent sur une ressource balisable. Ces actions prennent la ressource comme un paramètre obligatoire. Par exemple, vous pouvez définir un accès conditionnel sur `GetGroupVersion`. Vous ne pouvez pas définir d'accès conditionnel sur `AssociateServiceRoleToAccount` car aucune ressource balisable (par exemple, un groupe, une définition de noyau ou une définition d'appareil) n'est référencée dans la demande.

Pour plus d'informations, consultez la section Contrôle de l'accès à l'aide de balises et la référence de politique IAM JSON dans le guide de l'utilisateur IAM. La référence de politique JSON inclut une syntaxe détaillée, des descriptions et des exemples des éléments, des variables et de la logique d'évaluation des politiques JSON dans IAM.

Exemple de politiques IAM

L'exemple de stratégie suivant applique des autorisations reposant sur les balises qui limitent un utilisateur bêta aux actions sur les ressources bêta uniquement.

- La première instruction permet à un utilisateur IAM d'agir uniquement sur des ressources dotées de la balise env=beta.
- La deuxième instruction empêche un utilisateur IAM de supprimer la balise env=beta des ressources. Cela permet d'éviter à l'utilisateur de supprimer son propre accès.

> **Note**
>
> Si vous utilisez des balises pour contrôler l'accès aux ressources, vous devez également gérer les autorisations qui permettent aux utilisateurs d'ajouter ou de supprimer des balises dans ces mêmes ressources. Sinon, dans certains cas, il peut être possible pour les utilisateurs de contourner vos restrictions et d'obtenir l'accès à une ressource en modifiant ses balises.

```
{
    "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"
                }
            }
        }
    ]
}
```

Pour permettre aux utilisateurs de baliser lors de la création, vous devez leur accorder les autorisations appropriées. L'exemple de stratégie suivant inclut la `"aws:RequestTag/env": "beta"` condition sur les actions `greengrass:TagResource` et `greengrass:CreateGroup`, ce qui permet aux utilisateurs de créer un groupe uniquement s'ils balisent ce dernier avec env=beta. Cela oblige vraiment les utilisateurs à baliser les nouveaux groupes.

```
{
    "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"
                }
            }
        }
    ]
}
```

L'extrait suivant vous montre comment spécifier plusieurs valeurs de balise pour une clé de balise donnée en les plaçant dans une liste :

```
"StringEquals" : {
    "greengrass:ResourceTag/env" : ["dev", "test"]
}
```

Consultez aussi

- Balisage des ressources AWS dans le Référence générale d'Amazon Web Services

Prise en charge de AWS CloudFormation pour AWS IoT Greengrass

AWS CloudFormation est un service qui peut vous aider à créer, gérer et répliquer vos ressources AWS. Vous pouvez utiliser AWS CloudFormation des modèles pour définir AWS IoT Greengrass les groupes ainsi que les appareils clients, les abonnements et les autres composants que vous souhaitez déployer. Pour obtenir un exemple, consultez the section called “Exemple de modèle ”.

Les ressources et l'infrastructure que vous générez à partir d'un modèle sont nommées pile. Vous pouvez définir l'ensemble de vos ressources dans un modèle ou faire référence à des ressources à partir d'autres piles. Pour plus d'informations sur les AWS CloudFormation modèles et les fonctionnalités, voir Qu'est-ce que c'est AWS CloudFormation ? dans le guide de AWS CloudFormation l'utilisateur.

Création de ressources

Les modèles AWS CloudFormation sont des documents au format JSON ou YAML qui décrivent les propriétés et les relations des ressources AWS. Les ressources AWS IoT Greengrass suivantes sont prises en charge :

- Groups
- Cœurs
- Appareils clients (appareils)
- Fonctions Lambda
- Connecteurs
- Ressources (local, Machine Learning et secret)
- Abonnements
- Enregistreurs (configurations de journalisation)

Dans les modèles AWS CloudFormation, la structure et la syntaxe des ressources Greengrass sont basées sur l'API AWS IoT Greengrass. Par exemple, le modèle d'exemple associe un niveau supérieur `DeviceDefinition` à un `DeviceDefinitionVersion` contenant un appareil client individuel. Pour plus d’informations, consultez the section called “Présentation du modèle d'objet de groupe”.

La référence aux types de AWS IoT Greengrass ressources dans le guide de l'AWS CloudFormationutilisateur décrit les ressources Greengrass que vous pouvez gérer. AWS CloudFormation Lorsque vous utilisez des modèles AWS CloudFormation pour créer des ressources Greengrass, nous vous recommandons de les gérer uniquement à partir de AWS CloudFormation. Par exemple, vous devez mettre à jour votre modèle si vous souhaitez ajouter, modifier ou supprimer un appareil (au lieu d'utiliser l'AWS IoT GreengrassAPI ou la AWS IoT console). Cela vous permet d'utiliser la restauration et d'autres fonctionnalités de gestion des changements AWS CloudFormation. Pour plus d'informations sur la création et la gestion de vos ressources et de vos piles, consultez la section Utilisation AWS CloudFormation des piles dans le guide de l'AWS CloudFormationutilisateur.

Pour une présentation expliquant comment créer et déployer des AWS IoT Greengrass ressources dans un AWS CloudFormation modèle, consultez Automatiser la AWS IoT Greengrass configuration avec l'Internet des objets AWS CloudFormation sur le blog AWS officiel.

Déploiement des ressources

Après avoir créé une AWS CloudFormation pile contenant la version de votre groupe, vous pouvez utiliser la AWS IoT console AWS CLI or pour la déployer.

> **Note**
>
> Pour déployer un groupe, un rôle de service Greengrass doit être associé à votre. Compte AWS Le rôle de service permet d'accéder AWS IoT Greengrass à vos ressources dans AWS Lambda et à d'autres AWS services. Ce rôle devrait exister si vous avez déjà déployé un groupe Greengrass dans le passé. Région AWS Pour plus d'informations, consultez the section called “Rôle de service Greengrass”.

Pour déployer le groupe (AWS CLI)

- Exécutez la commande create-deployment.

```
aws greengrass create-deployment --group-id GroupId --group-version-
id GroupVersionId --deployment-type NewDeployment
```

> **Note**
>
> L'instruction `CommandToDeployGroup` de l'exemple de modèle montre comment générer la commande avec vos ID de groupe et de version de groupe lorsque vous créez une pile.

Pour déployer le groupe (console)

1. Dans le volet de navigation de la AWS IoT console, sous Gérer, développez les appareils Greengrass, puis choisissez Groups (V1).
2. Choisissez votre groupe.
3. Sur la page de configuration du groupe, choisissez Deploy.

Exemple de modèle

L'exemple de modèle suivant crée un groupe Greengrass qui contient un noyau, un appareil client, une fonction, un enregistreur, un abonnement et deux ressources. Pour ce faire, le modèle suit le modèle d'objet de l'API AWS IoT Greengrass. Par exemple, les appareils clients que vous souhaitez ajouter au groupe sont contenus dans une `DeviceDefinitionVersion` ressource associée à une `DeviceDefinition` ressource. Pour ajouter les appareils au groupe, la version de groupe fait référence à l'ARN de la `DeviceDefinitionVersion`.

Le modèle inclut des paramètres qui vous permettent de spécifier les ARN du certificat pour le cœur et le périphérique, ainsi que l'ARN de version de la fonction Lambda source (qui est AWS Lambda une ressource). Il utilise les fonctions intrinsèques `Ref` et `GetAtt` afin de référencer les ID, les ARN et d'autres attributs nécessaires pour créer les ressources Greengrass.

Le modèle définit également deux AWS IoT appareils (objets), qui représentent le périphérique principal et le périphérique client ajoutés au groupe Greengrass.

Après avoir créé la pile avec vos ressources Greengrass, vous pouvez utiliser la console AWS CLI ou la AWS IoT console pour déployer le groupe.

> **Note**
>
> L'instruction `CommandToDeployGroup` de l'exemple montre comment sortir une interface de ligne de commande create-deployment complète que vous pouvez utiliser pour déployer votre groupe.

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
```

Région AWS prises en charge

À l'heure actuelle, vous ne pouvez créer et gérer AWS IoT Greengrass des ressources que dans les Région AWSdomaines suivants :

- USA Est (Ohio)
- USA Est (Virginie du Nord)
- USA Ouest (Oregon)
- Asie-Pacifique (Mumbai)
- Asie-Pacifique (Séoul)
- Asie-Pacifique (Singapour)
- Asie-Pacifique (Sydney)
- Asia Pacific (Tokyo)
- Chine (Beijing)
- Europe (Francfort)
- Europe (Irlande)
- Europe (Londres)

- AWS GovCloud (US-Ouest)

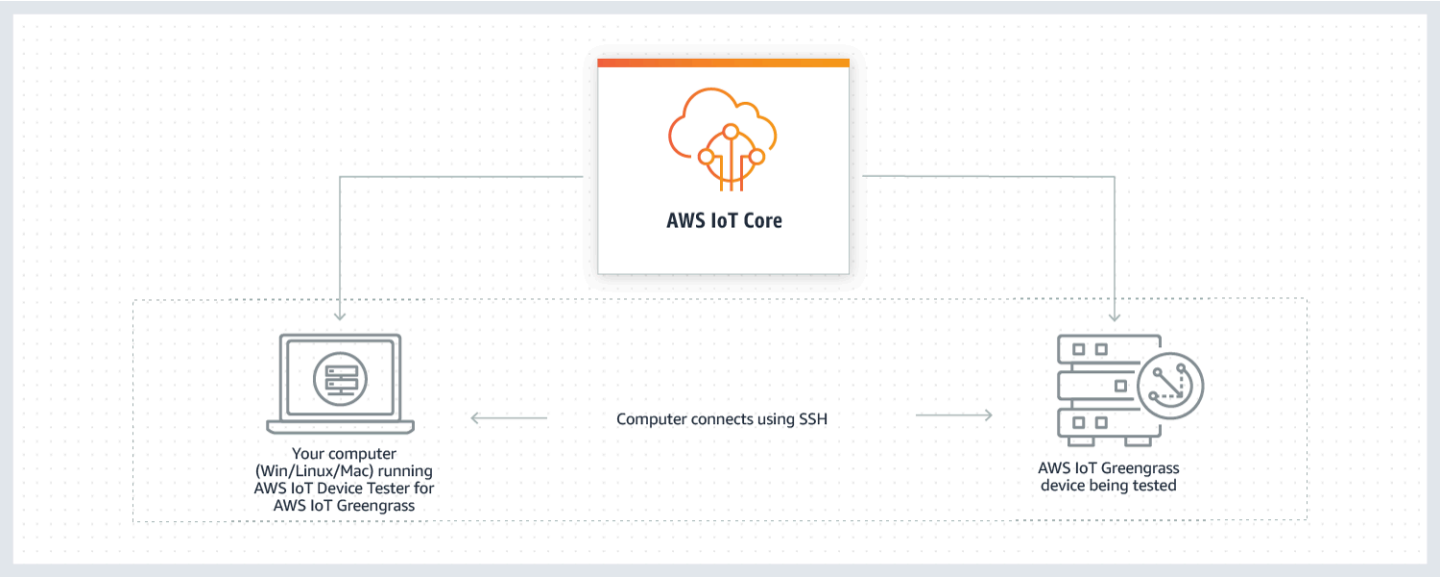
Utilisation de AWS IoT Device Tester pour AWS IoT Greengrass V1

AWS IoT Device Tester (IDT) est un framework de test téléchargeable qui vous permet de valider les appareils IoT. Étant AWS IoT Greengrass Version 1 donné qu'IDT est passé en mode maintenance, IDT AWS IoT Greengrass V1 ne génère plus de rapports de qualification signés. Vous ne serez plus en mesure de qualifier les nouveaux AWS IoT Greengrass V1 appareils à répertorier dans le catalogue des AWS Partner appareils dans le cadre du programme de qualification des AWS appareils. Cependant, vous pouvez continuer à utiliser IDT AWS IoT Greengrass V1 pour tester vos appareils Greengrass V1. Nous vous recommandons d'utiliser IDT pour qualifier et AWS IoT Greengrass V2 répertorier les appareils Greengrass dans AWS Partner le catalogue des appareils.

IDT for AWS IoT Greengrass s'exécute sur votre ordinateur hôte (Windows, macOS ou Linux) connecté à l'appareil à tester. L'outil exécute des tests et regroupe les résultats. Il fournit également une interface de ligne de commande pour gérer le processus de test.

AWS IoT Greengrass suite de qualifications

Utilisez IDT AWS IoT Greengrass pour vérifier que le logiciel AWS IoT Greengrass Core s'exécute sur votre matériel et peut communiquer avec le AWS Cloud. Il effectue également end-to-end des tests avec AWS IoT Core. Par exemple, il vérifie que votre appareil peut envoyer et recevoir des messages MQTT et les traiter correctement.



AWS IoT Device Tester AWS IoT Greengrass organise les tests en utilisant les concepts de suites de tests et de groupes de tests.

- Une suite de tests est l'ensemble des groupes de tests utilisés pour vérifier qu'un appareil fonctionne avec des versions particulières de AWS IoT Greengrass.
- Un groupe de tests est l'ensemble de tests individuels liés à une fonctionnalité particulière, tels que les déploiements de groupe Greengrass et la messagerie MQTT.

Pour plus d’informations, consultez Utilisez IDT pour exécuter leAWS IoT Greengrasssuite de qualification.

Suites de tests personnalisées

À partir de IDT v4.0.0, IDT for AWS IoT Greengrass combine une configuration standardisée et un format de résultat avec un environnement de suites de tests qui vous permet de développer des suites de tests personnalisées pour vos appareils et leurs logiciels. Vous pouvez ajouter des tests personnalisés pour votre propre validation interne ou les fournir à vos clients pour la vérification des appareils.

La façon dont un rédacteur de tests configure une suite de tests personnalisée détermine les configurations de paramètres requises pour exécuter des suites de tests personnalisées. Pour plus d'informations, voir Utilisez IDT pour développer et exécuter vos propres suites de tests.

Versions prises en charge de AWS IoT Device Tester pour AWS IoT Greengrass V1

Étant AWS IoT Greengrass Version 1 donné qu'IDT est passé en mode maintenance, IDT AWS IoT Greengrass V1 ne génère plus de rapports de qualification signés. Nous vous recommandons d'utiliser IDT pourAWS IoT Greengrass V2.

Pour plus d'informations sur IDT pour la AWS IoT Greengrass V2, consultez la section Utilisation du testeur de AWS IoT périphériques AWS IoT Greengrass V2 dans le guide du AWS IoT Greengrass V2 développeur.

> **Note**
>
> Vous recevez une notification lorsque vous démarrez une série de tests si IDT pour AWS IoT Greengrass n'est pas compatible avec la version de AWS IoT Greengrass que vous utilisez.

En téléchargeant le logiciel, vous acceptez le Contrat de licence AWS IoT Device Tester.

Versions IDT non prises en charge pour pour AWS IoT Greengrass

Cette rubrique répertorie les versions non prises en charge d'IDT pour AWS IoT Greengrass. Les versions non prises en charge ne reçoivent pas de corrections de bogues ou de mises à jour. Pour plus d'informations, consultez the section called "Politique de prise en charge d'AWS IoT Device Tester pour AWS IoT Greengrass V1".

IDT v4.4.1 pour les AWS IoT Greengrass versions v1.11.6, v1.10.5

Notes de mise à jour :

- Vous permet de valider et de qualifier les appareils exécutant les logiciels AWS IoT Greengrass principaux v1.11.6 et v1.10.5.
- Contient des corrections de bugs mineurs.

Versions de suite de tests :

`GGQ_1.3.1`

- Publié le 2021.12.20

IDT v4.1.0 pour les AWS IoT Greengrass versions v1.11.4, v1.10.4

Notes de mise à jour :

- Vous permet de valider et de qualifier les appareils exécutant les logiciels AWS IoT Greengrass principaux v1.11.4 et v1.10.4.
- Résout un problème à cause duquel les journaux affichés lors d'un test utilisaient des balises redondantes.

Versions de suite de tests :

`GGQ_1.3.0`

- Publié le 23 juin 2021
- Ajoute de nouvelles tentatives pour les appels d'API à Lambda, IAM, AWS STS et pour améliorer la gestion des problèmes de limitation ou de serveur.

- Ajoute le support de Python 3.8 aux scénarios de test ML et Docker.

IDT v4.0.2 pour les AWS IoT Greengrass versions v1.11.1, v1.11.0, v1.10.3

Notes de mise à jour :

- Correction d'un problème en raison duquel IDT masquait les erreurs HSI (Hardware Security Integration).
- Vous permet de développer et d'exécuter vos suites de tests personnalisées à l'aide de AWS IoT Device Tester pourAWS IoT Greengrass. Pour plus d'informations, consultez Utilisez IDT pour développer et exécuter vos propres suites de tests.
- Fournit des applications IDT signées par code pour macOS et Windows. Sous macOS, si un message d'avertissement de sécurité s'affiche, vous devrez peut-être accorder une exception de sécurité pour IDT. Pour plus d'informations, consultez Exception de sécurité sur macOS.

> **Note**
>
> AWS IoT Greengrassne fournit pas de Dockerfile ou d'image Docker pour la version 1.11.1 du logiciel principal. AWS IoT Greengrass Pour tester la qualification Docker de votre appareil, utilisez une version antérieure du logiciel AWS IoT Greengrass principal.

IDT v3.2.0 pour les AWS IoT Greengrass versions v1.11.0, v1.10.1, v1.10.0

Notes de mise à jour :

- Par défaut, IDT exécute uniquement les tests requis pour la qualification. Pour bénéficier de fonctionnalités supplémentaires, vous pouvez modifier le `device.json`fichier.
- Ajout d'un numéro de port `device.json` que vous pouvez configurer pour les connexions SSH.
- Docker ne prend en charge que le gestionnaire de flux et l'apprentissage automatique (ML) sans conteneurisation. Container, Docker et Hardware Security Integration (HSI) ne sont pas disponibles pour les appareils Docker.
- Nous avons fusionné `device-ml.json` et avons `device-hsm.json` intégré`device.json`.

IDT v3.1.3 pour les AWS IoT Greengrass versions : v1.10.x, v1.9.x, v1.8.x

Notes de mise à jour :

- Ajout de la prise en charge de la qualification des fonctions ML pour AWS IoT Greengrass v1.10.x et v1.9.x. Vous pouvez désormais utiliser IDT pour valider le fait que vos appareils peuvent effectuer des inférences ML localement avec des modèles stockés et formés dans le cloud.
- `--stop-on-first-failure` a été ajouté pour la commande `run-suite`. Vous pouvez utiliser cette option pour configurer IDT afin qu'il cesse de fonctionner lors du premier échec. Nous vous recommandons d'utiliser cette option lors de l'étape de débogage au niveau des groupes de tests.
- Ajout d'une vérification de la dérive de l'horloge pour les tests MQTT afin de garantir que le périphérique testé utilise l'heure système correcte. Le temps utilisé doit se situer dans un intervalle de temps acceptable.
- `--update-idt` a été ajouté pour la commande `run-suite`. Vous pouvez utiliser cette option pour définir la réponse à l'invite de mise à jour d'IDT.
- `--update-managed-policy` a été ajouté pour la commande `run-suite`. Vous pouvez utiliser cette option pour définir la réponse à l'invite de mise à jour de la politique gérée.
- Ajout d'une correction de bogue pour les mises à jour automatiques des versions de la suite de tests IDT. Le correctif garantit qu'IDT peut exécuter les dernières suites de tests disponibles pour votre AWS IoT Greengrass version.

IDT v3.0.1 pour AWS IoT Greengrass

Notes de mise à jour :

- Ajout de la prise en charge pour AWS IoT Greengrass version 1.10.1.
- Mises à jour automatiques des versions de la suite de tests IDT. IDT peut télécharger les dernières suites de tests disponibles pour votre version de AWS IoT Greengrass. Avec cette fonctionnalité :
  - Les suites de tests sont versionnées à l'aide d'un format *`major`*`.`*`minor`*`.`*`patch`*. La version initiale de la suite de tests est `GGQ_1.0.0`.
  - Vous pouvez télécharger de nouvelles suites de tests de manière interactive dans l'interface de ligne de commande ou définir l'indicateur `upgrade-test-suite` lorsque vous démarrez IDT.

  Pour plus d'informations, consultez the section called “Versions de la suite de tests”.

- Ajouté `list-supported-products`. Vous pouvez utiliser cette commande pour répertorier les versions de AWS IoT Greengrass et de suites de tests prises en charge par la version installée d'IDT.
- Ajouté `list-test-cases`. Vous pouvez utiliser cette commande pour répertorier les scénarios de test disponibles dans un groupe de test.
- `test-id` a été ajouté pour la commande `run-suite`. Vous pouvez utiliser cette option pour exécuter des scénarios de test individuels dans un groupe de test.

IDT v2.3.0 pour AWS IoT Greengrass v1.10, v1.9.x et v1.8.x

Lors des tests sur un appareil physique, les AWS IoT Greengrass v1.10, v1.9.x et v1.8.x sont pris en charge.

Lors d'un test dans un conteneur Docker, les AWS IoT Greengrass v1.10 et v1.9.x sont pris en charge.

Notes de mise à jour :

- Ajout de la prise en charge de the section called “Exécuter AWS IoT Greengrass dans un conteneur Docker”. Vous pouvez désormais utiliser IDT pour qualifier et valider que vos appareils peuvent exécuter AWS IoT Greengrass dans un conteneur Docker.
- Ajout d'une politique AWS gérée (`AWSIoTDeviceTesterForGreengrassFullAccess`) qui définit les autorisations requises pour exécuter AWS IoT Device Tester. Si les nouvelles versions nécessitent des autorisations supplémentaires, AWS ajoutez-les à cette politique gérée afin de ne pas avoir à mettre à jour vos autorisations IAM.
- Des vérifications ont été introduites pour valider que votre environnement (par exemple, la connectivité des appareils et la connectivité Internet) est configuré correctement avant d'exécuter les scénarios de test.
- Amélioration du vérificateur de dépendance Greengrass dans IDT pour le rendre plus flexible lors de la vérification de la libc sur les appareils.

IDT v2.2.0 pour AWS IoT Greengrass v1.10, v1.9.x et v1.8.x

Notes de mise à jour :

- Ajout de la prise en charge pour AWS IoT Greengrass version 1.10.
- Ajout de la prise en charge du connecteur de déploiement d'application Greengrass Docker.

- Ajout du support pour le gestionnaire de AWS IoT Greengrass flux.
- Ajout du support pour AWS IoT Greengrass la région de Chine (Pékin).

IDT v2.1.0 pour AWS IoT Greengrass v1.9.x, v1.8.x et v1.7.x

Notes de mise à jour :

- Ajout de la prise en charge pour AWS IoT Greengrass version 1.9.4.
- Ajout de la prise en charge des périphériques Linux-ARMv6l.

IDT v2.0.0 pour AWS IoT Greengrass v1.9.3, v1.9.2, v.1.9.1, v1.9.0, v1.8.4, v1.8.3 et v1.8.2

Notes de mise à jour :

- Suppression de la dépendance sur Python pour l'appareil testé.
- Le temps d'exécution de la suite de tests a été réduit de plus de 50 %, ce qui accélère le processus de qualification.
- La taille exécutable a été réduite de plus de 50 %, ce qui accélère le téléchargement et l'installation.
- Amélioration de la prise en charge du multiplicateur de délai d'attente pour tous les scénarios de test.
- Messages de post-diagnostic améliorés pour résoudre les erreurs plus rapidement.
- Mise à jour du modèle de stratégie d'autorisations requis pour exécuter IDT.
- Ajout de la prise en charge pour AWS IoT Greengrass version 3.9.1.

IDT v1.3.3 pour AWS IoT Greengrass v1.9.2, v1.9.1, v1.9.0, v1.8.3 et v1.8.2

Notes de mise à jour :

- Ajout de la prise en charge de Greengrass versions 1.9.2 et 1.8.3.
- Ajout du support pour Greengrass OpenWrt.
- Ajout de la connexion au périphérique du nom d'utilisateur et du mot de passe SSH.
- Ajout d'un correctif de bogue de test natif pour la plate-forme OpenWrt -ARMv7L.

IDT v1.2 pour AWS IoT Greengrass v1.8.1

Notes de mise à jour :

- Ajout d'un multiplicateur de délai d'expiration configurable pour traiter et résoudre des problèmes de dépassement de délai (par exemple, les connexions à bande passante faible).

IDT version 1.1 pour AWS IoT Greengrass version 1.8.0

Notes de mise à jour :

- Ajout de la prise en charge d'AWS IoT Greengrass Hardware Security Integration (HSI).
- Ajout de la prise en charge d'AWS IoT Greengrass avec conteneur et sans conteneur.
- Ajout de la création automatisée du rôle de service AWS IoT Greengrass.
- Amélioration du nettoyage de la ressource test.
- Ajout du rapport récapitulatif de l'exécution de test.

IDT version 1.1 pour AWS IoT Greengrass version 1.7.1

Notes de mise à jour :

- Ajout de la prise en charge d'AWS IoT Greengrass Hardware Security Integration (HSI).
- Ajout de la prise en charge d'AWS IoT Greengrass avec conteneur et sans conteneur.
- Ajout de la création automatisée du rôle de service AWS IoT Greengrass.
- Amélioration du nettoyage de la ressource test.
- Ajout du rapport récapitulatif de l'exécution de test.

IDT v1.0 pour AWS IoT Greengrass v1.6.1

Notes de mise à jour :

- Ajout d'un correctif de bogue de test OTA pour AWS IoT Greengrass la compatibilité des versions futures.

> **Note**
>
> Si vous utilisez IDT v1.0 pour AWS IoT Greengrass v1.6.1, vous devez créer un rôle de service Greengrass. Dans les versions ultérieures, IDT crée le rôle de service pour vous.

Utilisez IDT pour exécuter leAWS IoT Greengrasssuite de qualification

Vous pouvez utiliserAWS IoTDevice Tester (IDT) pourAWS IoT Greengrasspour vérifier que leAWS IoT GreengrassLe logiciel de base s'exécute sur votre matériel et peut communiquer avec le pluginAWS Cloud. Il exécute également end-to-end tests avecAWS IoT Core. Il vérifie par exemple que votre appareil peut envoyer et recevoir des messages MQTT et les traiter correctement.

Etant donné queAWS IoT Greengrass Version 1a été déplacé dans le documentmode maintenance, IDT pourAWS IoT Greengrass V1ne génère plus de rapports de qualification signés. Si vous souhaitez ajouter votre matériel au pluginAWS PartnerDevice Catalog, exécutez leAWS IoT Greengrass V2suite de qualifications pour générer des rapports de test que vous pouvez envoyer àAWS IoT. Pour de plus amples informations, veuillez consulterAWSProgramme de qualification des appareilsetVersions d'IDT prises en charge pourAWS IoT Greengrass V2.

Outre les appareils de test, IDT pourAWS IoT Greengrasscrée des ressources (par exemple,AWS IoTthings,AWS IoT Greengrassgroupes, fonctions Lambda, etc.) dans votreCompte AWSpour faciliter le processus de qualification.

Pour créer ces ressources, IDT forAWS IoT Greengrassutilise le pluginAWSinformations d'identification configurées dans le`config.json`fichier pour effectuer des appels d'API pour votre compte. Ces ressources sont allouées à différents moments d'un test.

Lorsque vous utilisez IDT pourAWS IoT Greengrasspour exécuter le pluginAWS IoT Greengrasssuite de qualification, IDT exécute les étapes suivantes :

1. Chargement et validation des informations d'identification et des informations d'identification de votre appareil.
2. Tests sélectionnés avec les ressources locales et les ressources cloud requises.
3. Élimination des ressources locales et des ressources cloud.
4. Génération de rapports de tests indiquant si votre appareil a réussi les tests obligatoires pour la qualification.

Versions de la suite de tests

IDT pour AWS IoT Greengrass organise les tests en suites de tests et groupes de tests.

- Une suite de tests est l'ensemble des groupes de tests utilisés pour vérifier qu'un appareil fonctionne avec des versions particulières de AWS IoT Greengrass.
- Un groupe de tests est l'ensemble de tests individuels liés à une fonctionnalité particulière, tels que les déploiements de groupe Greengrass et la messagerie MQTT.

À partir de IDT v3.0.0, les suites de tests sont versionnées à l'aide d'un format *major*.*minor*.*patch*, par exemple `GGQ_1.0.0`. Lorsque vous téléchargez IDT, le package inclut la version de suite de tests la plus récente.

> **Important**
>
> IDT prend en charge les trois dernières versions de la suite de tests pour la qualification des périphériques. Pour plus d'informations, consultez the section called “Politique de prise en charge d’AWS IoT Device Tester pour AWS IoT Greengrass V1”.
> Vous pouvez exécuter `list-supported-products` pour répertorier les versions de AWS IoT Greengrass et les suites de test prises en charge par votre version actuelle d'IDT. Les tests des versions de suite de tests non prises en charge ne sont pas valides pour la qualification des périphériques. IDT n'imprime pas les rapports de qualification pour les versions non prises en charge.

Mises à jour des paramètres de configuration IDT

De nouveaux tests peuvent introduire de nouveaux paramètres de configuration IDT.

- Si les paramètres sont facultatifs, IDT continue d'exécuter les tests.
- Si les paramètres sont requis, IDT vous en avertit et arrête l'exécution. Après avoir configuré les paramètres, redémarrez l'exécution des tests.

  Les paramètres de configuration se trouvent dans le dossier *<device-tester-extract-location>*`/configs`. Pour plus d'informations, consultez the section called “Configuration des paramètres IDT”.

Si une version de la suite de tests mise à jour ajoute des paramètres de configuration, IDT crée une copie du fichier de configuration d'origine dans *<device-tester-extract-location>*`/configs`.

Descriptions des groupes de tests

IDT v2.0.0 and later

Groupes de test requis pour la qualification du noyau

Ces groupes de tests sont nécessaires pour qualifier votreAWS IoT Greengrassappareil pour le pluginAWS PartnerCatalogue d'appareils.

Dépendances AWS IoT Greengrass Core

Valide que votre appareil répond à toutes les exigences logicielles et matérielles pour le logiciel AWS IoT Greengrass Core.

Le cas test `Software Packages Dependencies` de ce groupe de test n'est pas applicable lors d'un test dans un conteneur Docker.

Déploiement

Ce groupe de tests valide le déploiement des fonctions Lambda sur votre appareil.

MQTT

Vérifie leAWS IoT Greengrassfonctionnalité de routeur de messages en vérifiant la communication locale entre le noyau Greengrass et les appareils clients, qui sont des appareils IoT locaux.

Over-the-Air (OTA)

Valide que votre appareil peut effectuer une mise à jour OTA du logiciel AWS IoT Greengrass Core.

Ce groupe de test n'est pas applicable lors d'un test dans un conteneur Docker.

Version

Vérifie que la version de AWS IoT Greengrass fournie est compatible avec la version AWS IoT Device Tester que vous utilisez.

Groupes de test facultatifs

Ces groupes de test sont facultatifs. Si vous choisissez d'effectuer une qualification pour des tests facultatifs, votre appareil est répertorié avec des fonctionnalités supplémentaires dans le documentAWS PartnerCatalogue d'appareils.

Dépendances de conteneur

Ce groupe de tests valide toutes les exigences logicielles et matérielles requises pour exécuter des fonctions Lambda en mode conteneur sur un noyau Greengrass.

Ce groupe de test n'est pas applicable lors d'un test dans un conteneur Docker.

Conteneur de déploiement

Ce groupe de tests valide le déploiement des fonctions Lambda sur l'appareil et exécutées en mode conteneur sur un noyau Greengrass.

Ce groupe de test n'est pas applicable lors d'un test dans un conteneur Docker.

Dépendances Docker (prises en charge pour IDT v2.2.0 et versions ultérieures)

Valide que l'appareil est conforme à toutes les dépendances techniques requises pour pouvoir utiliser le connecteur de déploiement d'application Greengrass Docker pour exécuter des conteneurs

Ce groupe de test n'est pas applicable lors d'un test dans un conteneur Docker.

Intégration de sécurité matérielle (HSI)

Vérifie que la bibliothèque partagée HSI fournie peut interagir avec le module de sécurité matérielle (HSM) et qu'elle implémente correctement les API PKCS # 11 requises. La bibliothèque partagée/HSM doit signer une CSR, effectuer des opérations TLS et fournir les longueurs de clé et l'algorithme de clé publique corrects.

Dépendances du gestionnaire de flux (prises en charge pour IDT v2.2.0 et versions ultérieures)

Valide que l'appareil est conforme à toutes les dépendances techniques requises pour pouvoir exécuter le gestionnaire de flux AWS IoT Greengrass.

Dépendances de Machine Learning (prises en charge pour IDT v3.1.0 et versions ultérieures)

Valide que l'appareil est conforme à toutes les dépendances techniques requises pour pouvoir exécuter une inférence ML localement.

Tests d'inférence de Machine Learning (pris en charge pour IDT v3.1.0 et versions ultérieures)

Valide que l'inférence ML peut être effectuée sur l'appareil testé. Pour plus d'informations, consultez the section called “Facultatif : Configuration de votre appareil pour la qualification ML”.

Tests de conteneur d'inférence de Machine Learning (pris en charge pour IDT v3.1.0 et versions ultérieures)

Valide que l'inférence ML peut être effectuée sur l'appareil testé et exécutée en mode conteneur sur un noyau Greengrass. Pour plus d'informations, consultez the section called “Facultatif : Configuration de votre appareil pour la qualification ML”.

IDT v1.3.3 and earlier

Groupes de test requis pour la qualification du noyau

Ces tests sont nécessaires pour qualifier votreAWS IoT Greengrassappareil pour le pluginAWS PartnerCatalogue d'appareils.

Dépendances AWS IoT Greengrass Core

Valide que votre appareil répond à toutes les exigences logicielles et matérielles pour le logiciel AWS IoT Greengrass Core.

Combinaison (interaction de sécurité de l'appareil)

Vérifie la fonctionnalité du détecteur IP et du gestionnaire de certificat d'appareil du noyau Greengrass en modifiant les informations de connectivité au niveau du groupe Greengrass dans le cloud. Le groupe de tests fait tourner le certificat du serveur AWS IoT Greengrass et vérifie qu'AWS IoT Greengrass autorise les connexions.

Déploiement (requis pour IDT v1.2 et versions antérieures)

Ce groupe de tests valide le déploiement des fonctions Lambda sur votre appareil.

Gestionnaire de certificats de l'appareil

Vérifie que le gestionnaire de certificats des appareils AWS IoT Greengrass peut générer un certificat de serveur au démarrage et effectuer une rotation des certificats s'ils sont sur le point d'expirer.

Détection IP (IDP)

Vérifie que les informations sur la connectivité du noyau sont mises à jour en cas de modifications d'adresses IP sur un appareil Greengrass Core. Pour plus d'informations, consultez Activation de la la détection IP automatique.

Logging

Vérifie que leAWS IoT GreengrassLe service de journalisation peut écrire dans un fichier journal à l'aide de la fonction Lambda écrite en Python.

MQTT

Vérifie la fonctionnalité du routeur de messages AWS IoT Greengrass en envoyant des messages dans une rubrique qui est acheminée vers deux fonctions Lambda.

Natif

Vérifie que AWS IoT Greengrass peut exécuter des fonctions Lambda natives (compilées).

Over-the-Air (OTA)

Valide que votre appareil peut effectuer une mise à jour OTA du logiciel AWS IoT Greengrass Core.

Pénétration

Valide que le logiciel AWS IoT Greengrass Core ne parvient pas à démarrer si la protection hardlink/softlink et seccomp ne sont pas activés. Il vérifie également d'autres fonctions liées à la sécurité.

Shadow

Vérifie la fonctionnalité shadow locale et la fonctionnalité de synchronisation du Cloud shadow.

Spooler

Vérifie que les messages MQTT sont mis en file d'attente avec la configuration du spooler par défaut.

Service d'échange de jeton (TES)

Vérifie que.AWS IoT Greengrasspeut échanger son certificat de base contre un certificat valideAWSInformations d'identification .

Version

Vérifie que la version de AWS IoT Greengrass fournie est compatible avec la version AWS IoT Device Tester que vous utilisez.

Groupes de test facultatifs

Ces tests sont facultatifs. Si vous choisissez d'effectuer une qualification pour des tests facultatifs, votre appareil est répertorié avec des fonctionnalités supplémentaires dans le documentAWS PartnerCatalogue d'appareils.

Dépendances de conteneur

Vérifie que l'appareil répond à toutes les dépendances requises pour exécuter des fonctions Lambda en mode conteneur.

Intégration de sécurité matérielle (HSI)

Vérifie que la bibliothèque partagée HSI fournie peut interagir avec le module de sécurité matérielle (HSM) et qu'elle implémente correctement les API PKCS # 11 requises. La bibliothèque partagée/HSM doit signer une CSR, effectuer des opérations TLS et fournir les longueurs de clé et l'algorithme de clé publique corrects.

Accès aux ressources locales

Vérifie la fonctionnalité d'accès aux ressources locales (LRA) deAWS IoT Greengrassen fournissant l'accès aux fichiers et groupes locaux appartenant à différents utilisateurs et groupes Linux aux fichiers et groupes Linux à des fichiers et groupes Linux à des fonctions Lambda via leAWS IoT GreengrassAPI LRA. L'accès des fonctions Lambda aux ressources locales est autorisé ou refusé en fonction de la configuration d'accès aux ressources locales.

Réseau

Vérifie que les connexions socket peuvent être établies à partir d'une fonction Lambda. Ces connexions socket doivent être autorisées ou refusées en fonction de la configuration du noyau Greengrass.

Conditions préalables à l'exécution de la suite de AWS IoT Greengrass qualifications

Cette section décrit les conditions préalables à l'utilisation de AWS IoT Device Tester (IDT) AWS IoT Greengrass pour exécuter la suite de AWS IoT Greengrass qualification.

Téléchargez la dernière version de AWS IoT Device Tester pour AWS IoT Greengrass

Téléchargez la dernière version d'IDT et extrayez le logiciel dans un emplacement de votre système de fichiers où vous disposez d'autorisations de lecture et d'écriture.

> **Note**
>
> IDT ne prend pas en charge son exécution par plusieurs utilisateurs à partir d'un emplacement partagé, tel qu'un répertoire NFS ou un dossier partagé réseau Windows. Nous vous recommandons d'extraire le package IDT sur une unité locale et d'exécuter le fichier binaire IDT sur votre station de travail locale.
> Pour Windows, la limitation de la longueur du chemin est de 260 caractères. Si vous utilisez Windows, décompressez IDT dans un répertoire racine comme `C:\` ou `D:\` afin que la longueur de vos chemins respecte la limite de 260 caractères.

Créez et configurez un Compte AWS

Avant de pouvoir utiliser IDT pour AWS IoT Greengrass, vous devez effectuer les étapes suivantes :

1. Créez un Compte AWS. Si vous en avez déjà un Compte AWS, passez à l'étape 2.
2. Configurez les autorisations pour IDT.

Ces autorisations de compte permettent à IDT d'accéder aux AWS services et de créer des AWS ressources, telles que des AWS IoT objets, des groupes Greengrass et des fonctions Lambda, en votre nom.

Pour créer ces ressources, IDT for AWS IoT Greengrass utilise les AWS informations d'identification configurées dans le `config.json` fichier pour effectuer des appels d'API en votre nom. Ces ressources sont allouées à différents moments d'un test.

> **Note**
>
> Bien que la plupart des tests soient éligibles au niveau gratuit d'Amazon Web Services, vous devez fournir une carte de crédit lorsque vous vous inscrivez à un Compte AWS. Pour de plus amples informations, veuillez consulter Pourquoi ai-je besoin d'un mode de paiement si mon compte est couvert par le niveau gratuit ?.

Étape 1 : Création d'un Compte AWS

Au cours de cette étape, créez et configurez un Compte AWS. Si vous en avez déjà un Compte AWS, passez directement àthe section called “Étape 2 : Configurer les autorisations pour IDT”.

Inscrivez-vous pour un Compte AWS

Si vous n'en avez pas Compte AWS, procédez comme suit pour en créer un.

Pour vous inscrire à un Compte AWS

1. Ouvrez https://portal.aws.amazon.com/billing/signup.
2. Suivez les instructions en ligne.

   Dans le cadre de la procédure d‘inscription, vous recevrez un appel téléphonique et vous saisirez un code de vérification en utilisant le clavier numérique du téléphone.

   Lorsque vous vous inscrivez à un Compte AWS, un Utilisateur racine d'un compte AWSest créé. Par défaut, seul l‘utilisateur racine a accès à l‘ensemble des Services AWS et des ressources de ce compte. La meilleure pratique en matière de sécurité consiste à attribuer un accès administratif à un utilisateur et à n'utiliser que l'utilisateur root pour effectuer les tâches nécessitant un accès utilisateur root.

AWS vous envoie un e-mail de confirmation une fois le processus d'inscription terminé. Vous pouvez afficher l‘activité en cours de votre compte et gérer votre compte à tout moment en accédant à https://aws.amazon.com/ et en choisissant Mon compte.

Création d'un utilisateur doté d'un accès administratif

Une fois que vous vous êtes inscrit à un utilisateur administratif Compte AWS, que vous Utilisateur racine d'un compte AWS l'avez sécurisé AWS IAM Identity Center, que vous l'avez activé et que vous en avez créé un, afin de ne pas utiliser l'utilisateur root pour les tâches quotidiennes.

Sécurisez votre Utilisateur racine d'un compte AWS

1. Connectez-vous en AWS Management Consoletant que propriétaire du compte en choisissant Utilisateur root et en saisissant votre adresse Compte AWS e-mail. Sur la page suivante, saisissez votre mot de passe.

Pour obtenir de l‘aide pour vous connecter en utilisant l‘utilisateur racine, consultez Connexion en tant qu‘utilisateur racine dans le Guide de l‘utilisateur Connexion à AWS .

2. Activez l‘authentification multifactorielle (MFA) pour votre utilisateur racine.

   Pour obtenir des instructions, voir Activer un périphérique MFA virtuel pour votre utilisateur Compte AWS root (console) dans le guide de l'utilisateur IAM.

Création d'un utilisateur doté d'un accès administratif

1. Activez IAM Identity Center.

   Pour obtenir des instructions, consultez Activation d’ AWS IAM Identity Center dans le Guide de l’utilisateur AWS IAM Identity Center .

2. Dans IAM Identity Center, accordez un accès administratif à un utilisateur.

   Pour un didacticiel sur l'utilisation du Répertoire IAM Identity Center comme source d'identité, voir Configurer l'accès utilisateur par défaut Répertoire IAM Identity Center dans le Guide de AWS IAM Identity Center l'utilisateur.

Connectez-vous en tant qu'utilisateur disposant d'un accès administratif

- Pour vous connecter avec votre utilisateur IAM Identity Center, utilisez l‘URL de connexion qui a été envoyée à votre adresse e-mail lorsque vous avez créé l‘utilisateur IAM Identity Center.

  Pour obtenir de l'aide pour vous connecter en utilisant un utilisateur d'IAM Identity Center, consultez la section Connexion au portail AWS d'accès dans le guide de l'Connexion à AWS utilisateur.

Attribuer l'accès à des utilisateurs supplémentaires

1. Dans IAM Identity Center, créez un ensemble d'autorisations conforme aux meilleures pratiques en matière d'application des autorisations du moindre privilège.

   Pour obtenir des instructions, voir Création d'un ensemble d'autorisations dans le guide de AWS IAM Identity Center l'utilisateur.

2. Affectez des utilisateurs à un groupe, puis attribuez un accès d'authentification unique au groupe.

Pour obtenir des instructions, consultez la section Ajouter des groupes dans le guide de AWS IAM Identity Center l'utilisateur.

Étape 2 : Configurer les autorisations pour IDT

Dans cette étape, configurez les autorisations qu'IDT for AWS IoT Greengrass utilise pour exécuter des tests et collecter des données d'utilisation IDT. Vous pouvez utiliser le AWS Management Console ou AWS Command Line Interface (AWS CLI) pour créer une stratégie IAM et un utilisateur de test pour IDT, puis associer des politiques à l'utilisateur. Si vous avez déjà créé un utilisateur test pour IDT, passez à the section called “Configurer votre appareil afin d'exécuter des tests IDT” ou the section called “Facultatif : Configuration de votre conteneur Docker”.

- Pour configurer des autorisations pour IDT (console)
- Pour configurer des autorisations pour IDT (AWS CLI)

Pour configurer des autorisations pour IDT (console)

Procédez comme suit pour utiliser la console afin de configurer les autorisations pour IDT pour AWS IoT Greengrass.

1. Connectez-vous à la console IAM.
2. Créez une stratégie gérée par le client qui accorde des autorisations de création des rôles avec des autorisations spécifiques.

   a. Dans le volet de navigation, sélectionnez Politiques, puis Créer une politique.

   b. Sur l'onglet JSON, remplacez le contenu de l'espace réservé par la stratégie suivante.

```
{
    "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

La stratégie suivante accorde l'autorisation de créer et de gérer les rôles requis par IDT pour AWS IoT Greengrass. Cela inclut les autorisations permettant d'associer les politiques AWS gérées suivantes :

- AWSGreengrassResourceAccessRolePolicy
- Grassota verte UpdateArtifactAccess

- AWSLambdaBasicExecutionRole

c. Choisissez Suivant : Balises.

d. Choisissez Suivant : Vérification.

e. Pour Name (Nom), saisissez **`IDTGreengrassIAMPermissions`**. Sous Résumé, vérifiez les autorisations accordées par votre stratégie.

f. Choisissez Créer une politique.

3. Créez un utilisateur IAM et associez les autorisations requises par IDT pour. AWS IoT Greengrass

a. Créez un utilisateur IAM. Suivez les étapes 1 à 5 de la section Création d'utilisateurs IAM (console) dans le guide de l'utilisateur IAM.

b. Associez les autorisations à votre utilisateur IAM :

i. Sur la page Définir les autorisations, choisissez Joindre directement les politiques existantes.

ii. Recherchez la stratégie IDTGreengrassIAMPermissions que vous avez créée à l'étape précédente. Activez la case à cocher.

iii. Recherchez la AWSIoTDeviceTesterForGreengrassFullAccesspolitique. Activez la case à cocher.

> **Note**
>
> AWSIoTDeviceTesterForGreengrassFullAccessIl s'agit d'une politique AWS gérée qui définit les autorisations dont IDT a besoin pour créer et accéder aux AWS ressources utilisées pour les tests. Pour plus d'informations, consultez the section called “AWS politique gérée pour IDT”.

c. Choisissez Suivant : Identifications.

d. Choisissez Suivant : Réviser pour afficher un résumé de vos choix.

e. Choisissez Create user (Créer un utilisateur).

f. Pour afficher les clés d'accès de l'utilisateur (ID de clé d'accès et clés d'accès secrètes), choisissez Afficher en regard du mot de passe et de la clé d'accès. Pour enregistrer les clés d'accès, choisissez Télécharger .csv, puis enregistrez le fichier dans un emplacement

sécurisé sur votre ordinateur. Vous utiliserez ces informations ultérieurement pour configurer votre fichier AWS d'informations d'identification.

4. Étape suivante : Configurez votre appareil physique.

Pour configurer des autorisations pour IDT (AWS CLI)

Procédez comme suit pour utiliser le AWS CLI afin de configurer les autorisations pour IDT pour AWS IoT Greengrass. Si vous avez déjà configuré des autorisations dans la console, passez à the section called “Configurer votre appareil afin d'exécuter des tests IDT” ou the section called “Facultatif : Configuration de votre conteneur Docker”.

1. Sur votre ordinateur, installez et configurez le AWS CLI s'il n'est pas déjà installé. Suivez les étapes décrites AWS CLI dans la section Installation du guide de AWS Command Line Interface l'utilisateur.

   > **Note**
   >
   > AWS CLI Il s'agit d'un outil open source que vous pouvez utiliser pour interagir avec les AWS services à partir de votre shell de ligne de commande.

2. Créez une stratégie gérée par le client qui accorde les autorisations pour gérer IDT et les rôles 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**
>
> Cette étape inclut un exemple d'invite de commande Windows car elle utilise une syntaxe JSON différente de celle des commandes de terminal Linux, macOS ou Unix.

3. Créez un utilisateur IAM et associez les autorisations requises par IDT pour. AWS IoT Greengrass

   a. Créez un utilisateur IAM. Dans cet exemple de configuration, l'utilisateur est nommé `IDTGreengrassUser`.

   ```
   aws iam create-user --user-name IDTGreengrassUser
   ```

   b. Attachez la `IDTGreengrassIAMPermissions` politique que vous avez créée à l'étape 2 à votre utilisateur IAM. Remplacez <account-id>dans la commande par l'identifiant de votre Compte AWS.

   ```
   aws iam attach-user-policy --user-name IDTGreengrassUser --policy-arn
    arn:aws:iam::<account-id>:policy/IDTGreengrassIAMPermissions
   ```

   c. Attachez la `AWSIoTDeviceTesterForGreengrassFullAccess` politique à votre utilisateur IAM.

   ```
   aws iam attach-user-policy --user-name IDTGreengrassUser --policy-arn
    arn:aws:iam::aws:policy/AWSIoTDeviceTesterForGreengrassFullAccess
   ```

   > **Note**
   >
   > AWSIoTDeviceTesterForGreengrassFullAccessIl s'agit d'une politique AWS gérée qui définit les autorisations dont IDT a besoin pour créer et accéder aux AWS ressources utilisées pour les tests. Pour plus d'informations, consultez the section called “AWS politique gérée pour IDT”.

4. Créez une clé d'accès secrète pour l'utilisateur.

```
aws iam create-access-key --user-name IDTGreengrassUser
```

Stockez la sortie dans un emplacement sécurisé. Vous utiliserez ces informations ultérieurement pour configurer votre fichier AWS d'informations d'identification.

5. Étape suivante : Configurez votre appareil physique.

AWS politique gérée pour AWS IoT Device Tester

La politique AWSIoTDeviceTesterForGreengrassFullAccessgérée permet à IDT d'exécuter des opérations et de collecter des métriques d'utilisation. Cette stratégie accorde les autorisations IDT suivantes :

- `iot-device-tester:CheckVersion`. Vérifiez si un ensemble de versions AWS IoT Greengrass, une suite de tests et une version IDT sont compatibles.
- `iot-device-tester:DownloadTestSuite`. Téléchargez les suites de tests.
- `iot-device-tester:LatestIdt`. Obtenez des informations sur la dernière version d'IDT disponible en téléchargement.
- `iot-device-tester:SendMetrics`. Publiez les données d'utilisation collectées par IDT à propos de vos tests.
- `iot-device-tester:SupportedVersion`. Obtenez la liste des versions AWS IoT Greengrass de la suite de tests prises en charge par IDT. Ces informations apparaissent dans la fenêtre de ligne de commande.

Configurer votre appareil afin d'exécuter des tests IDT

Pour configurer votre appareil, vous devez installer les dépendances AWS IoT Greengrass, configurer le logiciel AWS IoT Greengrass Core, configurer votre ordinateur hôte pour accéder à votre appareil et configurer des autorisations utilisateur sur votre appareil.

Vérifier les dépendances AWS IoT Greengrass sur l'appareil testé

Avant qu'IDT pour AWS IoT Greengrass ne puisse tester vos appareils, assurez-vous que vous avez configuré votre appareil comme décrit dans Mise en route sur AWS IoT Greengrass. Pour de plus amples informations sur les plateformes prises en charge, veuillez consulter Plateformes prises en charge.

Configurer le logiciel AWS IoT Greengrass

IDT pour AWS IoT Greengrass teste la compatibilité de votre appareil avec une version spécifique d'AWS IoT Greengrass. IDT vous offre deux options pour tester AWS IoT Greengrass sur vos appareils :

- Téléchargez et utilisez une version du logiciel AWS IoT Greengrass Core. IDT installe le logiciel pour vous.
- Utilisez une version du logiciel AWS IoT Greengrass Core déjà installée sur votre appareil.

> **Note**
>
> Chaque version d'AWS IoT Greengrass possède une version IDT correspondante. Vous devez télécharger la version d'IDT qui correspond à la version d'AWS IoT Greengrass que vous utilisez.

Les sections suivantes décrivent ces options. Vous devez en choisir une seule.

Option 1 : Télécharger leAWS IoT GreengrassLogiciel Core et configurationAWS IoTTesteur de périphérique pour l'utiliser

Vous pouvez télécharger leAWS IoT GreengrassLogiciel Core du logicielAWS IoT GreengrassLogiciel Corepage des téléchargements.

1. Identifiez l'architecture et la distribution Linux appropriées, puis choisissez Download (Télécharger).
2. Copiez le fichier tar.gz dans *<device-tester-extract-location>*`/products/greengrass/ggc`.

> **Note**
>
> Ne modifiez pas le nom du fichier AWS IoT Greengrass tar.gz. Ne placez pas plusieurs fichiers dans ce répertoire pour le même système d'exploitation et la même architecture. Par exemple, si les fichiers `greengrass-linux-armv7l-1.7.1.tar.gz` et `greengrass-linux-armv7l-1.8.1.tar.gz` figurent tous les deux dans ce répertoire, le test échoue.

Option 2 : Utiliser une installation existante deAWS IoT GreengrassavecAWS IoTDevice Tester

Configurez IDT pour tester le logiciel AWS IoT Greengrass Core sur votre appareil en ajoutant l'attribut `greengrassLocation` dans le fichier `device.json` situé dans le dossier *<device-tester-extract-location>*`/configs`. Par exemple :

```
"greengrassLocation" : "<path-to-greengrass-on-device>"
```

Pour plus d'informations sur le fichier `device.json`, consultez Configurer device.json.

Sur les appareils Linux, l'emplacement par défaut du logiciel AWS IoT Greengrass Core est `/greengrass`.

> **Note**
>
> Votre appareil doit disposer d'une installation du logiciel AWS IoT Greengrass Core n'ayant pas été démarrée.
> Vérifiez que vous avez ajouté l'utilisateur `ggc_user` et `ggc_group` sur votre appareil. Pour de plus amples informations, veuillez consulter Configuration de l'environnement pour AWS IoT Greengrass.

Configurer votre ordinateur hôte pour accéder à l'appareil testé

IDT s'exécute sur votre ordinateur hôte et doit être en mesure d'utiliser SSH pour se connecter à votre appareil. Il existe deux options pour permettre à IDT d'obtenir un accès SSH à vos appareils testés :

1. Suivez les instructions indiquées ici pour créer une paire de clés SSH et autoriser votre clé à se connecter à votre appareil testé sans spécifier de mot de passe.
2. Fournissez un nom d'utilisateur et un mot de passe pour chaque appareil du fichier `device.json`. Pour plus d'informations, consultez Configurer device.json.

Vous pouvez utiliser n'importe quelle implémentation SSL pour créer une clé SSH. Les instructions suivantes vous montrent comment utiliser SSH-KEYGEN ou PuTTYgen (pour Windows). Si vous utilisez une autre implémentation SSL, veuillez vous reporter à la documentation correspondante.

IDT utilise les clés SSH pour s'authentifier avec votre appareil testé.

Pour créer une clé SSH avec SSH-KEYGEN

1. Créez une clé SSH.

   Vous pouvez utiliser la commande ssh-keygen Open SSH pour créer une paire de clés SSH. Si vous disposez déjà d'une paire de clés SSH sur votre ordinateur hôte, la bonne pratique consiste à créer une paire de clés SSH spécifique pour IDT. Ainsi, une fois que vous avez terminé le test, votre ordinateur hôte ne peut plus se connecter à votre appareil sans saisir de mot de passe. Cela vous permet également de restreindre l'accès à l'appareil à distance aux seules personnes qui en ont besoin.

   > **Note**
   >
   > Windows n'a pas de client SSH installé. Pour plus d'informations sur l'installation d'un client SSH sous Windows, consultez Download SSH Client Software.

   La commande ssh-keygen vous invite à indiquer un nom et un chemin d'accès pour stocker la paire de clés. Par défaut, les fichiers de la paire de clés sont nommés `id_rsa` (clé privée) et `id_rsa.pub` (clé publique). Sur Mac OS et Linux, l'emplacement par défaut de ces fichiers est `~/.ssh/`. Sur Windows, l'emplacement par défaut est `C:\Users\`*`<user-name>`*`\.ssh`.

   Lorsque vous y êtes invité, saisissez une expression clé pour protéger votre clé SSH. Pour de plus amples informations, veuillez consulter Generate a New SSH key.

2. Ajoutez des clés SSH autorisées à l'appareil testé.

   IDT doit utiliser votre clé SSH privée pour se connecter à l'appareil testé. Pour autoriser votre clé SSH privée à se connecter à l'appareil testé, utilisez la commande ssh-copy-id à partir de votre ordinateur hôte. Cette commande ajoute votre clé publique au fichier `~/.ssh/authorized_keys` sur l'appareil testé. Par exemple :

   $ ssh-copy-id ***<remote-ssh-user>***@***<remote-device-ip>***

   Où *`remote-ssh-user`* correspond au nom d'utilisateur utilisé pour se connecter à l'appareil à tester et où *`remote-device-ip`* est l'adresse IP de l'appareil à tester. Par exemple :

   ssh-copy-id pi@192.168.1.5

Lorsque vous y êtes invité, entrez le mot de passe du nom d'utilisateur que vous avez spécifié dans la commande ssh-copy-id.

ssh-copy-id suppose que la clé publique est nommée `id_rsa.pub` et est stockée à l'emplacement par défaut (sur Mac OS et Linux, `~/.ssh/` et sur Windows, `C:\Users\<user-name>\.ssh`). Si vous avez donné à la clé publique un autre nom ou si vous l'avez stockée à un autre emplacement, vous devez spécifier le chemin d'accès qualifié complet de votre clé publique SSH à l'aide de l'option -i pour ssh-copy-id (par exemple, ssh-copy-id -i ~/my/path/myKey.pub). Pour plus d'informations sur la création de clés SSH et la copie des clés publiques, consultez SSH-COPY-ID.

Pour créer une clé SSH à l'aide de PuTTYgen (Windows uniquement)

1. Assurez-vous que le serveur et le client OpenSSH sont installés sur votre appareil testé. Pour plus d'informations, consultez OpenSSH.
2. Installez PuTTYgen sur votre appareil testé.
3. Ouvrez PuTTYgen.
4. Choisissez Generate (Générer) et déplacez le curseur de la souris dans la zone pour générer une clé privée.
5. Dans le menu Conversions choisissez Export OpenSSH key, et enregistrez la clé privée avec une extension de fichier `.pem`.
6. Ajoutez la clé publique au fichier `/home/<user>/.ssh/authorized_keys` sur l'appareil testé.

   a. Copiez le texte de la clé publique à partir de la fenêtre PuTTYgen.

   b. Utilisez PuTTY pour créer une session sur votre appareil testé.

      i. À partir d'une invite de commande ou d'une fenêtre Windows Powershell, exécutez la commande suivante :

      C:/***<path-to-putty>***/putty.exe -ssh ***<user>***@***<dut-ip-address>***

      ii. Lorsque vous y êtes invité, entrez le mot de passe de votre appareil.

      iii. Utilisez vi ou un autre éditeur de texte pour ajouter la clé publique au fichier `/home/<user>/.ssh/authorized_keys` sur votre appareil testé.

7. Mettez à jour votre fichier `device.json` avec votre nom d'utilisateur, l'adresse IP et le chemin d'accès au fichier de clé privée que vous venez d'enregistrer sur votre ordinateur hôte pour chaque appareil testé. Pour plus d'informations, consultez the section called “Configurer device.json”. Assurez-vous de fournir le chemin d'accès complet et le nom de fichier à la clé privée et d'utiliser des barres obliques (« / »). Par exemple, pour le chemin Windows `C:\DT\privatekey.pem`, utilisez `C:/DT/privatekey.pem` dans le fichier `device.json`.

Configurer les autorisations utilisateur sur votre appareil

IDT effectue des opérations sur différents répertoires et fichiers d'un appareil testé. Certaines de ces opérations nécessitent des autorisations d'un niveau élevé (à l'aide de la commande sudo). Pour automatiser ces opérations, IDT pour AWS IoT Greengrass doit être en mesure d'exécuter des commandes avec la commande sudo sans être invité à saisir un mot de passe.

Suivez ces étapes sur l'appareil testé afin d'autoriser l'accès sudo sans avoir à saisir un mot de passe.

> **Note**
>
> `username` fait référence à l'utilisateur SSH utilisé par IDT pour accéder à l'appareil testé.

Pour ajouter l'utilisateur au groupe sudo

1. Sur l'appareil testé, exécutez `sudo usermod -aG sudo <username>`
2. Déconnectez-vous, puis reconnectez-vous pour que les modifications entrent en vigueur.
3. Vérifiez que votre nom d'utilisateur a été correctement ajouté et exécutez sudo echo test. Si vous n'êtes pas invité à saisir un mot de passe, cela signifie que votre utilisateur est configuré correctement.
4. Ouvrez le fichier `/etc/sudoers`, puis ajoutez la ligne suivante à la fin du fichier :

   ```
   <ssh-username> ALL=(ALL) NOPASSWD: ALL
   ```

Configurer votre appareil pour tester des fonctions facultatives

Les rubriques suivantes décrivent comment configurer vos appareils pour exécuter des tests IDT pour des fonctions facultatives. Suivez ces étapes de configuration uniquement si vous souhaitez tester ces fonctions. Dans le cas contraire, passez à the section called “Configuration des paramètres IDT”.

Rubriques

Facultatif : Configuration de votre conteneur Docker pour IDT pourAWS IoT Greengrass

AWS IoT Greengrass fournit une image Docker et un Dockerfile qui facilitent l'exécution du logiciel AWS IoT Greengrass Core dans un conteneur Docker. Après avoir configuré le conteneur AWS IoT Greengrass, vous pouvez exécuter des tests IDT. Actuellement, seules les architectures Docker x86_64 sont prises en charge pour exécuter IDT pour AWS IoT Greengrass.

Cette fonctionnalité nécessite IDT v2.3.0 ou une version ultérieure.

Le processus de configuration du conteneur Docker pour exécuter des tests IDT dépend de l'utilisation de l'image Docker ou du Dockerfile fourni par AWS IoT Greengrass.

- Utilisez l'image Docker. L'image Docker a le logiciel AWS IoT Greengrass Core et ses dépendances installés.
- Utilisez le Dockerfile. Le Dockerfile contient le code source que vous pouvez utiliser pour créer des images de conteneur AWS IoT Greengrass personnalisées. L'image peut être modifiée pour s'exécuter sur différentes architectures de plateformes ou pour réduire la taille de l'image.

> **Note**
>
> AWS IoT Greengrassne fournit pas d'images Dockerfiles ou Docker pourAWS IoT GreengrassVersion 1.11.1 du logiciel Core. Pour exécuter des tests IDT sur vos propres images de conteneur personnalisées, votre image doit inclure les dépendances définies dans le Dockerfile fourni par AWS IoT Greengrass.

Les fonctions suivantes ne sont pas disponibles lorsque vous exécutez AWS IoT Greengrass dans un conteneur Docker :

- Connecteurs qui s'exécutent en mode Greengrass container (Conteneur Greengrass). Pour exécuter un connecteur dans un conteneur Docker, le connecteur doit s'exécuter en mode No container (Aucun conteneur). Pour rechercher des connecteurs prenant en charge le mode No container (Aucun conteneur) veuillez consulter the section called “AWS- connecteurs Greengrass fournis”. Certains de ces connecteurs ont un paramètre de mode d'isolement que vous devez définir sur Aucun conteneur.
- Ressources de volumes et d'appareils locales. Vos fonctions Lambda définies par l'utilisateur qui s'exécutent dans le conteneur Docker doivent accéder directement aux appareils et aux volumes du noyau.

Configurer l'image Docker fournie par AWS IoT Greengrass

Procédez comme suit pour configurer l'image AWS IoT Greengrass Docker afin d'exécuter des tests IDT.

Conditions préalables

Avant de commencer ce didacticiel, vous devez effectuer les opérations suivantes.

- Vous devez installer les logiciels et les versions suivants sur votre ordinateur hôte en fonction de l'AWS Command Line Interface(AWS CLI) version que vous choisissez.

  AWS CLI version 2

  - Dockerversion 18.09 ou suivante. Les versions antérieures sont susceptibles d'également fonctionner, mais nous recommandons la version 18.09 ou suivante.
  - AWS CLIversion 2.0.0 ou suivante.
    - Pour installerAWS CLIversion 2, voirInstallation deAWS CLIversion 2.
    - Pour configurer l'AWS CLI, voirConfiguration deAWS CLI.

    > **Note**
    >
    > Pour passer à une version ultérieureAWS CLIExécuter la version 2 d'un ordinateur Windows, vous devez répéter l'Installation MSIprocessus.

AWS CLI version 1

- Dockerversion 18.09 ou suivante. Les versions antérieures sont susceptibles d'également fonctionner, mais nous recommandons la version 18.09 ou suivante.
- Pythonversion 3.6 ou suivante.
- pip version 18.1 ou suivante.
- AWS CLIversion 1.17.10 ou suivante
  - Pour installerAWS CLIversion 1, voirInstallation deAWS CLIversion 1.
  - Pour configurer l'AWS CLI, voirConfiguration deAWS CLI.
  - Pour effectuer une mise à niveau vers la dernière version du kitAWS CLIExécutez la commande ci-après.

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

    > **Note**
    >
    > Si vous utilisez le pluginInstallation MSIduAWS CLIPour Windows, vous devez être conscient des points suivants :
    >
    > - Si l'icôneAWS CLIL'installation de la version 1 ne parvient pas à pour installer botocore, essayez d'utiliser le kitInstallation Python et pip.
    > - Pour passer à une version ultérieureAWS CLIVersion 1, vous devez répéter le processus d'installation MSI.

- Pour accéder aux ressources Amazon Elastic Container Registry (Amazon ECR), vous devez accorder l'autorisation suivante.
  - Amazon ECR exige que les utilisateurs accordent le`ecr:GetAuthorizationToken`autorisation par le biais d'unAWS Identity and Access Management(IAM) avant qu'ils puissent s'authentifier auprès d'un référentiel et transmettre ou extraire des images à partir d'un référentiel Amazon ECR. Pour de plus amples informations, veuillez consulterExemples de politique de référentiel Amazon ECRetAccéder à un seul référentiel Amazon ECRdans leGuide de l'utilisateur Amazon Elastic Container.

1. Téléchargez l'image Docker et configurez le conteneur. Vous pouvez télécharger l'image prédéfinie à partir deDocker HubouAmazon Elastic Container Registry(Amazon ECR) et l'exécuter sur les plateformes Windows, macOS et Linux (x86_64).

   Pour télécharger l'image Docker à partir d'Amazon ECR, suivez toutes les étapes de la sectionthe section called “Obtenir l'image deAWS IoT Greengrass conteneur auprès d'Amazon ECR”. Revenez ensuite à cette rubrique pour continuer la configuration.

2. Utilisateurs Linux uniquement : Assurez-vous que l'utilisateur qui exécute IDT a l'autorisation d'exécuter les commandes Docker. Pour de plus amples informations, veuillez consulter Gérer Docker en tant qu'utilisateur non-racine dans la documentation Docker.

3. Pour exécuter le conteneur AWS IoT Greengrass, utilisez la commande de votre système d'exploitation :

   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>
   ```

   - Remplacez *<host-path-to-kernel-config-file>* par le chemin d'accès au fichier de configuration du noyau sur l'hôte et *<container-path>* par le chemin d'accès où le volume est monté dans le conteneur.

     Le fichier de configuration du noyau sur l'hôte se trouve généralement dans `/proc/config.gz` ou `/boot/config-`*<kernel-release-date>*. Vous pouvez exécuter `uname -r` pour trouver la valeur de *<kernel-release-date>*.

     Exemple : Pour monter le fichier de configuration à partir de`/boot/config-`*<kernel-release-date>*

     ```
     -v /boot/config-4.15.0-74-generic:/boot/config-4.15.0-74-generic \
     ```

     Exemple : Pour monter le fichier de configuration à partir de`proc/config.gz`

     ```
     -v /proc/config.gz:/proc/config.gz \
     ```

- Remplacez *<image-repository>*:*<tag>* dans la commande avec le nom du référentiel et la balise de l'image cible.

  Exemple : Pour pointer vers la dernière version du kitAWS IoT GreengrassLogiciel Core

  ```
  216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
  ```

  Pour obtenir la liste des images AWS IoT Greengrass Docker, exécutez la commande suivante.

  ```
  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>
```

- Remplacez *<image-repository>*:*<tag>* dans la commande avec le nom du référentiel et la balise de l'image cible.

  Exemple : Pour pointer vers la dernière version du kitAWS IoT GreengrassLogiciel Core

  ```
  216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
  ```

  Pour obtenir la liste des images AWS IoT Greengrass Docker, exécutez la commande suivante :

  ```
  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>
```

- Remplacez *<image-repository>*:*<tag>* dans la commande avec le nom du référentiel et la balise de l'image cible.

  Exemple : Pour pointer vers la dernière version du kitAWS IoT GreengrassLogiciel Core

  ```
  216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
  ```

  Pour obtenir la liste des images AWS IoT Greengrass Docker, exécutez la commande suivante :

  ```
  aws ecr list-images --region us-west-2 --registry-id 216483018798 --
  repository-name aws-iot-greengrass
  ```

> **Important**
>
> Lors du test avec IDT, n'incluez pas l'argument `--entrypoint /greengrass-entrypoint.sh \` utilisé pour exécuter l'image à des fins générales AWS IoT Greengrass.

4. Étape suivante: Configurez votreAWSinformations d'identification et`device.json`fichier.

Configurer le Dockerfile fourni par AWS IoT Greengrass

Procédez comme suit pour configurer l'image Docker créée à partir du Dockerfile AWS IoT Greengrass afin d'exécuter des tests IDT.

1. À partir de the section called “Logiciel AWS IoT Greengrass Docker”, téléchargez le package Dockerfile sur votre ordinateur hôte et extrayez-le.
2. Ouvrir `README.md`. Les trois étapes suivantes font référence aux sections de ce fichier.
3. Assurez-vous que vous remplissez les conditions requises indiquées dans la section Conditions préalables.
4. Utilisateurs Linux uniquement : Terminez le kitActiver la protection Symlink et HardlinketActiver le transfert réseau IPv4étapes.

5. Pour générer l'image Docker, suivez toutes les étapes de la sectionÉtape 1. Créez le kitAWS IoT GreengrassDocker Image. Revenez ensuite à cette rubrique pour continuer la configuration.
6. Pour exécuter le conteneur AWS IoT Greengrass, utilisez la commande de votre système d'exploitation :

   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>
   ```

   - Remplacez *<host-path-to-kernel-config-file>* par le chemin d'accès au fichier de configuration du noyau sur l'hôte et *<container-path>* par le chemin d'accès où le volume est monté dans le conteneur.

     Le fichier de configuration du noyau sur l'hôte se trouve généralement dans `/proc/config.gz` ou `/boot/config-`*<kernel-release-date>*. Vous pouvez exécuter `uname -r` pour trouver la valeur de *<kernel-release-date>*.

     Exemple : Pour monter le fichier de configuration à partir de`/boot/config-`*<kernel-release-date>*

     ```
     -v /boot/config-4.15.0-74-generic:/boot/config-4.15.0-74-generic \
     ```

     Exemple : Pour monter le fichier de configuration à partir de`proc/config.gz`

     ```
     -v /proc/config.gz:/proc/config.gz \
     ```

   - Remplacez *<image-repository>*:*<tag>* dans la commande avec le nom du référentiel et la balise de l'image cible.

     Exemple : Pour pointer vers la dernière version du kitAWS IoT GreengrassLogiciel Core

     ```
     216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
     ```

     Pour obtenir la liste des images AWS IoT Greengrass Docker, exécutez la commande suivante.

```
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>
```

- Remplacez *<image-repository>*:*<tag>* dans la commande avec le nom du référentiel et la balise de l'image cible.

  Exemple : Pour pointer vers la dernière version du kitAWS IoT GreengrassLogiciel Core

  ```
  216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
  ```

  Pour obtenir la liste des images AWS IoT Greengrass Docker, exécutez la commande suivante :

  ```
  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>
```

- Remplacez *<image-repository>*:*<tag>* dans la commande avec le nom du référentiel et la balise de l'image cible.

  Exemple : Pour pointer vers la dernière version du kitAWS IoT GreengrassLogiciel Core

  ```
  216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
  ```

Pour obtenir la liste des images AWS IoT Greengrass Docker, exécutez la commande suivante :

```
aws ecr list-images --region us-west-2 --registry-id 216483018798 --
repository-name aws-iot-greengrass
```

> **Important**
>
> Lors du test avec IDT, n'incluez pas l'argument `--entrypoint /greengrass-entrypoint.sh \` utilisé pour exécuter l'image à des fins générales AWS IoT Greengrass.

7. Étape suivante: Configurez votreAWSinformations d'identification et`device.json`fichier.

Dépannage de la configuration de votre conteneur Docker pour IDT pour AWS IoT Greengrass

Aidez-vous des informations suivantes pour résoudre les problèmes que vous êtes susceptible de rencontrer lors de l'exécution d'un conteneur Docker pour IDT afin de tester AWS IoT Greengrass.

AVERTISSEMENT : Erreur lors du chargement de config file : /home/user/config.json - stat /home/ / home/ <user>/.docker/config.json : autorisation refusée

Si vous obtenez cette erreur lors de l'exécution des commandes `docker` sous Linux, exécutez la commande suivante. Remplacez *`<user>`* dans la commande suivante par l'utilisateur qui exécute IDT.

```
sudo chown <user>:<user> /home/<user>/.docker -R
sudo chmod g+rwx /home/<user>/.docker -R
```

Facultatif : Configuration de votre appareil pour la qualification ML

IDT pour AWS IoT Greengrass fournit des tests de qualification Machine Learning pour valider le fait que vos appareils peuvent réaliser des inférences ML localement à l'aide de modèles formés dans le cloud.

Pour exécuter des tests de qualification ML, vous devez d'abord configurer vos appareils comme décrit dans the section called “Configurer votre appareil afin d'exécuter des tests IDT”. Ensuite,

suivez les étapes de cette rubrique pour installer les dépendances pour les frameworks ML que vous souhaitez exécuter.

IDT v3.1.0 ou version ultérieure est nécessaire pour exécuter les tests de qualification ML.

Installation des dépendances du framework ML

Toutes les dépendances du framework ML doivent être installées sous le répertoire `/usr/local/lib/python3.x/site-packages`. Pour vous assurer qu'elles sont installées dans le répertoire correct, nous vous recommandons d'utiliser les autorisations racine `sudo` lors de l'installation des dépendances. Les environnements virtuels ne sont pas pris en charge pour les tests de qualification.

> **Note**
>
> Si vous testez des fonctions Lambda exécutées avecconteneurisation(dansConteneur Greengrassmode), créant des liens symboliques pour les bibliothèques Python sous`/usr/local/lib/python3.x`n'est pas pris en charge. Pour éviter les erreurs, vous devez installer les dépendances dans le répertoire correct.

Suivez les étapes pour installer les dépendances pour votre framework cible :

- Installation des dépendances MxNet
- the section called “Install TensorFlow dépendances”
- Installation des dépendances DLR

Installation des dépendances Apache MxNet

Les tests de qualification IDT pour ce framework ont les dépendances suivantes :

- Python 3.6 ou Python 3.7.

> **Note**
>
> Si vous utilisez Python 3.6, vous devez créer un lien symbolique entre les fichiers binaires de Python 3.7 vers Python 3.6. Ceci configure votre appareil de sorte qu'il réponde aux exigence de Python pour AWS IoT Greengrass. Par exemple :

```
sudo ln -s path-to-python-3.6/python3.6 path-to-python-3.7/python3.7
```

- Apache MxNet v1.2.1 ou version ultérieure.
- NumPy. La version doit être compatible avec votre version MxNet.

Installation de MxNet

Suivez les instructions de la documentation MxNet pour installer MxNet.

> **Note**
>
> Si Python 2.x et Python 3.x sont tous deux installés sur votre appareil, utilisez Python 3.x dans les commandes que vous exécutez pour installer les dépendances.

Validation de l'installation de MxNet

Choisissez l'une des options suivantes pour valider l'installation de MxNet.

Option 1 : SSH dans votre appareil et exécuter des scripts

1. SSH dans votre appareil.
2. Exécutez les scripts suivants pour vérifier que les dépendances sont correctement installées.

   ```
   sudo python3.7 -c "import mxnet; print(mxnet.__version__)"
   ```

   ```
   sudo python3.7 -c "import numpy; print(numpy.__version__)"
   ```

   La sortie imprime le numéro de version et le script doit se terminer sans erreur.

Option 2 : Exécuter le test de dépendance IDT

1. Assurez-vous que le fichier `device.json` est configuré pour la qualification ML. Pour plus d'informations, consultez the section called “Configurer device.json pour la qualification ML”.
2. Exécutez le test des dépendances pour le framework.

```
devicetester_[linux | mac | win_x86-64] run-suite --group-id mldependencies --test-
id mxnet_dependency_check
```

Le récapitulatif du test affiche un résultat `PASSED` pour `mldependencies`.

Install TensorFlow dépendances

Les tests de qualification IDT pour ce framework ont les dépendances suivantes :

- Python 3.6 ou Python 3.7.

> **Note**
>
> Si vous utilisez Python 3.6, vous devez créer un lien symbolique entre les fichiers binaires de Python 3.7 vers Python 3.6. Ceci configure votre appareil de sorte qu'il réponde aux exigence de Python pour AWS IoT Greengrass. Par exemple :
>
> ```
> sudo ln -s path-to-python-3.6/python3.6 path-to-python-3.7/python3.7
> ```

- TensorFlow 1.x.

Installation de TensorFlow

Suivez les instructions de la section TensorFlow documentation à installer TensorFlow 1.xavec pipouà partir de source.

> **Note**
>
> Si Python 2.x et Python 3.x sont tous deux installés sur votre appareil, utilisez Python 3.x dans les commandes que vous exécutez pour installer les dépendances.

Validation du TensorFlow mise en place

Choisissez une des options suivantes pour valider le TensorFlow installation.

Option 1 : SSH dans votre appareil et exécuter un script

1. SSH dans votre appareil.
2. Exécutez le script suivant pour vérifier que la dépendance est correctement installée.

   ```
   sudo python3.7 -c "import tensorflow; print(tensorflow.__version__)"
   ```

   La sortie imprime le numéro de version et le script doit se terminer sans erreur.

Option 2 : Exécuter le test de dépendance IDT

1. Assurez-vous que le fichier `device.json` est configuré pour la qualification ML. Pour plus d'informations, consultez the section called “Configurer device.json pour la qualification ML”.
2. Exécutez le test des dépendances pour le framework.

   ```
   devicetester_[linux | mac | win_x86-64] run-suite --group-id mldependencies --test-
   id tensorflow_dependency_check
   ```

   Le récapitulatif du test affiche un résultat `PASSED` pour `mldependencies`.

Installez Amazon SageMaker Dépendances Neo Deep Learning Runtime (DLR)

Les tests de qualification IDT pour ce framework ont les dépendances suivantes :

- Python 3.6 ou Python 3.7.

  > **Note**
  >
  > Si vous utilisez Python 3.6, vous devez créer un lien symbolique entre les fichiers binaires de Python 3.7 vers Python 3.6. Ceci configure votre appareil de sorte qu'il réponde aux exigence de Python pour AWS IoT Greengrass. Par exemple :
  >
  > ```
  > sudo ln -s path-to-python-3.6/python3.6 path-to-python-3.7/python3.7
  > ```

- SageMaker Neo DLR.
- numpy.

Après avoir installé les dépendances de test DLR, vous devez compiler le modèle.

Installation de DLR

Suivez les instructions de la documentation DLR pour installer le Neo DLR.

> **Note**
>
> Si Python 2.x et Python 3.x sont tous deux installés sur votre appareil, utilisez Python 3.x dans les commandes que vous exécutez pour installer les dépendances.

Validation de l'installation de DLR

Choisissez une des options suivantes pour valider l'installation de DLR.

Option 1 : SSH dans votre appareil et exécuter des scripts

1. SSH dans votre appareil.
2. Exécutez les scripts suivants pour vérifier que les dépendances sont correctement installées.

   ```
   sudo python3.7 -c "import dlr; print(dlr.__version__)"
   ```

   ```
   sudo python3.7 -c "import numpy; print(numpy.__version__)"
   ```

   La sortie imprime le numéro de version et le script doit se terminer sans erreur.

Option 2 : Exécuter le test de dépendance IDT

1. Assurez-vous que le fichier `device.json` est configuré pour la qualification ML. Pour plus d'informations, consultez the section called “Configurer device.json pour la qualification ML”.
2. Exécutez le test des dépendances pour le framework.

   ```
   devicetester_[linux | mac | win_x86-64] run-suite --group-id mldependencies --test-id dlr_dependency_check
   ```

   Le récapitulatif du test affiche un résultat `PASSED` pour `mldependencies`.

Compilation du modèle DLR

Vous devez compiler le modèle DLR avant de pouvoir l'utiliser pour les tests de qualification ML. Pour les étapes, choisissez une des options suivantes :

Option 1 : Utiliser Amazon SageMaker pour compiler le modèle

Suivez ces étapes pour utiliser SageMaker pour compiler le modèle ML fourni par IDT. Ce modèle est préformé avec Apache MxNet.

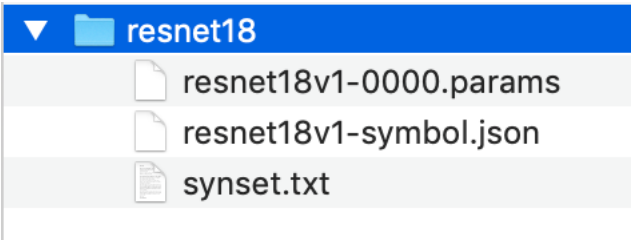
1. Vérifiez que le type de votre appareil est pris en charge par SageMaker. Pour plus d'informations, consultez le .options d'équipement cibleleAmazon SageMaker API Reference. Si le type de votre appareil n'est pas actuellement pris en charge par SageMaker, suivez les étapes dethe section called “Option 2 : Utiliser TVM pour compiler le modèle DLR”.

   > **Note**
   >
   > Exécuter le test DLR avec un modèle compilé par SageMaker peut prendre 4 ou 5 minutes. N'arrêtez pas IDT au cours de cette période.

2. Téléchargez le fichier tarball qui contient le modèle MxNet préformé non compilé pour DLR :

   - dlr-noncompiled-model-1.0.tar.gz

3. Décompressez le fichier tarball. Cette commande génère la structure de répertoire suivante.

4. Déplacez `synset.txt` hors du répertoire `resnet18`. Notez le nouvel emplacement. Vous copierez ce fichier dans le répertoire du modèle compilé ultérieurement.
5. Compressez le contenu du répertoire `resnet18`.

   ```
   tar cvfz model.tar.gz resnet18v1-symbol.json resnet18v1-0000.params
   ```

6. Chargez le fichier compressé sur un compartiment Amazon S3 de votreCompte AWS, puis suivez la procédure décrite dansCompilez un modèle (Console)pour créer une tâche de compilation.

a. Pour Configuration d'entrée, utilisez les valeurs suivantes :

   - Pour Configuration d'entrée de données, entrez `{"data": [1, 3, 224, 224]}`.
   - Pour Cadre de machine learning, choisissez `MXNet`.

b. Pour Configuration de sortie, utilisez les valeurs suivantes :

   - PourEmplacement de sortie S3, entrez le chemin d'accès au compartiment Amazon S3 ou au dossier dans lequel vous souhaitez stocker le modèle compilé.
   - Pour Périphérique cible, choisissez votre type d'appareil.

7. Téléchargez le modèle compilé à partir de l'emplacement de sortie spécifié, puis décompressez le fichier.
8. Copiez `synset.txt` dans le répertoire du modèle compilé.
9. Renommez le répertoire du modèle compilé en `resnet18`.

   Votre répertoire de modèle compilé doit avoir la structure de répertoire suivante.

Option 2 : Utiliser TVM pour compiler le modèle DLR

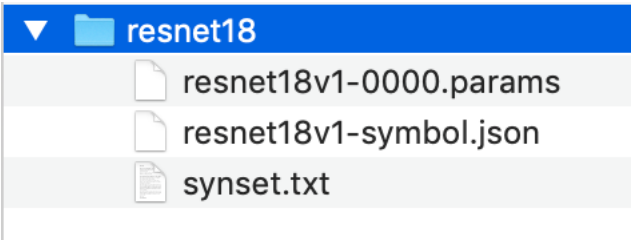
Suivez ces étapes pour utiliser TVM afin de compiler le modèle ML fourni par IDT. Ce modèle étant préformé avec Apache MxNet, vous devez installer MxNet sur l'ordinateur ou l'appareil sur lequel vous compilez le modèle. Pour installer MxNet, suivez les instructions de la documentation MxNet.

> **Note**
>
> Nous vous recommandons de compiler le modèle sur votre appareil cible. Cette pratique est facultative, mais elle peut aider à assurer la compatibilité et à atténuer les problèmes potentiels.

1. Téléchargez le fichier tarball qui contient le modèle MxNet préformé non compilé pour DLR :

   - dlr-noncompiled-model-1.0.tar.gz

2. Décompressez le fichier tarball. Cette commande génère la structure de répertoire suivante.



3. Suivez les instructions de la documentation TVM pour créer et installer TVM à partir de la source pour votre plateforme.

4. Une fois TVM créé, exécutez la compilation TVM pour le modèle resnet18. Les étapes suivantes sont basées sur le didacticiel Quick Start Tutorial for Compiling Deep Learning Models, de la documentation TVM.

   a. Ouvrez le fichier `relay_quick_start.py` à partir du référentiel TVM cloné.

   b. Mettez à jour le code qui définit un réseau neuronal en relais. Vous pouvez utiliser une des options suivantes :

      - Option 1 : Utilisez`mxnet.gluon.model_zoo.vision.get_model`Pour obtenir le module et les paramètres de relais :

```
from mxnet.gluon.model_zoo.vision import get_model
block = get_model('resnet18_v1', pretrained=True)
mod, params = relay.frontend.from_mxnet(block, {"data": data_shape})
```

      - Option 2 : À partir du modèle non compilé que vous avez téléchargé à l'étape 1, copier les fichiers suivants dans le même répertoire que le fichier`relay_quick_start.py`dans le fichier. Ces fichiers contiennent le module et les paramètres de relais.

        - `resnet18v1-symbol.json`
        - `resnet18v1-0000.params`

   c. Mettez à jour le code qui enregistre et charge le module compilé pour utiliser le code suivant.

```
from tvm.contrib import util
path_lib = "deploy_lib.so"
#  Export the model library based on your device architecture
```

```
lib.export_library("deploy_lib.so", cc="aarch64-linux-gnu-g++")
with open("deploy_graph.json", "w") as fo:
    fo.write(graph)
with open("deploy_param.params", "wb") as fo:
    fo.write(relay.save_param_dict(params))
```

d. Créez le modèle :

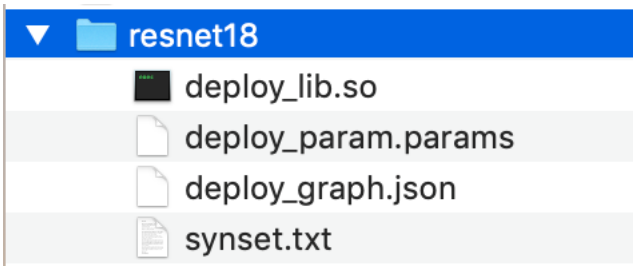
```
python3 tutorials/relay_quick_start.py --build-dir ./model
```

Cette commande génère les fichiers suivants.

- `deploy_graph.json`
- `deploy_lib.so`
- `deploy_param.params`

5. Copiez les fichiers du modèle généré dans un répertoire nommé `resnet18`. Ceci est votre répertoire de modèle compilé.
6. Copiez le répertoire de modèle compilé sur votre ordinateur hôte. Copiez ensuite `synset.txt` à partir du modèle non compilé que vous avez téléchargé à l'étape 1 dans le répertoire de modèle compilé.

   Votre répertoire de modèle compilé doit avoir la structure de répertoire suivante.



Ensuite, vous devez .configurez votreAWSinformations d'identification et`device.json`fichier.

Configurez les paramètres IDT pour exécuter leAWS IoT Greengrasssuite de qualification

Avant d'exécuter des tests, vous devez configurer les paramètres pourAWSinformations d'identification et appareils sur votre ordinateur hôte.

Configuration de vos informations d'identification pour l'AWS

Vous devez configurer les informations d'identification utilisateur IAM dans le fichier*<device-tester-extract-location>* `/configs/config.json`dans le fichier. Utilisez les informations d'identification pour l'IDT pour l'utilisateur AWS IoT Greengrass créé dans the section called “Créez et configurez un Compte AWS”. Vous pouvez spécifier vos informations d'identification de deux manières :

- Fichier d'informations d'identification.
- Variables d’environnement

ConfigurationAWSinformations d'identification à l'aide d'un fichier

IDT utilise le même fichier d'informations d'identification que l'AWS CLI. Pour de plus amples informations, veuillez consulter Fichiers de configuration et d'informations d'identification.

L'emplacement du fichier d'informations d'identification varie en fonction du système d'exploitation que vous utilisez :

- macOS, Linux : `~/.aws/credentials`
- Windows: `C:\Users\`*`UserName`*`\.aws\credentials`

Ajoutez votreAWSInformations d'identification pour le kit`credentials`fichier au format suivant :

```
[default]
aws_access_key_id = <your_access_key_id>
aws_secret_access_key = <your_secret_access_key>
```

Pour configurer IDT pourAWS IoT Greengrassà utiliserAWSInformations d'identification de votre fichier`credentials`, modifiez votre fichier`config.json`fichier comme suit :

```
{
 "awsRegion": "us-west-2",
 "auth": {
  "method": "file",
  "credentials": {
   "profile": "default"
  }
 }
```

```
}
```

> **Note**
>
> Si vous n'utilisez pas le kit`default` AWSassurez-vous de changer le nom du profil dans votre profil`config.json`dans le fichier. Pour de plus amples informations, veuillez consulter Profils nommés.

ConfigurationAWSinformations d'identification avec des variables d'environnement

Les variables d'environnement sont des variables gérées par le système d'exploitation et utilisées par les commandes du système. Elles ne sont pas enregistrées si vous fermez la session SSH. IDT pourAWS IoT Greengrasspeut utiliser le kit`AWS_ACCESS_KEY_ID`et`AWS_SECRET_ACCESS_KEY`variables d'environnement pour stocker votreAWSInformations d'identification .

Pour définir ces variables sous Linux, macOS ou Unix, utilisez export:

```
export AWS_ACCESS_KEY_ID=<your_access_key_id>
export AWS_SECRET_ACCESS_KEY=<your_secret_access_key>
```

Pour définir ces variables sous Windows, utilisez set :

```
set AWS_ACCESS_KEY_ID=<your_access_key_id>
set AWS_SECRET_ACCESS_KEY=<your_secret_access_key>
```

Pour configurer l'IDT afin que l'outil utilise les variables d'environnement, modifiez la section `auth` dans votre fichier `config.json`. Voici un exemple :

```
{
 "awsRegion": "us-west-2",
 "auth": {
  "method": "environment"
 }
}
```

Configurer device.json

Outre le kitAWSInformations d'identification, IDT pourAWS IoT Greengrassa besoin d'informations sur les appareils sur lesquels les tests sont effectués (par exemple, l'adresse IP, les informations de connexion, le système d'exploitation et l'architecture d'UC).

Vous pouvez fournir ces informations à l'aide du modèle `device.json` situé dans *<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**
>
> Spécifiez `privKeyPath` uniquement si `method` est défini sur `pki`.
> Spécifiez `password` uniquement si `method` est défini sur `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>"
        }
      }
    ]
  }
]
```

Tous les champs qui contiennent des valeurs sont requis, comme indiqué ici :

`id`

ID alphanumérique défini par l'utilisateur qui identifie de façon unique un ensemble d'appareils appelé un groupe d'appareils. Le matériel doit être identique pour les appareils d'un même groupe. Lorsque vous exécutez une suite de tests, les appareils du groupe sont utilisés pour paralléliser la charge de travail. Plusieurs appareils sont utilisés pour exécuter différents tests.

`sku`

Valeur alphanumérique qui identifie de façon unique l'appareil que vous testez. La référence est utilisée pour effectuer le suivi des cartes qualifiées.

> **Note**
>
> Si vous souhaitez ajouter votre carte dans le catalogue d'appareils AWS Partner, la référence que vous indiquez ici doit correspondre à celle indiquée pendant le processus d'élaboration de la liste.

`features`

Un tableau contenant les fonctions prises en charge de l'appareil. Toutes les fonctionnalités sont requises.

`os` et `arch`

Combinaisons d'architecture et de systèmes d'exploitation (OS) prises en charge :

- `linux`, `x86_64`
- `linux`, `armv6l`
- `linux`, `armv7l`
- `linux`, `aarch64`
- `ubuntu`, `x86_64`
- `openwrt`, `armv7l`
- `openwrt`, `aarch64`

> **Note**
>
> Si vous utilisez IDT pour testerAWS IoT Greengrasss'exécutant dans un conteneur Docker, seule l'architecture Docker x86_64 est prise en charge.

`container`

Valide que l'appareil répond à toutes les exigences logicielles et matérielles requises pour exécuter des fonctions Lambda en mode conteneur sur un noyau Greengrass.

La valeur valide est`yes`ou`no`.

`docker`

Valide que l'appareil répond à toutes les dépendances techniques requises pour pouvoir utiliser le connecteur de déploiement d'application Greengrass Docker pour exécuter des conteneurs.

La valeur valide est`yes`ou`no`.

`streamManagement`

Valide que l'appareil est conforme à toutes les dépendances techniques requises pour pouvoir exécuter le gestionnaire de flux AWS IoT Greengrass.

La valeur valide est`yes`ou`no`.

`hsi`

Vérifie que la bibliothèque partagée HSI fournie peut interagir avec le module de sécurité matérielle (HSM) et qu'elle implémente correctement les API PKCS # 11 requises. La bibliothèque partagée/HSM doit signer une CSR, effectuer des opérations TLS et fournir les longueurs de clé et l'algorithme de clé publique corrects.

La valeur valide est`yes`ou`no`.

`ml`

Valide que l'appareil est conforme à toutes les dépendances techniques requises pour pouvoir exécuter une inférence ML localement.

La valeur valide peut être une combinaison quelconque de`mxnet`,`tensorflow`,`dlr`, et`no`(par exemple,`mxnet`,`mxnet,tensorflow`,`mxnet,tensorflow,dlr`, ou`no`).

`mlLambdaContainerizationMode`

Valide que l'appareil est conforme à toutes les dépendances techniques requises pour pouvoir exécuter une inférence ML en mode conteneur sur un appareil Greengrass.

La valeur valide est`container`,`process`, ou`both`.

`processor`

Valide que l'appareil répond à toutes les exigences matérielles requises pour le type de processeur spécifié.

La valeur valide est`cpu`ou`gpu`.

> **Note**
>
> Si vous ne souhaitez pas utiliser la stratégie`container`,`docker`,`streamManager`,`hsi`, ou`ml`, vous pouvez définir la fonction correspondante`value`pour`no`.
> Docker prend uniquement en charge la qualification des fonctionnalités pour`streamManagement`et`ml`.

`machineLearning`

Facultatif. Informations de configuration pour les tests de qualification ML. Pour plus d'informations, consultez the section called “Configurer device.json pour la qualification ML”.

`hsm`

Facultatif. Informations de configuration pour les tests avec un module de sécurité matérielle (HSM) AWS IoT Greengrass. Sinon, la propriété `hsm` doit être omise. Pour plus d'informations, consultez Intégration de sécurité matérielle.

Cette propriété s'applique uniquement si `connectivity.protocol` est défini sur `ssh`.

`hsm.p11Provider`

Chemin absolu vers la bibliothèque libdl-loadable d'implémentation PKCS # 11.

`hsm.slotLabel`

Étiquette d'emplacement utilisée pour identifier le module matériel.

`hsm.slotUserPin`

PIN de l'utilisateur utilisé pour authentifier le fichierAWS IoT Greengrassau cœur du module.

`hsm.privateKeyLabel`

Étiquette utilisée pour identifier la clé dans le module matériel.

`hsm.openSSLEngine`

Chemin d'accès absolu au fichier `.so` du moteur OpenSSL qui active la prise en charge PKCS # 11 sur OpenSSL. C'est l'approche utilisée par l'agent des mises à jour OTA AWS IoT Greengrass.

`devices.id`

Un identificateur unique défini par l'utilisateur pour l'appareil testé.

`connectivity.protocol`

Le protocole de communication utilisé pour communiquer avec cet appareil. Actuellement, les seules valeurs prises en charge concernent `ssh` pour les appareils physiques et `docker` pour les conteneurs Docker.

`connectivity.ip`

L'adresse IP de l'appareil testé.

Cette propriété s'applique uniquement si `connectivity.protocol` est défini sur `ssh`.

`connectivity.containerId`

ID de conteneur ou nom du conteneur Docker en cours de test.

Cette propriété s'applique uniquement si `connectivity.protocol` est défini sur `docker`.

`connectivity.auth`

Informations d'authentification pour la connexion.

Cette propriété s'applique uniquement si `connectivity.protocol` est défini sur `ssh`.

`connectivity.auth.method`

Méthode d'authentification utilisée pour accéder à un appareil sur le protocole de connectivité donné.

Les valeurs prises en charge sont :

- `pki`
- `password`

`connectivity.auth.credentials`

Informations d'identification utilisées pour l'authentification.

`connectivity.auth.credentials.password`

Mot de passe utilisé pour se connecter à l'appareil à tester.

Cette valeur s'applique uniquement si `connectivity.auth.method` est défini sur `password`.

`connectivity.auth.credentials.privKeyPath`

Chemin complet de la clé privée utilisée pour se connecter à l'appareil testé.

Cette valeur s'applique uniquement si `connectivity.auth.method` est défini sur `pki`.

`connectivity.auth.credentials.user`

Nom d'utilisateur pour la connexion à l'appareil testé.

`connectivity.auth.credentials.privKeyPath`

Chemin complet à la clé privée utilisée pour se connecter à l'appareil testé.

`connectivity.port`

Facultatif. Numéro de port à utiliser pour les connexions SSH.

La valeur par défaut est 22.

Cette propriété s'applique uniquement si`connectivity.protocol`a la valeur`ssh`.

`greengrassLocation`

Emplacement du logiciel AWS IoT Greengrass Core sur vos appareils.

Pour les appareils physiques, cette valeur est utilisée uniquement lorsque vous utilisez une installation de AWS IoT Greengrass. Utilisez cet attribut pour indiquer à IDT qu'il doit utiliser le logiciel AWS IoT Greengrass Core installé sur vos appareils.

Lorsque vous exécutez des tests dans un conteneur Docker à partir de l'image Docker ou de Dockerfile fourni par AWS IoT Greengrass, définissez cette valeur sur `/greengrass`.

`kernelConfigLocation`

Facultatif. Chemin d'accès au fichier de configuration du noyau.AWS IoT Device Tester utilise ce fichier pour vérifier si les fonctions de noyau requises sont activées sur les appareils. S'il n'est pas spécifié, IDT utilise les chemins suivants pour rechercher le fichier de configuration du noyau :`/proc/config.gz`et`/boot/config-`*`<kernel-version>`*.AWS IoT Device Tester utilise le premier chemin qu'il trouve.

Configurer device.json pour la qualification ML

Cette section décrit les propriétés facultatives du fichier de configuration d'appareil qui s'appliquent à la qualification ML. Si vous prévoyez d'exécuter des tests pour la qualification ML, vous devez définir les propriétés qui s'appliquent à votre cas d'utilisation.

Vous pouvez utiliser le modèle `device-ml.json` pour définir les paramètres de configuration de votre appareil. Ce modèle contient les propriétés ML facultatives. Vous pouvez également utiliser

`device.json` et ajouter les propriétés de qualification ML. Ces fichiers se trouvent dans *<device-tester-extract-location>*`/configs` et incluent les propriétés de qualification ML. Si vous utilisez `device-ml.json`, vous devez renommer le fichier `device.json` avant d'exécuter des tests IDT.

Pour de plus amples informations sur les propriétés de configuration d'appareil qui ne s'appliquent pas à la qualification ML, veuillez consulter the section called “Configurer device.json”.

`ml` dans le tableau `features`

Les frameworks ML que votre carte prend en charge. Cette propriété nécessite IDT v3.1.0 ou version ultérieure.

- Si votre carte ne prend en charge qu'un seul framework, spécifiez le framework. Par exemple :

```
{
    "name": "ml",
    "value": "mxnet"
}
```

- Si votre carte prend en charge plusieurs frameworks, spécifiez les frameworks sous la forme d'une liste séparée par des virgules. Par exemple :

```
{
    "name": "ml",
    "value": "mxnet,tensorflow"
}
```

`mlLambdaContainerizationMode` dans le tableau `features`

Mode de conteneurisation que vous souhaitez utiliser pour le test. Cette propriété nécessite IDT v3.1.0 ou version ultérieure.

- Choisissez`process`pour exécuter le code d'inférence ML avec une fonction Lambda non conteneurisée. Cette option nécessite AWS IoT Greengrass v1.10.x ou version ultérieure.
- Choisissez`container`pour exécuter le code d'inférence ML avec une fonction Lambda conteneurisée.
- Choisissez `both` pour exécuter le code d'inférence ML avec les deux modes. Cette option nécessite AWS IoT Greengrass v1.10.x ou version ultérieure.

`processor` dans le tableau `features`

Indique l'accélérateur matériel pris en charge par votre carte. Cette propriété nécessite IDT v3.1.0 ou version ultérieure.

- Choisissez `cpu` si votre carte utilise un processeur de type CPU.
- Choisissez `gpu` si votre carte utilise un processeur de type GPU.

`machineLearning`

Facultatif. Informations de configuration pour les tests de qualification ML. Cette propriété nécessite IDT v3.1.0 ou version ultérieure.

`dlrModelPath`

Requis pour utiliser le framework `dlr`. Chemin absolu vers le répertoire de votre modèle compilé DLR, qui doit être nommé `resnet18`. Pour plus d'informations, consultez the section called “Compilation du modèle DLR”.

> **Note**
>
> Voici un exemple de chemin d'accès sur macOS : `/Users/<user>/Downloads/resnet18`.

`environmentVariables`

Tableau de paires clé-valeur qui peuvent transmettre dynamiquement des paramètres aux tests d'inférence ML. Facultatif pour les appareils avec processeur de type CPU. Vous pouvez utiliser cette section pour ajouter des variables d'environnement spécifiques au framework requises par votre type d'appareil. Pour de plus amples informations sur ces exigences, veuillez consulter le site Web officiel du framework ou de l'appareil. Par exemple, pour exécuter des tests d'inférence MxNet sur certains appareils, les variables d'environnement suivantes peuvent être requises.

```
"environmentVariables": [
    ...
    {
        "key": "PYTHONPATH",
        "value": "$MXNET_HOME/python:$PYTHONPATH"
    },
    {
```

```
        "key": "MXNET_HOME",
        "value": "$HOME/mxnet/"
    },
    ...
]
```

> **Note**
>
> Le champ `value` peut varier en fonction de votre installation MxNet.

Si vous testez des fonctions Lambda exécutées avecconteneurisationSur les appareils GPU, ajoutez des variables d'environnement pour la bibliothèque GPU. Cela permet au GPU d'effectuer des calculs. Pour utiliser des bibliothèques GPU différentes, veuillez consulter la documentation officielle de la bibliothèque ou de l'appareil.

> **Note**
>
> Configurez les clés suivantes si la fonction `mlLambdaContainerizationMode` est définie sur `container` ou `both`.

```
"environmentVariables": [
    {
        "key": "PATH",
        "value": "<path/to/software/bin>:$PATH"
    },
    {
        "key": "LD_LIBRARY_PATH",
        "value": "<path/to/ld/lib>"
    },
    ...
]
```

`deviceResources`

Requis par les appareils GPU. Contientressources localesaccessibles par les fonctions Lambda. Utilisez cette section pour ajouter des ressources d'appareil et de volume locales.

- Pour les ressources d'appareil, spécifiez `"type": "device"`. Pour les appareils GPU, les ressources d'appareil doivent être des fichiers d'appareil liés au GPU sous `/dev`.

> **Note**
>
> Le répertoire `/dev/shm` est une exception. Il peut être configuré en tant que ressource de volume uniquement.

- Pour les ressources de volume, spécifiez `"type": "volume"`.

Exécutez leAWS IoT GreengrassSuite de qualification

Après avoir défini la configuration requise, vous pouvez démarrer les tests. L'exécution de l'ensemble de la suite de tests dépend de votre matériel. Pour référence, il faut environ 30 minutes pour terminer la suite de tests complète sur un Raspberry Pi 3B.

Les exemples de commande `run-suite` suivants vous montrent comment exécuter les tests de qualification pour un groupe de périphériques. Un groupe de périphériques est un ensemble de périphériques identiques.

IDT v3.0.0 and later

Exécutez tous les groupes de tests dans une suite de tests spécifiée.

```
devicetester_[linux | mac | win_x86-64] run-suite --suite-id GGQ_1.0.0 --pool-
id <pool-id>
```

Utilisez la commande `list-suites` pour répertorier les suites de tests qui se trouvent dans le dossier `tests`.

Exécutez un groupe de tests spécifique dans une suite de tests.

```
devicetester_[linux | mac | win_x86-64] run-suite --suite-id GGQ_1.0.0 --group-
id <group-id> --pool-id <pool-id>
```

Utilisez la commande `list-groups` pour répertorier les groupes de tests dans une suite de tests.

Exécutez un cas de test spécifique dans un groupe de tests.

```
devicetester_[linux | mac | win_x86-64] run-suite --group-id <group-id> --test-
id <test-id>
```

Exécutez plusieurs scénarios de test dans un groupe de tests.

```
devicetester_[linux | mac | win_x86-64] run-suite --group-id <group-id> --test-
id <test-id1>,<test-id2>
```

Répertoriez les scénarios de test dans un groupe de tests.

```
devicetester_[linux | mac | win_x86-64] list-test-cases --group-id <group-id>
```

Les options de la commande `run-suite` sont facultatives. Par exemple, vous pouvez omettre `pool-id` si un seul groupe de périphériques est défini dans votre fichier `device.json`. Ou, vous pouvez omettre `suite-id` si vous souhaitez exécuter la dernière version de la suite de tests dans le dossier `tests`.

> **Note**
>
> IDT vous demande si une version de suite de tests plus récente est disponible en ligne. Pour plus d'informations, consultez the section called “Définition du comportement de mise à jour par défaut”.

Pour plus d'informations sur `run-suite` et d'autres commandes IDT, veuillez consulter the section called “Commandes IDT”.

IDT v2.3.0 and earlier

Exécutez tous les groupes de tests dans une suite spécifiée.

```
devicetester_[linux | mac | win_x86-64] run-suite --suite-id GGQ_1 --pool-
id <pool-id>
```

Exécutez un groupe de tests spécifique.

```
devicetester_[linux | mac | win_x86-64] run-suite --suite-id GGQ_1 --group-
id <group-id> --pool-id <pool-id>
```

`suite-id` et `pool-id` sont facultatifs si vous exécutez une suite de tests unique sur un groupe de périphériques unique. Cela signifie que vous n'avez qu'un seul groupe de périphériques défini dans votre fichier `device.json`.

Vérifiez les dépendances de Greengrass

Nous vous recommandons d'exécuter le groupe de test du vérificateur de dépendance pour vous assurer que toutes les dépendances Greengrass sont installées avant d'exécuter des groupes de test associés. Par exemple :

- Exécutez `ggcdependencies` avant les groupes de tests de qualification du noyau.
- Exécutez `containerdependencies` avant les groupes de tests spécifiques au conteneur.
- Exécutez `dockerdependencies` avant les groupes de tests spécifiques à Docker.
- Exécutez `ggcstreammanagementdependencies` avant les groupes de tests spécifiques au gestionnaire de flux.

Définition du comportement de mise à jour par défaut

Lorsque vous démarrez un test, IDT recherche en ligne une version plus récente de la suite de tests. Si une version est disponible, IDT vous invite à utiliser la dernière version disponible. Vous pouvez définir l'indicateur `upgrade-test-suite` (ou u) pour contrôler le comportement de mise à jour par défaut. Les valeurs valides sont :

- y. IDT télécharge et utilise la dernière version disponible.
- n (default). IDT utilise la version spécifiée dans l'option `suite-id`. Si`suite-id`n'est pas spécifié, IDT utilise la dernière version dans le`tests`folder.

Si vous n'incluez pas l'indicateur `upgrade-test-suite`, IDT vous invite lorsqu'une mise à jour est disponible et attend votre réponse (y ou n) pendant 30 secondes. Si vous n’entrez pas de réponse, la valeur par défaut n est utilisée et l’exécution des tests se poursuit.

Les exemples suivants illustrent les cas d'utilisation courants de cette fonctionnalité :

Utiliser automatiquement les derniers tests disponibles pour un groupe de tests.

```
devicetester_linux run-suite -u y --group-id mqtt --pool-id DevicePool1
```

Exécuter des tests dans une version de suite de tests spécifique.

```
devicetester_linux run-suite -u n --suite-id GGQ_1.0.0 --group-id mqtt --pool-id
 DevicePool1
```

Demander des mises à jour lors de l'exécution.

```
devicetester_linux run-suite --pool-id DevicePool1
```

Commandes IDT pour AWS IoT Greengrass

Les commandes IDT se trouvent dans le répertoire *<device-tester-extract-location>*`/bin`. Utilisez-les pour les opérations suivantes :

IDT v3.0.0 and later

`help`

Répertorie les informations sur la commande spécifiée.

`list-groups`

Répertorie les groupes dans une suite de tests donnée.

`list-suites`

Répertorie les suites de tests disponibles.

`list-supported-products`

Répertorie les produits pris en charge, dans ce cas les versions AWS IoT Greengrass, et les versions de la suite de tests pour la version IDT actuelle.

`list-test-cases`

Répertorie les cas de tests d'un groupe de tests donné. L'option suivante est prise en charge :

- `group-id`. Groupe de tests à rechercher. Cette option est obligatoire et doit spécifier un groupe unique.

`run-suite`

Exécute une suite de tests sur un groupe d'appareils. Voici quelques options prises en charge :

- `suite-id`. Version de la suite de tests à exécuter. Si celle-ci n’est pas spécifiée, IDT utilise la dernière version dans le dossier `tests`.
- `group-id`. Les groupes de tests à exécuter, sous la forme d'une liste séparée par des virgules. Si cette option n'est pas spécifiée, IDT exécute tous les groupes de tests de la suite de tests.

- `test-id`. Cas de test à exécuter, sous la forme d'une liste séparée par des virgules. Lorsqu'il est spécifié, `group-id` doit spécifier un seul groupe.
- `pool-id`. Le groupe de périphériques à tester. Vous devez spécifier un groupe si plusieurs groupes de périphériques sont définis dans votre fichier `device.json`.
- `upgrade-test-suite`. Contrôle la gestion des mises à jour de version de la suite de tests. À partir de IDT v3.0.0, IDT vérifie en ligne les versions mises à jour de la suite de tests. Pour plus d'informations, consultez the section called “Versions de la suite de tests”.
- `stop-on-first-failure`. Configure IDT pour arrêter l'exécution à la première défaillance. Cette option doit être utilisée avec `group-id` pour déboguer les groupes de tests spécifiés. N'utilisez pas cette option lors de l'exécution d'une suite de tests complète pour générer un rapport de qualification.
- `update-idt`. Définit la réponse pour l'invite de mise à jour d'IDT.YEnter permet d'arrêter l'exécution du test si IDT détecte qu'il existe une version plus récente.Nalors que l'entrée continue l'exécution du test.
- `update-managed-policy`. Entrer `Y` permet d'arrêter l'exécution du test si IDT détecte que la stratégie gérée de l'utilisateur n'est pas mise à jour. Entrer `N` permet de poursuivre l'exécution du test.

Pour de plus amples informations sur les options `run-suite`, utilisez l' option `help` suivante :

```
devicetester_[linux | mac | win_x86-64] run-suite -h
```

IDT v2.3.0 and earlier

`help`

Répertorie les informations sur la commande spécifiée.

`list-groups`

Répertorie les groupes dans une suite de tests donnée.

`list-suites`

Répertorie les suites de tests disponibles.

`run-suite`

Exécute une suite de tests sur un groupe d'appareils.

Pour de plus amples informations sur les options `run-suite`, utilisez l' option `help` suivante :

```
devicetester_[linux | mac | win_x86-64] run-suite -h
```

Présentation des résultats et des journaux

Cette section explique comment afficher et interpréter les journaux et les rapports de résultats IDT.

Affichage des résultats

Lorsqu'il s'exécute, IDT écrit les erreurs sur la console, les fichiers journaux et les rapports de tests. Une fois que l'outil a terminé la suite de tests, il génère deux rapports de tests. Ces rapports se trouvent à l'emplacement *<device-tester-extract-location>*`/results/`*<execution-id>*`/`. Les deux rapports capturent les résultats de l'exécution de la suite de tests de qualification.

Le`awsiotdevicetester_report.xml`est le rapport de test de qualification que vous soumettez àAWSpour répertorier votre appareil dans leAWS PartnerDispositif Catalog. Ce rapport contient les éléments suivants :

- La version IDT.
- La version AWS IoT Greengrass qui a été testée.
- La référence et le nom du groupe d'appareils spécifié dans le fichier `device.json`.
- Les caractéristiques du groupe d'appareils spécifié dans le fichier `device.json`.
- Le récapitulatif des résultats des tests.
- La répartition des résultats des tests pour les bibliothèques qui ont été testées en fonction des caractéristiques de l'appareil (par exemple, l'accès aux ressources locales, shadow, MQTT, etc).

Le rapport `GGQ_Result.xml` est au format JUnit XML. Vous pouvez intégrer des plateformes de déploiement/d'intégration continues tels que Jenkins, Bamboo, etc. Ce rapport contient les éléments suivants :

- Un récapitulatif des résultats des tests.
- Une répartition des résultats des tests en fonction de la fonctionnalité AWS IoT Greengrass testée.

Interprétation des rapports IDT

La section de rapport dans les fichiers `awsiotdevicetester_report.xml` ou `awsiotdevicetester_report.xml` répertorie les tests qui ont été exécutés ainsi que leurs résultats.

La première balise XML `<testsuites>` contient le résumé de l'exécution des tests. Par exemple :

```
<testsuites name="GGQ results" time="2299" tests="28" failures="0" errors="0"
 disabled="0">
```

Attributs utilisés dans la balise **`<testsuites>`**

`name`

Nom de la suite de tests.

`time`

Durée, en secondes, nécessaire pour exécuter la suite de qualification.

`tests`

Nombre de tests exécutés.

`failures`

Nombre de tests exécutés mais dont le résultat n'est pas probant.

`errors`

Nombre de tests qu'IDT n'a pas pu exécuter.

`disabled`

Cet attribut n'est pas utilisé et peut être ignoré.

Le fichier `awsiotdevicetester_report.xml` contient une balise `<awsproduct>` qui contient des informations relatives au produit testé et les caractéristiques du produit qui ont été validées par une suite de tests.

Attributs utilisés dans la balise **`<awsproduct>`**

`name`

Nom du produit testé.

`version`

Version du produit testé.

`features`

Caractéristiques validées. Les caractéristiques portant la mention `required` sont requises pour pouvoir envoyer votre carte en vue de sa certification. L'extrait de code suivant montre comment ces informations apparaissent dans le fichier `awsiotdevicetester_report.xml`.

```
<feature name="aws-iot-greengrass-no-container" value="supported" type="required"></
feature>
```

Les fonctionnalités marquées comme `optional` ne sont pas requises pour l'éligibilité. Les extraits suivants illustrent des fonctions facultatives.

```
<feature name="aws-iot-greengrass-container" value="supported" type="optional"></
feature>

<feature name="aws-iot-greengrass-hsi" value="not-supported" type="optional"></
feature>
```

S'il n'y a pas d'erreurs ou d'échecs de tests pour les fonctionnalités requises, votre appareil répond aux exigences techniques requises pour exécuter AWS IoT Greengrass et peut interagir avec les services AWS IoT. Si vous souhaitez mettre en vente votre appareil dans leAWS PartnerDevice Catalog, vous pouvez utiliser ce rapport comme preuve de qualification.

En cas d'erreurs ou d'échecs de tests, vous pouvez identifier les tests concernés à l'aide des balises XML `<testsuites>`. Les balises XML `<testsuite>` au sein de la balise `<testsuites>` montrent le récapitulatif des résultats d'un groupe de tests. Par exemple :

```
<testsuite name="combination" package="" tests="1" failures="0" time="161" disabled="0"
 errors="0" skipped="0">
```

Le format est similaire à la balise `<testsuites>`, mais avec un attribut appelé `skipped` qui n'est pas utilisé et qui ne peut pas être ignoré. Chaque balise XML `<testsuite>` inclut des balises `<testcase>` pour chaque test exécuté pour un groupe de tests. Par exemple :

```
<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>>
```

Attributs utilisés dans la balise **<testcase>**

`name`

Nom du test.

`attempts`

Nombre de fois où IDT a exécuté le test.

Lorsqu'un test échoue ou qu'une erreur se produit, les balises `<failure>` ou `<error>` sont ajoutées à la balise `<testcase>` avec des informations relatives au dépannage. Par exemple :

```
<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>
```

Affichage des journaux

IDT génère des journaux à partir de l'exécution du test dans *<devicetester-extract-location>*`/results/`*<execution-id>*`/logs`. Deux ensembles de journaux sont générés :

`test_manager.log`

Les journaux générés à partir du composant Gestionnaire de tests d'AWS IoT Device Tester (par exemple, les journaux liés à la configuration, au séquençage des tests et à la génération de rapports).

<test_case_id>`.log (for example, ota.log)`

Journaux du groupe de tests, y compris les journaux liés à l'appareil qui est testé. Lorsqu'un test échoue, un fichier tar.gz qui contient les journaux de l'appareil testé pour le test correspondant est créé (par exemple, `ota_prod_test_1_ggc_logs.tar.gz`).

Pour plus d'informations, consultez Résolution des problèmes liés à IDT pour AWS IoT Greengrass.

Utilisez IDT pour développer et exécuter vos propres suites de tests

Depuis IDT v4.0.0, IDT pourAWS IoT Greengrasscombine une configuration normalisée et un format de résultat avec un environnement de suite de tests qui vous permet de développer des suites de tests personnalisées pour vos appareils et vos logiciels de périphériques. Vous pouvez ajouter des tests personnalisés pour votre propre validation interne ou les fournir à vos clients pour vérification des appareils.

Utilisez IDT pour développer et exécuter des suites de tests personnalisées, comme suit :

Pour développer des suites de tests personnalisées

- Créez des suites de tests avec une logique de test personnalisée pour l'appareil Greengrass que vous souhaitez tester.
- Fournissez à IDT vos suites de tests personnalisées aux coureurs de test. Incluez des informations sur les configurations de paramètres spécifiques de vos suites de tests.

Pour exécuter des suites de tests personnalisées

- Configurez l'appareil que vous souhaitez tester.
- Implémentez les configurations de paramètres requises par les suites de tests que vous souhaitez utiliser.
- Utilisez IDT pour exécuter vos suites de tests personnalisées.
- Affichez les résultats des tests et les journaux d'exécution des tests exécutés par IDT.

Télécharger la dernière version d'AWS IoT Device Tester pour AWS IoT Greengrass

Télécharger leDernière versiond'IDT et extrayez le logiciel à un emplacement de votre système de fichiers pour lequel vous disposez d'autorisations de lecture et d'écriture.

> **Note**
>
> IDT ne prend pas en charge son exécution par plusieurs utilisateurs à partir d'un emplacement partagé, tel qu'un répertoire NFS ou un dossier partagé réseau Windows. Nous vous recommandons d'extraire le package IDT sur une unité locale et d'exécuter le fichier binaire IDT sur votre station de travail locale.

Pour Windows, la limitation de la longueur du chemin est de 260 caractères. Si vous utilisez Windows, décompressez IDT dans un répertoire racine comme `C:\` ou `D:\` afin que la longueur de vos chemins respecte la limite de 260 caractères.

Workflow de création de suite de tests

Les suites de tests se composent de trois types de fichiers :

- Fichiers de configuration JSON qui fournissent à IDT des informations sur l'exécution de la suite de tests.
- Testez les fichiers exécutables utilisés par IDT pour exécuter des scénarios de test.
- Fichiers supplémentaires nécessaires à l'exécution des tests.

Suivez les étapes de base suivantes pour créer des tests IDT personnalisés :

1. Création de fichiers JSON de configurationpour votre suite de tests.
2. Créer des exécutables de cas de testqui contiennent la logique de test de votre suite de tests.
3. Vérifiez et documentez leinformations de configuration requises pour les coureurs de testpour exécuter la suite de tests.
4. Vérifiez qu'IDT peut exécuter votre suite de tests et produireRésultats de testcomme prévu.

Pour créer rapidement un exemple de suite personnalisée et l'exécuter, suivez les instructions deDidacticiel : Générez et exécutez l'exemple de suite de tests IDT.

Pour commencer à créer une suite de tests personnalisée en Python, voirDidacticiel : Développer une suite de tests IDT simple.

Didacticiel : Générez et exécutez l'exemple de suite de tests IDT

LeAWS IoTLe téléchargement de Device Tester inclut le code source d'un exemple de suite de tests. Vous pouvez suivre ce didacticiel pour créer et exécuter l'exemple de suite de tests afin de comprendre comment utiliserAWS IoTDevice Tester pourAWS IoT Greengrasspour exécuter des suites de tests personnalisées.

Dans ce didacticiel, vous allez réaliser les étapes suivantes :

1. Création de la suite de tests d'échantillons
2. Utiliser IDT pour exécuter la suite de tests d'échantillons

Prérequis

Pour suivre ce didacticiel, vous aurez besoin des éléments suivants :

- Configuration requise de l'ordinateur hôte
  - Dernière version deAWS IoTDevice Tester
  - Python3.7 ou version ultérieure

    Pour vérifier la version de Python installée sur votre ordinateur, exécutez la commande suivante :

    ```
    python3 --version
    ```

    Sous Windows, si l'utilisation de cette commande renvoie une erreur, utilisez`python --version`à la place. Si le numéro de version renvoyé est 3.7 ou supérieur, exécutez la commande suivante dans un terminal Powershell pour définir`python3`comme alias pour`python`commande.

    ```
    Set-Alias -Name "python3" -Value "python"
    ```

    Si aucune information de version n'est renvoyée ou si le numéro de version est inférieur à 3,7, suivez les instructions deTéléchargement de Pythonpour installer Python 3.7+. Pour plus d'informations, consultez le .Documentation Python.
  - urllib3

    Pour vérifier que`urllib3`est installé correctement, exécutez la commande suivante :

    ```
    python3 -c 'import urllib3'
    ```

    Si`urllib3`n'a pas encore été installé, exécutez la commande suivante pour l'installer :

    ```
    python3 -m pip install urllib3
    ```

- Exigences relatives aux dispositifs
  - Un appareil doté d'un système d'exploitation Linux et d'une connexion réseau au même réseau que votre ordinateur hôte.

    Nous vous recommandons d'utiliser unRaspberry Piavec Raspberry Pi OS Assurez-vous que vous avez configuréSSHsur votre Raspberry Pi pour se connecter à distance.

Configuration des informations sur les appareils pour IDT

Configurez les informations de votre appareil pour qu'IDT puisse exécuter le test. Vous devez mettre à jour la`device.json`modèle situé dans le*<device-tester-extract-location>*/`configs`dossier contenant les informations suivantes.

```
[
  {
    "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>"
            }
          }
        }
      }
    ]
  }
]
```

Dans`devices`, fournissez les informations suivantes :

`id`

Un identificateur unique défini par l'utilisateur pour votre appareil.

`connectivity.ip`

Adresse IP de votre appareil.

`connectivity.port`

Facultatif. Numéro de port à utiliser pour les connexions SSH à votre appareil.

`connectivity.auth`

Informations d'authentification pour la connexion.

Cette propriété s'applique uniquement si `connectivity.protocol` est défini sur `ssh`.

`connectivity.auth.method`

Méthode d'authentification utilisée pour accéder à un appareil sur le protocole de connectivité donné.

Les valeurs prises en charge sont :

- `pki`
- `password`

`connectivity.auth.credentials`

Informations d'identification utilisées pour l'authentification.

`connectivity.auth.credentials.user`

Le nom d'utilisateur utilisé pour se connecter à votre appareil.

`connectivity.auth.credentials.privKeyPath`

Chemin complet à la clé privée utilisée pour se connecter à votre appareil.

Cette valeur s'applique uniquement si `connectivity.auth.method` est défini sur `pki`.

`devices.connectivity.auth.credentials.password`

Mot de passe utilisé pour se connecter à votre appareil.

Cette valeur s'applique uniquement si `connectivity.auth.method` est défini sur `password`.

> **Note**
>
> Spécifiez `privKeyPath` uniquement si `method` est défini sur `pki`.
> Spécifiez `password` uniquement si `method` est défini sur `password`.

Création de la suite de tests d'échantillons

Le*<device-tester-extract-location>*`/samples/python`contient des exemples de fichiers de configuration, de code source et du kit SDK client IDT que vous pouvez combiner dans une suite de tests à l'aide des scripts de génération fournis. L'arborescence de répertoires suivante indique l'emplacement de ces exemples de fichiers :

```
<device-tester-extract-location>
### ...
### tests
### samples
#   ### ...
#   ### python
#       ### configuration
#       ### src
#       ### build-scripts
#           ### build.sh
#           ### build.ps1
### sdks
    ### ...
    ### python
        ### idt_client
```

Pour créer la suite de tests, exécutez les commandes suivantes sur votre ordinateur hôte :

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
```

Cela crée l'exemple de suite de tests dans le`IDTSampleSuitePython_1.0.0`dans le dossier`<device-tester-extract-location>/tests`folder. Passez en revue les fichiers dans le`IDTSampleSuitePython_1.0.0`pour comprendre comment l'exemple de suite de tests est structuré et voir divers exemples d'exécutables de cas de test et de fichiers JSON de configuration de test.

Étape suivante: Utilisez IDT pourExécutez la suite de tests d'échantillonsque vous avez créé.

Utiliser IDT pour exécuter la suite de tests d'échantillons

Pour exécuter l'exemple de suite de tests, exécutez les commandes suivantes sur votre ordinateur hôte :

```
cd <device-tester-extract-location>/bin
./devicetester_[linux | mac | win_x86-64] run-suite --suite-id IDTSampleSuitePython
```

IDT exécute la suite de tests d'exemple et diffuse les résultats vers la console. Lorsque le test est terminé, les informations suivantes s'affichent :

```
========== 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
```

Dépannage

Utilisez les informations suivantes pour résoudre les problèmes liés à la réalisation du didacticiel.

Le cas de test ne s'exécute pas correctement

Si le test ne s'exécute pas correctement, IDT diffuse les journaux d'erreurs vers la console, ce qui peut vous aider à résoudre le problème de l'exécution du test. Assurez-vous que vous rencontrez tous lesexigencespour ce didacticiel.

Impossible de se connecter à l'appareil testé

Vérifiez les paramètres suivants :

- Votre`device.json`contient l'adresse IP, le port et les informations d'authentification correctes.
- Vous pouvez vous connecter à votre appareil via SSH à partir de votre ordinateur hôte.

Didacticiel : Développer une suite de tests IDT simple

Une suite de tests combine les éléments suivants :

- Exécutables de test contenant la logique de test
- Fichiers de configuration JSON qui décrivent la suite de tests

Ce didacticiel vous montre comment utiliser IDT pourAWS IoT Greengrasspour développer une suite de tests Python contenant un seul cas de test. Dans ce didacticiel, vous allez réaliser les étapes suivantes :

1. Création d'un répertoire de suite de tests
2. Créez des fichiers de configuration JSON
3. Créer l'exécutable du cas de test
4. Exécutez la suite de tests

Prérequis

Pour suivre ce didacticiel, vous aurez besoin des éléments suivants :

- Configuration requise de l'ordinateur hôte
  - Dernière version deAWS IoTDevice Tester
  - Python3.7 ou version ultérieure

    Pour vérifier la version de Python installée sur votre ordinateur, exécutez la commande suivante :

```
python3 --version
```

Sous Windows, si l'utilisation de cette commande renvoie une erreur, utilisez`python --version`à la place. Si le numéro de version renvoyé est 3.7 ou supérieur, exécutez la commande suivante dans un terminal Powershell pour définir`python3`comme alias pour`python`commande.

```
Set-Alias -Name "python3" -Value "python"
```

Si aucune information de version n'est renvoyée ou si le numéro de version est inférieur à 3,7, suivez les instructions deTéléchargement de PythonPour installer Python 3.7+. Pour plus d'informations, consultez le .Documentation Python.

- urllib3

  Pour vérifier que`urllib3`est installé correctement, exécutez la commande suivante :

  ```
  python3 -c 'import urllib3'
  ```

  Si`urllib3`n'a pas encore été installé, exécutez la commande suivante pour l'installer :

  ```
  python3 -m pip install urllib3
  ```

- Exigences relatives aux dispositifs
  - Un appareil doté d'un système d'exploitation Linux et d'une connexion réseau au même réseau que votre ordinateur hôte.

    Nous vous recommandons d'utiliser unRaspberry Piavec Raspberry Pi OS. Veillez à configurerSSHsur votre Raspberry Pi pour pouvoir se connecter à distance.

Création d'un répertoire de suite de tests

IDT sépare logiquement les cas de test en groupes de tests au sein de chaque suite de tests. Chaque scénario de test doit se trouver à l'intérieur d'un groupe de tests. Pour ce didacticiel, créez un dossier appelé`MyTestSuite_1.0.0`et créez l'arborescence de répertoires suivante dans ce dossier :

```
MyTestSuite_1.0.0
### suite
    ### myTestGroup
        ### myTestCase
```

Créez des fichiers de configuration JSON

Votre suite de tests doit contenir ce qui suit :Fichiers de configuration JSON :

Fichiers JSON requis

`suite.json`

Contient des informations à propos de la suite de tests. Consultez Configurer suite.json.

`group.json`

Contient des informations à propos d'un groupe de tests. Vous devez créer un`group.json`pour chaque groupe de tests de votre suite de tests. Consultez Configurer group.json.

`test.json`

Contient des informations à propos d'un scénario de test. Vous devez créer un`test.json`pour chaque cas de test de votre suite de tests. Consultez Configurer test.json.

1. Dans`MyTestSuite_1.0.0/suite`, créez un`suite.json`File avec la structure suivante :

   ```
   {
       "id": "MyTestSuite_1.0.0",
       "title": "My Test Suite",
       "details": "This is my test suite.",
       "userDataRequired": false
   }
   ```

2. Dans`MyTestSuite_1.0.0/myTestGroup`, créez un`group.json`File avec la structure suivante :

   ```
   {
       "id": "MyTestGroup",
       "title": "My Test Group",
       "details": "This is my test group.",
       "optional": false
   ```

```
}
```

3. Dans`MyTestSuite_1.0.0/myTestGroup/myTestCase`, créez un`test.json`File avec la structure suivante :

```
{
    "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'arborescence du répertoire de votre`MyTestSuite_1.0.0`folder devrait ressembler à ce qui suit :

```
MyTestSuite_1.0.0
### suite
    ### suite.json
    ### myTestGroup
        ### group.json
        ### myTestCase
            ### test.json
```

Obtenez le SDK client IDT

Vous utilisez leKit SDK client IDTpour permettre à IDT d'interagir avec l'appareil testé et de signaler les résultats des tests. Pour ce didacticiel, vous allez utiliser la version Python du SDK.

De la*<device-tester-extract-location>*`/sdks/python/`, copiez le dossier`idt_client`dans votre dossier`MyTestSuite_1.0.0/suite/myTestGroup/myTestCase`folder.

Pour vérifier que le kit SDK a bien été copié, exécutez la commande suivante.

```
cd MyTestSuite_1.0.0/suite/myTestGroup/myTestCase
python3 -c 'import idt_client'
```

Créer l'exécutable du cas de test

Les exécutables de scénario de test contiennent la logique de test que vous voulez exécuter. Une suite de tests peut contenir plusieurs exécutables de cas de test. Pour ce didacticiel, vous allez créer un seul exécutable de cas de test.

1. Créez le fichier de la suite de tests.

   Dans`MyTestSuite_1.0.0/suite/myTestGroup/myTestCase`, créez un`myTestCase.py`Fichier comportant le contenu suivant :

   ```
   from idt_client import *

   def main():
       # Use the client SDK to communicate with IDT
       client = Client()

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

2. Utilisez les fonctions SDK client pour ajouter la logique de test suivante à votre`myTestCase.py`dans le fichier:

   a. Exécutez une commande SSH sur l'appareil testé.

      ```
      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. Envoyez le résultat du test à 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()
```

Configuration des informations sur les appareils pour IDT

Configurez les informations de votre appareil pour qu'IDT puisse exécuter le test. Vous devez mettre à jour la`device.json`modèle situé dans le`<device-tester-extract-location>/configs`dossier contenant les informations suivantes.

```
[
  {
    "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>"
            }
          }
        }
      }
    ]
  }
]
```

Dans`devices`, fournissez les informations suivantes :

`id`

Un identificateur unique défini par l'utilisateur pour votre appareil.

`connectivity.ip`

Adresse IP de votre appareil.

`connectivity.port`

Facultatif. Numéro de port à utiliser pour les connexions SSH à votre appareil.

`connectivity.auth`

Informations d'authentification pour la connexion.

Cette propriété s'applique uniquement si `connectivity.protocol` est défini sur `ssh`.

`connectivity.auth.method`

Méthode d'authentification utilisée pour accéder à un appareil sur le protocole de connectivité donné.

Les valeurs prises en charge sont :

- `pki`
- `password`

`connectivity.auth.credentials`

Informations d'identification utilisées pour l'authentification.

`connectivity.auth.credentials.user`

Le nom d'utilisateur utilisé pour se connecter à votre appareil.

`connectivity.auth.credentials.privKeyPath`

Chemin complet à la clé privée utilisée pour se connecter à votre appareil.

Cette valeur s'applique uniquement si `connectivity.auth.method` est défini sur `pki`.

`devices.connectivity.auth.credentials.password`

Mot de passe utilisé pour se connecter à votre appareil.

Cette valeur s'applique uniquement si `connectivity.auth.method` est défini sur `password`.

> **Note**
>
> Spécifiez `privKeyPath` uniquement si `method` est défini sur `pki`.
> Spécifiez `password` uniquement si `method` est défini sur `password`.

Exécutez la suite de tests

Une fois que vous avez créé votre suite de tests, vous devez vous assurer qu'elle fonctionne comme prévu. Effectuez les étapes suivantes pour exécuter la suite de tests avec votre pool de périphériques existant.

1. Copiez votre`MyTestSuite_1.0.0`folder in`<device-tester-extract-location>`/`tests`.
2. Exécutez les commandes suivantes :

```
cd <device-tester-extract-location>/bin
./devicetester_[linux | mac | win_x86-64] run-suite --suite-id MyTestSuite
```

IDT exécute votre suite de tests et diffuse les résultats vers la console. Lorsque le test est terminé, les informations suivantes s'affichent :

```
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
```

Dépannage

Utilisez les informations suivantes pour résoudre les problèmes liés à la réalisation du didacticiel.

Le cas de test ne s'exécute pas correctement

Si le test ne s'exécute pas correctement, IDT diffuse les journaux d'erreurs vers la console, ce qui peut vous aider à résoudre le problème de l'exécution du test. Avant de vérifier les journaux d'erreurs, vérifiez les points suivants :

- Le kit SDK client IDT se trouve dans le dossier approprié, comme décrit dansCette étape.
- Vous rencontrez tous lesexigencespour ce didacticiel.

Impossible de se connecter à l'appareil testé

Vérifiez les paramètres suivants :

- Votre`device.json`contient l'adresse IP, le port et les informations d'authentification correctes.
- Vous pouvez vous connecter à votre appareil via SSH à partir de votre ordinateur hôte.

Création de fichiers de configuration de suite de tests IDT

Cette section décrit les formats dans lesquels vous créez des fichiers de configuration JSON que vous incluez lorsque vous écrivez une suite de tests personnalisée.

Fichiers JSON requis

`suite.json`

Contient des informations à propos de la suite de tests. Consultez Configurer suite.json.

`group.json`

Contient des informations à propos d'un groupe de tests. Vous devez créer un outil`group.json`pour chaque groupe de tests de votre suite de tests. Consultez Configurer group.json.

`test.json`

Contient des informations à propos d'un scénario de test. Vous devez créer un outil`test.json`pour chaque cas de test de votre suite de tests. Consultez Configurer test.json.

Fichiers JSON en option

`state_machine.json`

Définit comment les tests sont exécutés lorsque IDT exécute la suite de tests. Consultez Configurer state_machine.json.

`userdata_schema.json`

Définit le schéma pour le`userdata.json`fichierque les coureurs de test peuvent inclure dans leur configuration de réglage. Le`userdata.json`est utilisé pour toutes les informations de configuration supplémentaires nécessaires à l'exécution du test, mais qui n'est pas présent dans le`device.json`dans le fichier. Consultez Configurer userdata_schema.json.

Les fichiers de configuration JSON sont placés dans votre*<custom-test-suite-folder>*comme illustré ici.

```
<custom-test-suite-folder>
### suite
    ### suite.json
    ### state_machine.json
    ### userdata_schema.json
    ### <test-group-folder>
        ### group.json
        ### <test-case-folder>
            ### test.json
```

Configurer suite.json

Le`suite.json`file définit les variables d'environnement et détermine si les données utilisateur sont requises pour exécuter la suite de tests. Utilisez le modèle suivant pour configurer votre outil*<custom-test-suite-folder>*`/suite/suite.json`dans le fichier:

```
{
    "id": "<suite-name>_<suite-version>",
    "title": "<suite-title>",
```

```
    "details": "<suite-details>",
    "userDataRequired": true | false,
    "environmentVariables": [
        {
            "key": "<name>",
            "value": "<value>",
        },
        ...
        {
            "key": "<name>",
            "value": "<value>",
        }
    ]
}
```

Tous les champs qui contiennent des valeurs sont requis, comme indiqué ici :

`id`

ID unique défini par l'utilisateur pour la suite de tests. Pour`id`doit correspondre au nom du dossier de suite de tests dans lequel le dossier de la suite de tests`suite.json`se trouve dans le fichier. Le nom de la suite et la version de suite doivent également respecter les critères suivants :

- *<suite-name>*ne peut contenir de traits de soulignement.
- *<suite-version>*est indiqué comme suit :*x.x.x*, où`x`est un nombre.

L'ID est affiché dans les rapports de test générés par IDT.

`title`

Nom défini par l'utilisateur pour le produit ou la fonctionnalité testé par cette suite de tests. Le nom est affiché dans l'interface de ligne de commande IDT pour les coureurs de test.

`details`

Description succincte de l'objectif de la suite de tests.

`userDataRequired`

Définit si les exécuteurs de tests doivent inclure des informations personnalisées dans un`userdata.json`dans le fichier. Si vous définissez cette valeur sur`true`, vous devez également inclure le`userdata_schema.json`fichierdans le dossier de votre suite de tests.

`environmentVariables`

Facultatif. Tableau des variables d'environnement à définir pour cette suite de tests.

`environmentVariables.key`

Nom de la variable d'environnement.

`environmentVariables.value`

Valeur de la variable d'environnement.

Configurer group.json

Le`group.json`définit si un groupe de tests est obligatoire ou facultatif. Utilisez le modèle suivant pour configurer votre outil*<custom-test-suite-folder>*/suite/*<test-group>*/`group.json`dans le fichier:

```
{
    "id": "<group-id>",
    "title": "<group-title>",
    "details": "<group-details>",
    "optional": true | false,
}
```

Tous les champs qui contiennent des valeurs sont requis, comme indiqué ici :

`id`

Identifiant unique défini par l'utilisateur pour le groupe de test. Pour`id`doit correspondre au nom du dossier de groupe de tests dans lequel le dossier de groupe de tests`group.json`est situé et ne doit pas comporter de traits de soulignement (_). L'ID est utilisé dans les rapports de test générés par IDT.

`title`

Nom descriptif du groupe de test. Le nom est affiché dans l'interface de ligne de commande IDT pour les coureurs de test.

`details`

Description succincte de l'objectif du groupe de tests.

`optional`

Facultatif. Définissez sur`true`pour afficher ce groupe de tests en tant que groupe facultatif une fois qu'IDT a terminé l'exécution des tests requis. La valeur par défaut est `false`.

Configurer test.json

Le`test.json`détermine les exécutables de cas de test et les variables d'environnement utilisées par un cas de test. Pour plus d'informations sur la création d'exécutables de scénario de test, consultezCréer des exécutables de cas de test IDT.

Utilisez le modèle suivant pour configurer votre outil`<custom-test-suite-folder>/suite/<test-group>/<test-case>/test.json`dans le fichier:

```
{
    "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>",
        }
    ]
}
```

Tous les champs qui contiennent des valeurs sont requis, comme indiqué ici :

`id`

Un identifiant unique défini par l'utilisateur pour le cas de test. Pour`id`doit correspondre au nom du dossier de scénario de test dans lequel le dossier de test`test.json`est situé et ne doit pas comporter de traits de soulignement (_). L'ID est utilisé dans les rapports de test générés par IDT.

`title`

Nom descriptif pour le scénario de test. Le nom est affiché dans l'interface de ligne de commande IDT pour les coureurs de test.

`details`

Description succincte de l'objectif du scénario de test.

`requireDUT`

Facultatif. Définissez sur`true`si un appareil est requis pour exécuter ce test, sinon réglé sur`false`. La valeur par défaut est `true`. Les coureurs de test vont configurer les appareils qu'ils utiliseront pour exécuter le test dans leur`device.json`dans le fichier.

`requiredResources`

Facultatif. Une baie qui fournit des informations sur les périphériques de ressources nécessaires à l'exécution de ce test.

`requiredResources.name`

Nom unique à attribuer au périphérique de ressources lorsque ce test est en cours d'exécution.

`requiredResources.features`

Un ensemble de fonctionnalités de périphériques de ressources définies par l'utilisateur.

`requiredResources.features.name`

Nom de la fonction. Fonctionnalité de l'appareil pour laquelle vous souhaitez utiliser cet appareil. Ce nom est comparé au nom de la fonction fourni par le lanceur de test dans le`resource.json`dans le fichier.

`requiredResources.features.version`

Facultatif. La version de la fonctionnalité. Cette valeur est comparée à la version de fonctionnalité fournie par le lanceur de test dans le`resource.json`dans le fichier. Si aucune version n'est fournie, la fonctionnalité n'est pas cochée. Si aucun numéro de version n'est requis pour l'entité, laissez ce champ vide.

`requiredResources.features.jobSlots`

Facultatif. Nombre de tests simultanés que cette fonctionnalité peut prendre en charge. La valeur par défaut est 1. Si vous souhaitez qu'IDT utilise des appareils distincts pour des fonctions individuelles, nous vous recommandons de définir cette valeur sur1.

`execution.timeout`

Temps (en millisecondes) pendant lequel IDT attend la fin de l'exécution du test. Pour plus d'informations sur la définition de cette valeur, consultezCréer des exécutables de cas de test IDT.

`execution.`*`os`*

Les exécutables de cas de test à exécuter en fonction du système d'exploitation de l'ordinateur hôte qui exécute IDT. Les valeurs prises en charge sont `linux`, `mac` et `win`.

`execution.`*`os`*`.cmd`

Chemin d'accès à l'exécutable du cas de test que vous souhaitez exécuter pour le système d'exploitation spécifié. Cet emplacement doit se trouver dans le chemin système.

`execution.`*`os`*`.args`

Facultatif. Les arguments à fournir pour exécuter l'exécutable du cas de test.

`environmentVariables`

Facultatif. Tableau des variables d'environnement défini pour ce scénario de test.

`environmentVariables.key`

Nom de la variable d'environnement.

`environmentVariables.value`

Valeur de la variable d'environnement.

> **Note**
>
> Si vous spécifiez la même variable d'environnement dans le`test.json`et dans le fichier`suite.json`, la valeur du fichier`test.json`est prioritaire.

Configurer state_machine.json

Une machine à états est une construction qui contrôle le flux d'exécution de la suite de tests. Il détermine l'état de départ d'une suite de tests, gère les transitions d'état en fonction de règles définies par l'utilisateur et continue de passer par ces états jusqu'à ce qu'il atteigne l'état final.

Si votre suite de tests n'inclut pas de machine à états définie par l'utilisateur, IDT générera une machine d'état pour vous. La machine d'état par défaut remplit les fonctions suivantes :

- Offre aux coureurs de tests la possibilité de sélectionner et d'exécuter des groupes de tests spécifiques, au lieu de toute la suite de tests.
- Si des groupes de tests spécifiques ne sont pas sélectionnés, exécute tous les groupes de tests de la suite de tests dans un ordre aléatoire.
- Génère des rapports et imprime un résumé de la console qui affiche les résultats des tests pour chaque groupe de test et chaque cas de test.

Pour plus d'informations sur le fonctionnement de la machine d'état IDT, consultezConfigurer la machine d'état IDT.

Configurer userdata_schema.json

Le`userdata_schema.json`détermine le schéma dans lequel les exécuteurs de test fournissent des données utilisateur. Les données utilisateur sont requises si votre suite de tests nécessite des informations qui ne sont pas présentes dans le`device.json`dans le fichier. Par exemple, vos tests

peuvent nécessiter des informations d'identification réseau Wi-Fi, des ports ouverts spécifiques ou des certificats qu'un utilisateur doit fournir. Ces informations peuvent être fournies à IDT sous la forme d'un paramètre d'entrée appeléuserdata, la valeur pour laquelle est unuserdata.json, que les utilisateurs créent dans leur*<device-tester-extract-location>*/configfolder. Le format de l'outiluserdata.jsonest basé sur le fichieruserdata_schema.jsonfichier que vous incluez dans la suite de tests.

Pour indiquer que les coureurs d'essai doivent fournir unuserdata.jsondans le fichier:

1. Danssuite.jsonFichier, ensembleuserDataRequiredpourtrue.
2. Dans vos produits*<custom-test-suite-folder>*, créez unuserdata_schema.jsondans le fichier.
3. Modification du contenuuserdata_schema.jsonfichier pour créer un fichier valideSchéma JSON version préliminaire de l'IETF v4.

Lorsque IDT exécute votre suite de tests, il lit automatiquement le schéma et l'utilise pour valider lauserdata.jsonfichier fourni par le lanceur de test. S'il est valide, le contenu duuserdata.jsonsont disponibles dans les deuxContexte IDTet dans leContexte de machine d'état.

Configurer la machine d'état IDT

Une machine à états est une construction qui contrôle le flux d'exécution de la suite de tests. Il détermine l'état de départ d'une suite de tests, gère les transitions d'état en fonction de règles définies par l'utilisateur et continue de passer par ces états jusqu'à ce qu'il atteigne l'état final.

Si votre suite de tests n'inclut pas de machine à états définie par l'utilisateur, IDT générera une machine d'état pour vous. La machine d'état par défaut remplit les fonctions suivantes :

- Offre aux coureurs de tests la possibilité de sélectionner et d'exécuter des groupes de tests spécifiques, au lieu de toute la suite de tests.
- Si des groupes de tests spécifiques ne sont pas sélectionnés, exécute tous les groupes de tests de la suite de tests dans un ordre aléatoire.
- Génère des rapports et imprime un résumé de la console qui affiche les résultats des tests pour chaque groupe de test et chaque cas de test.

La machine d'état d'une suite de tests IDT doit répondre aux critères suivants :

- Chaque état correspond à une action que IDT doit entreprendre, par exemple pour exécuter un groupe de test ou un produit un fichier de rapport.
- La transition vers un état exécute l'action associée à cet état.
- Chaque état définit la règle de transition pour l'état suivant.
- L'état final doit être soit`Succeed`ou`Fail`.

Format de la machine d'état

Vous pouvez utiliser le modèle suivant pour configurer le vôtre*<custom-test-suite-folder>*/`suite/state_machine.json`dans le fichier:

```
{
  "Comment": "<description>",
  "StartAt": "<state-name>",
  "States": {
    "<state-name>": {
      "Type": "<state-type>",
      // Additional state configuration
    }

    // Required states
    "Succeed": {
      "Type": "Succeed"
    },
    "Fail": {
      "Type": "Fail"
    }
  }
}
```

Tous les champs qui contiennent des valeurs sont requis, comme indiqué ici :

`Comment`

Description de la machine d'état.

`StartAt`

Nom de l'état auquel IDT commence à exécuter la suite de tests. Pour`StartAt`doit être défini sur l'un des états répertoriés dans la`States`objet.

`States`

Objet qui mappe les noms d'états définis par l'utilisateur à des états IDT valides. Chaque État.*`nom-état`*contient la définition d'un état valide mappé à la*`nom-état`*.

Le`States`l'objet doit inclure le`Succeed`et`Fail`états. Pour plus d'informations sur les états valides, consultezDéfinitions d'états et d'états valides.

Définitions d'états et d'états valides

Cette section décrit les définitions d'état de tous les états valides pouvant être utilisés dans la machine d'état IDT. Certains des états suivants prennent en charge les configurations au niveau du cas de test. Toutefois, nous vous recommandons de configurer des règles de transition d'état au niveau du groupe de test plutôt qu'au niveau du scénario de test, sauf si cela est absolument nécessaire.

Définitions des états

- RunTask
- Choice
- Parallèle
- Ajouter des fonctionnalités du produit
- Rapport
- Message de journal
- Sélectionner un groupe
- Fail
- Succeed

RunTask

Le`RunTask`state exécute les cas de tests d'un groupe de tests défini dans la suite de tests.

```
{
    "Type": "RunTask",
    "Next": "<state-name>",
    "TestGroup": "<group-id>",
    "TestCases": [
```

```
        "<test-id>"
    ],
    "ResultVar": "<result-name>"
}
```

Tous les champs qui contiennent des valeurs sont requis, comme indiqué ici :

`Next`

Nom de l'état auquel passer après avoir exécuté les actions de l'état actuel.

`TestGroup`

Facultatif. ID du groupe de test à exécuter. Si cette valeur n'est pas spécifiée, IDT exécute le groupe de test sélectionné par le lanceur de test.

`TestCases`

Facultatif. Tableau d'ID de cas de test du groupe spécifié dans`TestGroup`. Basé sur les valeurs de`TestGroup`et`TestCases`, IDT détermine le comportement d'exécution du test comme suit :

- Quand les deux`TestGroup`et`TestCases`sont spécifiés, IDT exécute les cas de test spécifiés à partir du groupe de tests.
- Quand`TestCases`sont spécifiés mais`TestGroup`n'est pas spécifié, IDT exécute les cas de test spécifiés.
- Quand`TestGroup`est spécifié, mais`TestCases`n'est pas spécifié, IDT exécute tous les cas de tests du groupe de tests spécifié.
- Quand aucun des deux`TestGroup`ou`TestCases`est spécifié, IDT exécute tous les cas de test du groupe de test sélectionné par le lanceur de test dans l'interface de ligne de commande IDT. Pour activer la sélection de groupes pour les coureurs d'essais, vous devez inclure les deux`RunTask`et`Choice`états dans votre`statemachine.json`dans le fichier. Pour obtenir un exemple montrant la façon dont cela fonctionne, consultezExemple de machine d'état : Exécuter des groupes de test sélectionnés par l'utilisateur.

  Pour plus d'informations sur l'activation des commandes IDT CLI pour les exécuteurs de tests, consultezthe section called “Activer les commandes IDT CLI”.

`ResultVar`

Nom de la variable de contexte à définir avec les résultats de l'exécution du test. Ne spécifiez pas cette valeur si vous n'avez pas spécifié de valeur pour`TestGroup`. IDT définit la valeur de la variable que vous définissez dans`ResultVar`pour`true`ou`false`basé sur les éléments suivants :

- Si le nom de la variable est du formulaire*text*_*text*_passed, la valeur est alors définie sur si tous les tests du premier groupe de test ont réussi ou ont été ignorés.
- Dans tous les autres cas, la valeur est définie sur si tous les tests de tous les groupes de tests ont réussi ou ont été ignorés.

En général, vous utiliserezRunTaskpour spécifier un ID de groupe de test sans spécifier d'ID de cas de test individuels, de sorte qu'IDT exécute tous les cas de test dans le groupe de test spécifié. Tous les cas de test exécutés par cet état sont exécutés en parallel, dans un ordre aléatoire. Toutefois, si tous les cas de test nécessitent l'exécution d'un périphérique et qu'un seul appareil est disponible, les cas de test seront exécutés de manière séquentielle.

Gestion des erreurs

Si l'un des groupes de test ou des identifiants de cas de test spécifiés n'est pas valide, cet état émet la valeurRunTaskErrorerreur d'exécution. Si l'état rencontre une erreur d'exécution, il définit également le paramètrehasExecutionErrorvariable dans le contexte de la machine à étatstrue.

Choice

LeChoicevous permet de définir dynamiquement l'état suivant vers lequel passer la transition en fonction des conditions définies par l'utilisateur.

```
{
    "Type": "Choice",
    "Default": "<state-name>",
    "FallthroughOnError": true | false,
    "Choices": [
        {
            "Expression": "<expression>",
            "Next": "<state-name>"
        }
    ]
}
```

Tous les champs qui contiennent des valeurs sont requis, comme indiqué ici :

`Default`

État auquel passer si aucune des expressions définies dansChoicespeuvent être évalués pourtrue.

`FallthroughOnError`

Facultatif. Spécifie le comportement lorsque l'état rencontre une erreur lors de l'évaluation des expressions. Définit sur`true`si vous souhaitez ignorer une expression si l'évaluation entraîne une erreur. Si aucune expression ne correspond, la machine d'état passe à la`Default`État. Si l'icône`FallthroughOnError`La valeur n'est pas spécifiée, elle est par défaut`false`.

`Choices`

Tableau d'expressions et d'états permettant de déterminer l'état vers lequel effectuer la transition après avoir exécuté les actions dans l'état actuel.

`Choices.Expression`

Chaîne d'expression qui correspond à une valeur booléenne. Si l'expression est évaluée à`true`, puis la machine d'état passe à l'état défini dans`Choices.Next`. Les chaînes d'expression récupèrent les valeurs du contexte de la machine d'état, puis effectuent des opérations sur elles pour obtenir une valeur booléenne. Pour plus d'informations sur l'accès au contexte de la machine d'état, voirContexte des machines d'état.

`Choices.Next`

Nom de l'état auquel passer si l'expression définie dans`Choices.Expression`évalue à`true`.

Gestion des erreurs

Le`Choice`l'état peut nécessiter la gestion des erreurs dans les cas suivants :

- Certaines variables des expressions de choix n'existent pas dans le contexte de la machine d'état.
- Le résultat d'une expression n'est pas une valeur booléenne.
- Le résultat d'une recherche JSON n'est pas une chaîne, un nombre ou un booléen.

Vous ne pouvez pas utiliser un`Catch`pour gérer les erreurs de cet état. Si vous souhaitez arrêter l'exécution de la machine d'état en cas d'erreur, vous devez définir`FallthroughOnError`pour`false`. Toutefois, nous vous recommandons de définir`FallthroughOnError`pour`true`, et selon votre cas d'utilisation, effectuez l'une des actions suivantes :

- Si une variable à laquelle vous accédez ne devrait pas exister dans certains cas, utilisez la valeur de`Default`et supplémentaires`Choices`blocs permettant de spécifier l'état suivant.

- Si une variable à laquelle vous accédez doit toujours exister, définissez la valeur`Default`État à`Fail`.

Parallèle

Le`Parallel`vous permet de définir et d'exécuter de nouvelles machines à états parallel les unes aux autres.

```
{
    "Type": "Parallel",
    "Next": "<state-name>",
    "Branches": [
        <state-machine-definition>
    ]
}
```

Tous les champs qui contiennent des valeurs sont requis, comme indiqué ici :

`Next`

Nom de l'état auquel passer après avoir exécuté les actions de l'état actuel.

`Branches`

Tableau de définitions de machines à états à exécuter. Chaque définition de machine d'état doit contenir la sienne`StartAt`,`Succeed`, et`Fail`états. Les définitions de machines d'état de cette baie ne peuvent pas faire référence à des états en dehors de leur propre définition.

> **Note**
>
> Étant donné que chaque machine d'état de branche partage le même contexte de machine d'état, la définition de variables dans une branche, puis la lecture de ces variables à partir d'une autre branche peut entraîner un comportement inattendu.

Le`Parallel`state passe à l'état suivant uniquement après avoir exécuté toutes les machines d'état de branche. Chaque état nécessitant un appareil attend d'être exécuté jusqu'à ce que l'appareil soit disponible. Si plusieurs appareils sont disponibles, cet état exécute des scénarios de test à partir de plusieurs groupes en parallel. Si suffisamment de périphériques ne sont pas disponibles, les cas de test seront exécutés de manière séquentielle. Étant donné que les cas de test sont exécutés dans un

ordre aléatoire lorsqu'ils sont exécutés en parallel, différents périphériques peuvent être utilisés pour exécuter des tests à partir du même groupe de tests.

Gestion des erreurs

Assurez-vous que la machine d'état de branche et la machine d'état parent transitent vers le`Fail`pour gérer les erreurs d'exécution.

Étant donné que les machines d'état de branche ne transmettent pas d'erreurs d'exécution à la machine d'état parente, vous ne pouvez pas utiliser un`Catch`pour gérer les erreurs d'exécution dans les machines d'état de branche. Utilisez plutôt le`hasExecutionErrors`valeur dans le contexte de la machine à états partagés. Pour obtenir un exemple montrant la façon dont cela fonctionne, consultezExemple de machine d'état : Exécutez deux groupes de tests en parallel.

Ajouter des fonctionnalités du produit

Le`AddProductFeatures`vous permet d'ajouter des fonctionnalités du produit à la`awsiotdevicetester_report.xml`fichier généré par IDT.

Une fonctionnalité produit est constituée d'informations définies par l'utilisateur sur des critères spécifiques auxquels un appareil peut répondre. Par exemple, les recettes`MQTT`peut indiquer que l'appareil publie correctement les messages MQTT. Dans le rapport, les fonctionnalités du produit sont définies comme suit :`supported`,`not-supported`, ou une valeur personnalisée, selon que les tests spécifiés ont réussi ou non.

> **Note**
>
> Le`AddProductFeatures`state ne génère pas de rapports par lui-même. Cet état doit passer à la`Report`étatpour générer des rapports.

```
{
    "Type": "Parallel",
    "Next": "<state-name>",
    "Features": [
        {
            "Feature": "<feature-name>",
            "Groups": [
                "<group-id>"
```

```
            ],
            "OneOfGroups": [
                "<group-id>"
            ],
            "TestCases": [
                "<test-id>"
            ],
            "IsRequired": true | false,
            "ExecutionMethods": [
                "<execution-method>"
            ]
        }
    ]
}
```

Tous les champs qui contiennent des valeurs sont requis, comme indiqué ici :

`Next`

Nom de l'état auquel passer après avoir exécuté les actions de l'état actuel.

`Features`

Un éventail de fonctionnalités du produit à afficher dans le`awsiotdevicetester_report.xml`dans le fichier.

`Feature`

Nom de la fonction

`FeatureValue`

Facultatif. La valeur personnalisée à utiliser dans le rapport au lieu de`supported`. Si cette valeur n'est pas spécifiée, en fonction des résultats des tests, la valeur de l'entité est définie sur`supported`ou`not-supported`.

Si vous utilisez une valeur personnalisée pour`FeatureValue`, vous pouvez tester la même fonction avec des conditions différentes, et IDT concatène les valeurs des entités pour les conditions prises en charge. Par exemple, l'extrait suivant montre la`MyFeature`avec deux valeurs d'entités distinctes :

```
...
{
    "Feature": "MyFeature",
```

```
    "FeatureValue": "first-feature-supported",
    "Groups": ["first-feature-group"]
},
{
    "Feature": "MyFeature",
    "FeatureValue": "second-feature-supported",
    "Groups": ["second-feature-group"]
},
...
```

Si les deux groupes de test réussissent, la valeur de l'entité est définie sur`first-feature-supported, second-feature-supported`.

`Groups`

Facultatif. Tableau des ID de groupe de test. Tous les tests au sein de chaque groupe de test spécifié doivent réussir pour que la fonctionnalité soit prise en charge.

`OneOfGroups`

Facultatif. Tableau des ID de groupe de test. Tous les tests au sein d'au moins un des groupes de tests spécifiés doivent réussir pour que la fonctionnalité soit prise en charge.

`TestCases`

Facultatif. Tableau des ID de cas de test. Si vous spécifiez cette valeur, les éléments suivants s'appliquent :

- Tous les cas de test spécifiés doivent réussir pour que la fonctionnalité soit prise en charge.
- `Groups`doit contenir un seul ID de groupe de tests.
- `OneOfGroups`ne doit pas être spécifié.

`IsRequired`

Facultatif. Définit sur`false`pour marquer cette fonctionnalité comme une fonctionnalité facultative dans le rapport. La valeur par défaut est `true`.

`ExecutionMethods`

Facultatif. Un tableau de méthodes d'exécution qui correspondent à la`protocol`valeur spécifiée dans le`device.json`dans le fichier. Si cette valeur est spécifiée, les exécuteurs de test doivent spécifier un`protocol`valeur qui correspond à l'une des valeurs de ce tableau pour inclure l'entité dans le rapport. Si cette valeur n'est pas spécifiée, l'entité sera toujours incluse dans le rapport.

Pour utiliser le pluginAddProductFeatures, vous devez définir la valeur deResultVardans leRunTaskindique l'une des valeurs suivantes :

- Si vous avez spécifié des ID de cas de test individuels, définissezResultVarpour*group-id_test-id*_passed.
- Si vous n'avez pas spécifié d'ID de cas de test individuels, définissezResultVarpour*group-id*_passed.

LeAddProductFeaturesvérifiez l'état des résultats des tests de la façon suivante :

- Si vous n'avez pas spécifié d'ID de cas de test, le résultat de chaque groupe de test est déterminé à partir de la valeur de la*group-id*_passedvariable dans le contexte de la machine d'état.
- Si vous avez spécifié des identifiants de cas de test, le résultat de chacun des tests est déterminé à partir de la valeur de la*group-id_test-id*_passedvariable dans le contexte de la machine d'état.

Gestion des erreurs

Si un ID de groupe fourni dans cet état n'est pas un ID de groupe valide, cet état génère leAddProductFeaturesErrorerreur d'exécution. Si l'état rencontre une erreur d'exécution, il définit également le paramètrehasExecutionErrorsvariable dans le contexte de la machine à étatstrue.

Rapport

LeReportgénère le*suite-name*_Report.xmletawsiotdevicetester_report.xmlfichiers suivants. Cet état diffuse également le rapport vers la console.

```
{
    "Type": "Report",
    "Next": "<state-name>"
}
```

Tous les champs qui contiennent des valeurs sont requis, comme indiqué ici :

`Next`

Nom de l'état auquel passer après avoir exécuté les actions de l'état actuel.

Vous devez toujours passer au`Report`état vers la fin du flux d'exécution du test afin que les coureurs de test puissent afficher les résultats des tests. Généralement, l'état suivant après cet état est`Succeed`.

Gestion des erreurs

Si cet état rencontre des problèmes lors de la génération des rapports, il émet alors le`ReportError`erreur d'exécution.

Message de journal

Le`LogMessage`génère le`test_manager.log`et diffuse le message de journal dans la console.

```
{
    "Type": "LogMessage",
    "Next": "<state-name>"
    "Level": "info | warn | error"
    "Message": "<message>"
}
```

Tous les champs qui contiennent des valeurs sont requis, comme indiqué ici :

`Next`

Nom de l'état auquel passer après avoir exécuté les actions de l'état actuel.

`Level`

Niveau d'erreur auquel créer le message de journal. Si vous spécifiez un niveau non valide, cet état génère un message d'erreur et le rejette.

`Message`

Message à enregistrer.

Sélectionner un groupe

Le`SelectGroup`state met à jour le contexte de la machine d'état pour indiquer quels groupes sont sélectionnés. Les valeurs définies par cet état sont utilisées par les autres`Choice`états.

```
{
    "Type": "SelectGroup",
```

```
    "Next": "<state-name>"
    "TestGroups": [
        <group-id>"
    ]
}
```

Tous les champs qui contiennent des valeurs sont requis, comme indiqué ici :

`Next`

Nom de l'état auquel passer après avoir exécuté les actions de l'état actuel.

`TestGroups`

Tableau de groupes de tests qui seront marqués comme sélectionnés. Pour chaque ID de groupe de test de cette baie, le*group-id*`_selected`est définie sur`true`dans le contexte. Assurez-vous de fournir des ID de groupe de test valides, car IDT ne vérifie pas si les groupes spécifiés existent.

Fail

Le`Fail`indique que la machine d'état ne s'est pas exécutée correctement. Il s'agit d'un état final pour la machine à états, et chaque définition de machine d'état doit inclure cet état.

```
{
    "Type": "Fail"
}
```

Succeed

Le`Succeed`indique que la machine d'état a été correctement exécutée. Il s'agit d'un état final pour la machine à états, et chaque définition de machine d'état doit inclure cet état.

```
{
    "Type": "Succeed"
}
```

Contexte des machines d'état

Le contexte de la machine d'état est un document JSON en lecture seule qui contient des données disponibles pour la machine à états pendant l'exécution. Le contexte de la machine d'état est accessible uniquement à partir de la machine à états et contient des informations qui déterminent

le flux de test. Par exemple, vous pouvez utiliser les informations configurées par les exécuteurs de tests dans le`userdata.json`pour déterminer si un test spécifique est nécessaire à l'exécution.

Le contexte de la machine d'état utilise le format suivant :

```
{
    "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`

Informations sur le pool de périphériques sélectionné pour le test. Pour un pool de périphériques sélectionné, ces informations sont extraites de l'élément de tableau de pool de périphériques de niveau supérieur correspondant défini dans le`device.json`dans le fichier.

`userData`

Informations dans le`userdata.json`dans le fichier.

`config`

Informations épinglez le`config.json`dans le fichier.

`suiteFailed`

La valeur est définie sur`false`lorsque la machine d'état démarre. Si un groupe de test échoue dans un`RunTask`, cette valeur est alors définie sur`true`pendant la durée restante de l'exécution de la machine d'état.

`specificTestGroups`

Si le lanceur de tests sélectionne des groupes de tests spécifiques à exécuter au lieu de toute la suite de tests, cette clé est créée et contient la liste des ID de groupes de tests spécifiques.

`specificTestCases`

Si le lanceur de tests sélectionne des cas de test spécifiques à exécuter au lieu de toute la suite de tests, cette clé est créée et contient la liste des ID de cas de test spécifiques.

`hasExecutionErrors`

Ne se ferme pas lorsque la machine d'état démarre. Si un état rencontre des erreurs d'exécution, cette variable est créée et définie sur`true`pendant la durée restante de l'exécution de la machine d'état.

Vous pouvez interroger le contexte à l'aide de la notation JSONPath. La syntaxe des requêtes JSONPath dans les définitions d'état est`{{$.query}}`. Vous pouvez utiliser des requêtes JSONPath comme chaînes d'espace réservé dans certains états. IDT remplace les chaînes d'espace réservé par la valeur de la requête JSONPath évaluée à partir du contexte. Vous pouvez utiliser des espaces réservés pour les valeurs suivantes :

- `LeTestCases`valeur dans`RunTask`états.
- `LeExpression`valeur`Choice`État.

Lorsque vous accédez à des données du contexte de la machine d'état, vérifiez que les conditions suivantes sont remplies :

- Vos chemins JSON doivent commencer par`$.`
- Chaque valeur doit être évaluée en chaîne, en nombre ou en booléen.

Pour plus d'informations sur l'utilisation de la notation JSONPath pour accéder aux données du contexte, consultezUtiliser le contexte IDT.

Erreurs d'exécution

Les erreurs d'exécution sont des erreurs dans la définition de la machine d'état rencontrées par la machine d'état lors de l'exécution d'un état. IDT enregistre les informations relatives à chaque erreur dans le`test_manager.log`et diffuse le message de journal dans la console.

Vous pouvez utiliser les méthodes suivantes pour gérer les erreurs d'exécution :

- Ajout d'unCatchblocdans la définition de l'état.
- Vérifiez la valeur duhasExecutionErrorsvaleurdans le contexte de la machine d'état.

Capture

Pour utiliserCatch, ajoutez ce qui suit à votre définition d'état :

```
"Catch": [
    {
        "ErrorEquals": [
            "<error-type>"
        ]
        "Next": "<state-name>"
    }
]
```

Tous les champs qui contiennent des valeurs sont requis, comme indiqué ici :

`Catch.ErrorEquals`

Tableau des types d'erreurs à catch. Si une erreur d'exécution correspond à l'une des valeurs spécifiées, la machine d'état passe à l'état spécifié dansCatch.Next. Consultez chaque définition d'état pour plus d'informations sur le type d'erreur qu'elle produit.

`Catch.Next`

État suivant à passer si l'état actuel rencontre une erreur d'exécution correspondant à l'une des valeurs spécifiées dansCatch.ErrorEquals.

Les blocs de capture sont gérés de manière séquentielle jusqu'à ce qu'un bloc corresponde Si les erreurs sans erreur correspondent à celles répertoriées dans les blocs Catch, les machines d'état continuent de s'exécuter. Étant donné que les erreurs d'exécution sont le résultat de définitions d'état incorrectes, nous vous recommandons de passer à l'état Échec lorsqu'un état rencontre une erreur d'exécution.

Erreur d'exécution

Lorsque certains états rencontrent des erreurs d'exécution, en plus d'émettre l'erreur, ils définissent également le paramètrehasExecutionErrorvaleur pourtruedans le contexte de la machine

d'état. Vous pouvez utiliser cette valeur pour détecter quand une erreur se produit, puis utiliser un`Choice`pour faire passer la machine à états vers le`Fail`État.

Cette méthode possède les caractéristiques suivantes.

- La machine d'état ne démarre avec aucune valeur attribuée à`hasExecutionError`, et cette valeur n'est pas disponible tant qu'un état particulier ne l'a pas définie. Cela signifie que vous devez définir explicitement la`FallthroughOnError`pour`false`pour la`Choice`les états qui accèdent à cette valeur pour empêcher l'arrêt de la machine d'état si aucune erreur d'exécution ne se produit.
- Une fois qu'il est réglé sur`true`,`hasExecutionError`n'est jamais défini sur false ou supprimé du contexte. Cela signifie que cette valeur n'est utile que la première fois qu'elle est définie sur`true`, et pour tous les États subséquents, il ne fournit pas de valeur significative.
- Le`hasExecutionError`est partagée avec toutes les machines d'état de branche dans le`Parallel`, ce qui peut entraîner des résultats inattendus en fonction de l'ordre dans lequel il est accédé.

En raison de ces caractéristiques, nous ne recommandons pas d'utiliser cette méthode si vous pouvez utiliser un bloc Catch à la place.

Exemple de machines d'état

Cette section fournit quelques exemples de configurations de machines d'état.

Exemples

- Exemple de machine d'état : Exécutez un groupe de test unique
- Exemple de machine d'état : Exécuter des groupes de test sélectionnés par l'utilisateur
- Exemple de machine d'état : Exécutez un groupe de test unique avec des fonctionnalités du produit
- Exemple de machine d'état : Exécutez deux groupes de tests en parallel

Exemple de machine d'état : Exécutez un groupe de test unique

Cette machine d'état :

- Exécutez le groupe de test avec l'identifiant`GroupA`, qui doit être présente dans la suite d'un`group.json`dans le fichier.
- Vérifie les erreurs d'exécution et les transitions vers`Fail`s'il y en a.

- Génère un rapport et effectue des transitions vers`Succeeds`'il n'y a pas d'erreurs, et`Fail`autrement.

```
{
    "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"
        }
    }
}
```

Exemple de machine d'état : Exécuter des groupes de test sélectionnés par l'utilisateur

Cette machine d'état :

- Vérifie si le coureur de test a sélectionné des groupes de test spécifiques. La machine d'état ne vérifie pas les cas de test spécifiques car les exécuteurs de tests ne peuvent pas sélectionner de cas de test sans avoir également sélectionné un groupe de test.
- Si les groupes de test sont sélectionnés :
  - Exécute les cas de test au sein des groupes de test sélectionnés. Pour ce faire, la machine d'état ne spécifie explicitement aucun groupe de test ni aucun cas de test dans le`RunTask`État.
  - Génère un rapport après avoir exécuté tous les tests et les sorties.
- Si les groupes de test ne sont pas sélectionnés :
  - Exécute des tests dans un groupe de test`GroupA`.
  - Génère des rapports et des sorties.

```
{
    "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"
        }
    }
}
```

Exemple de machine d'état : Exécutez un groupe de test unique avec des fonctionnalités du produit

Cette machine d'état :

- Exécutez le groupe de test`GroupA`.

- Vérifie les erreurs d'exécution et les transitions vers`Fails`'il y en a.
- Ajoute le`FeatureThatDependsOnGroupA`fonction de la fonction`awsiotdevicetester_report.xml`dans le fichier:
  - Si`GroupA`passe, la fonction est définie sur`supported`.
  - La fonctionnalité n'est pas marquée comme facultative dans le rapport.
- Génère un rapport et effectue des transitions vers`Succeeds`'il n'y a pas d'erreurs, et`Fail`autrement

```
{
    "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"
        }
    }
}
```

Exemple de machine d'état : Exécutez deux groupes de tests en parallel

Cette machine d'état :

- Exécutez le`GroupA`et`GroupB`groupes de test en parallel. Le`ResultVar`variables stockées dans le contexte par le`RunTask`les états dans les machines d'état de branche sont disponibles pour le`AddProductFeatures`État.
- Vérifie les erreurs d'exécution et les transitions vers`Fails`'il y en a. Cette machine d'état n'utilise pas de`Catch`bloc car cette méthode ne détecte pas les erreurs d'exécution dans les machines d'état de branche.
- Ajoute des fonctionnalités au`awsiotdevicetester_report.xml`fichier basé sur les groupes qui passent
  - Si`GroupA`passe, la fonction est définie sur`supported`.
  - La fonctionnalité n'est pas marquée comme facultative dans le rapport.
- Génère un rapport et effectue des transitions vers`Succeeds`'il n'y a pas d'erreurs, et`Fail`autrement

Si deux appareils sont configurés dans le pool de périphériques, les deux`GroupA`et`GroupB`peut fonctionner en même temps. Toutefois, si l'un ou l'autre`GroupA`ou`GroupB`contient plusieurs tests,

puis les deux appareils peuvent être alloués à ces tests. Si un seul appareil est configuré, les groupes de test seront exécutés de manière séquentielle.

```
{
    "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"
        }
    }
}
```

Créer des exécutables de cas de test IDT

Vous pouvez créer et placer des exécutables de cas de test dans un dossier de suite de tests de la manière suivante :

- Pour les suites de tests qui utilisent des arguments ou des variables d'environnement de la`test.json`pour déterminer les tests à exécuter, vous pouvez créer un exécutable de cas de test unique pour l'ensemble de la suite de tests ou un exécutable de test pour chaque groupe de tests de la suite de tests.
- Pour une suite de tests dans laquelle vous souhaitez exécuter des tests spécifiques basés sur des commandes spécifiées, vous créez un exécutable de cas de test pour chaque cas de test de la suite de tests.

En tant que rédacteur de tests, vous pouvez déterminer quelle approche convient à votre cas d'utilisation et structurer l'exécutable de votre cas de test en conséquence. Assurez-vous que vous fournissez le chemin exécutable de cas de test correct dans chaque cas`test.json`et que l'exécutable spécifié s'exécute correctement.

Lorsque tous les appareils sont prêts pour l'exécution d'un scénario de test, IDT lit les fichiers suivants :

- L`etest.json`pour le cas de test sélectionné, détermine les processus à démarrer et les variables d'environnement à définir.
- L`esuite.json`pour la suite de tests détermine les variables d'environnement à définir.

IDT lance le processus exexutable de test requis en fonction des commandes et des arguments spécifiés dans l`etest.json`et transmet les variables d'environnement requises au processus.

Utiliser le SDK Client IDT

Les kits SDK Client IDT vous permettent de simplifier la façon dont vous écrivez une logique de test dans votre exécutable de test avec des commandes API que vous pouvez utiliser Interact avec IDT et vos appareils testés. IDT fournit actuellement les kits de développement logiciel suivants :

- Kit SDK client IDT pour Python
- Kit SDK client IDT pour Go

Ces kits SDK sont situés dans le`<device-tester-extract-location>/sdks`folder. Lorsque vous créez un nouvel exécutable de cas de test, vous devez copier le kit SDK que vous souhaitez utiliser dans le dossier contenant l'exécutable de votre cas de test et référencer le SDK dans votre code. Cette section fournit une brève description des commandes d'API disponibles que vous pouvez utiliser dans vos exécutables de scénario de test.

Dans cette section

- Interaction entre appareils
- Interaction IDT
- Interaction avec l'

Interaction entre appareils

Les commandes suivantes vous permettent de communiquer avec l'appareil testé sans avoir à implémenter d'autres fonctions d'interaction et de gestion de la connectivité des appareils.

`ExecuteOnDevice`

Permet aux suites de tests d'exécuter des commandes shell sur un périphérique prenant en charge les connexions SSH ou Docker Shell.

`CopyToDevice`

Permet aux suites de tests de copier un fichier local depuis la machine hôte qui exécute IDT vers un emplacement spécifié sur un périphérique prenant en charge les connexions SSH ou Docker Shell.

`ReadFromDevice`

Permet aux suites de tests de lire à partir du port série des appareils prenant en charge les connexions UART.

> **Note**
>
> Étant donné que IDT ne gère pas les connexions directes aux appareils créés à l'aide des informations d'accès aux appareils à partir du contexte, nous vous recommandons d'utiliser ces commandes API d'interaction avec les appareils dans vos exécutables de cas de test. Toutefois, si ces commandes ne répondent pas aux exigences de votre scénario de test, vous pouvez récupérer les informations d'accès à l'appareil à partir du contexte IDT et les utiliser pour établir une connexion directe à l'appareil à partir de la suite de tests.
> Pour établir une connexion directe, récupérez les informations dans le`device.connectivity`et l'`resource.devices.connectivity`pour votre appareil en cours de test et pour les périphériques de ressources, respectivement. Pour plus d'informations sur l'utilisation du contexte IDT, consultezUtiliser le contexte IDT.

Interaction IDT

Les commandes suivantes permettent à vos suites de tests de communiquer avec IDT.

`PollForNotifications`

Permet aux suites de tests de vérifier la présence de notifications provenant d'IDT.

`GetContextValue` et `GetContextString`

Permet aux suites de tests de récupérer des valeurs à partir du contexte IDT. Pour plus d'informations, consultez Utiliser le contexte IDT.

`SendResult`

Permet aux suites de tests de signaler les résultats des scénarios de test à IDT. Cette commande doit être appelée à la fin de chaque cas de test dans une suite de tests.

Interaction avec l'

La commande suivante permet à vos suites de tests de communiquer avec la machine hôte.

`PollForNotifications`

Permet aux suites de tests de vérifier la présence de notifications provenant d'IDT.

`GetContextValue` et `GetContextString`

Permet aux suites de tests de récupérer des valeurs à partir du contexte IDT. Pour plus d'informations, consultez Utiliser le contexte IDT.

`ExecuteOnHost`

Permet aux suites de tests d'exécuter des commandes sur la machine locale et permet à IDT de gérer le cycle de vie de l'exécutable du cas de test.

Activer les commandes IDT CLI

Le`run-suite`commande IDT CLI propose plusieurs options qui permettent au lanceur de test de personnaliser l'exécution des tests. Pour permettre aux exécuteurs de tests d'utiliser ces options pour exécuter votre suite de tests personnalisée, vous implémentez la prise en charge de l'interface de ligne de commande IDT. Si vous n'implémentez pas la prise en charge, les exécuteurs de tests pourront toujours exécuter des tests, mais certaines options CLI ne fonctionneront pas correctement. Pour offrir une expérience client idéale, nous vous recommandons de mettre en œuvre la prise en charge des arguments suivants pour le`run-suite`dans l'IDT CLI :

`timeout-multiplier`

Spécifie une valeur supérieure à 1,0 qui sera appliquée à tous les délais d'attente lors de l'exécution des tests.

Les exécuteurs de test peuvent utiliser cet argument pour augmenter le délai d'attente des cas de test qu'ils souhaitent exécuter. Lorsqu'un lanceur de test spécifie cet argument dans son`run-suite`, IDT l'utilise pour calculer la valeur de la variable d'environnement IDT_TEST_TIMEOUT et définit la valeur`config.timeoutMultiplier`dans le contexte IDT. Pour soutenir cet argument, vous devez procéder comme suit :

- Au lieu d'utiliser directement la valeur du délai d'attente de l`atest.json`, lisez la variable d'environnement IDT_TEST_TIMEOUT pour obtenir la valeur de délai d'expiration correctement calculée.
- Récupérer le`config.timeoutMultiplier`à partir du contexte IDT et l'appliquer à des délais d'attente longs.

Pour plus d'informations sur la sortie anticipée en raison d'événements de délai d'attente, consultezSpécifier le comportement de sortie.

`stop-on-first-failure`

Spécifie qu'IDT doit cesser d'exécuter tous les tests s'il rencontre un échec.

Lorsqu'un lanceur de test spécifie cet argument dans son`run-suite`, IDT cessera d'exécuter les tests dès qu'il rencontre un échec. Toutefois, si les cas de test sont exécutés en parallel, cela peut entraîner des résultats inattendus. Pour implémenter le support, assurez-vous que si IDT rencontre cet événement, votre logique de test demande à tous les cas de test en cours d'arrêt, de nettoyage des ressources temporaires et de signaler un résultat de test à IDT. Pour plus d'informations sur la sortie anticipée des échecs, consultezSpécifier le comportement de sortie.

`group-id` et `test-id`

Spécifie qu'IDT doit exécuter uniquement les groupes de test ou les cas de test sélectionnés.

Les coureurs de test peuvent utiliser ces arguments avec leur`run-suite`pour spécifier le comportement d'exécution de test suivant :

- Exécutez tous les tests au sein des groupes de tests spécifiés.
- Exécutez une sélection de tests à partir d'un groupe de tests spécifié.

Pour prendre en charge ces arguments, la machine d'état de votre suite de tests doit inclure un ensemble spécifique de`RunTasketChoice`dans votre machine d'état. Si vous n'utilisez pas

de machine à états personnalisée, la machine d'état IDT par défaut inclut les états requis pour vous et vous n'avez pas besoin de prendre d'autres mesures. Toutefois, si vous utilisez une machine à états personnalisée, utilisezExemple de machine d'état : Exécuter des groupes de test sélectionnés par l'utilisateurcomme exemple pour ajouter les états requis dans votre machine d'état.

Pour plus d'informations sur les commandes IDT CLI, consultezDéboguer et exécuter des suites de tests personnalisées.

rédigent des journaux d'événements

Pendant que le test est en cours, vous envoyez des données à`stdout`et`stderr`pour écrire des journaux d'événements et des messages d'erreur sur la console. Pour plus d'informations sur le format des messages de console, consultezFormat des messages de la console.

Lorsque l'IDT a terminé d'exécuter la suite de tests, ces informations sont également disponibles dans le`test_manager.log`situé dans le fichier*<devicetester-extract-location>*/`results/`*<execution-id>*`/logs`folder.

Vous pouvez configurer chaque cas de test pour écrire les journaux de son exécution de test, y compris les journaux de l'appareil testé, dans le*<group-id>*_*<test-id>*situé dans le fichier*<device-tester-extract-location>*`/results/`*execution-id*`/logs`folder. Pour ce faire, récupérez le chemin d'accès au fichier journal à partir du contexte IDT avec le`testData.logFilePath`, créez un fichier à ce chemin d'accès et écrivez le contenu que vous souhaitez y accéder. IDT met automatiquement à jour le chemin en fonction du scénario de test en cours d'exécution. Si vous choisissez de ne pas créer le fichier journal pour un cas de test, aucun fichier n'est généré pour ce cas de test.

Vous pouvez également configurer votre exécutable de texte pour créer des fichiers journaux supplémentaires au besoin dans le*<device-tester-extract-location>*`/logs`folder. Nous vous recommandons de spécifier des préfixes uniques pour les noms de fichiers journaux afin que vos fichiers ne soient pas écrasés.

Signaler les résultats à IDT

IDT écrit les résultats des tests sur le`awsiotdevicetester_report.xml`et l'*suite-name*`_report.xml`fichiers suivants. Ces fichiers de rapport se trouvent dans*<device-tester-extract-location>*`/results/`*<execution-id>*`/`. Les deux rapports capturent les résultats

de l'exécution de la suite de tests. Pour plus d'informations sur les schémas utilisés par IDT pour ces rapports, consultezExaminer les résultats des tests IDT et les journaux

Pour renseigner le contenu de l'*suite-name*_report.xml, vous devez utiliser leSendResultpour signaler les résultats des tests à IDT avant la fin de l'exécution du test. Si IDT ne peut pas localiser les résultats d'un test, il génère une erreur pour le cas de test. L'extrait Python suivant montre les commandes permettant d'envoyer un résultat de test à IDT :

```
request-variable = SendResultRequest(TestResult(result))
client.send_result(request-variable)
```

Si vous ne signalez pas de résultats via l'API, IDT recherche les résultats des tests dans le dossier des artefacts de test. Le chemin d'accès à ce dossier est stocké dans letestData.testArtifactsPathclassé dans le contexte IDT. Dans ce dossier, IDT utilise le premier fichier XML trié par ordre alphabétique qu'il trouve comme résultat du test.

Si votre logique de test produit des résultats XML JUnit, vous pouvez écrire les résultats du test dans un fichier XML dans le dossier des artefacts pour fournir directement les résultats à IDT au lieu d'analyser les résultats, puis d'utiliser l'API pour les soumettre à IDT.

Si vous utilisez cette méthode, assurez-vous que votre logique de test résume avec précision les résultats du test et formate votre fichier de résultats dans le même format que le*suite-name*_report.xmldans le fichier. IDT n'effectue aucune validation des données que vous fournissez, à l'exception des exceptions suivantes :

- IDT ignore toutes les propriétés dutestsuitesétiquette. Au lieu de cela, il calcule les propriétés des balises à partir d'autres résultats de groupes de tests signalés.
- Au moins unetestsuitela balise doit exister danstestsuites.

Étant donné que IDT utilise le même dossier d'artefacts pour tous les cas de test et ne supprime pas les fichiers de résultats entre les essais, cette méthode peut également entraîner des rapports erronés si IDT lit le fichier incorrect. Nous vous recommandons d'utiliser le même nom pour le fichier de résultats XML généré dans tous les cas de test afin d'écraser les résultats de chaque cas de test et de vous assurer que les résultats corrects sont disponibles pour IDT. Bien que vous puissiez utiliser une approche mixte pour générer des rapports dans votre suite de tests, c'est-à-dire utiliser un fichier de résultats XML pour certains cas de test et soumettre des résultats via l'API pour d'autres, nous ne recommandons pas cette approche.

Spécifier le comportement de sortie

Configurez votre exécutable de texte pour qu'il quitte toujours avec un code de sortie 0, même si un cas de test signale une défaillance ou un résultat d'erreur. Utilisez des codes de sortie non nuls uniquement pour indiquer qu'un cas de test n'a pas été exécuté ou si l'exécutable du cas de test n'a pas pu communiquer de résultats à IDT. Lorsque IDT reçoit un code de sortie différent de zéro, il marque que le cas de test a rencontré une erreur qui l'a empêché de s'exécuter.

IDT peut demander ou s'attendre à ce qu'un cas de test cesse de s'exécuter avant qu'il ne soit terminé dans les événements suivants. Utilisez ces informations pour configurer l'exécutable de votre cas de test afin de détecter chacun de ces événements à partir du scénario de test :

Expiration

Se produit lorsqu'un scénario de test s'exécute plus longtemps que la valeur du délai d'attente spécifiée dans le champ`test.json`dans le fichier. Si le coureur de test a utilisé le`timeout-multiplier`pour spécifier un multiplicateur de délai d'attente, puis IDT calcule la valeur du délai d'attente avec le multiplicateur.

Pour détecter cet événement, utilisez la variable d'environnement IDT_TEST_TIMEOUT. Lorsqu'un lanceur de tests lance un test, IDT définit la valeur de la variable d'environnement IDT_TEST_TIMEOUT sur la valeur de délai d'expiration calculée (en secondes) et transmet la variable à l'exécutable du scénario de test. Vous pouvez lire la valeur de la variable pour définir un minuteur approprié.

Interrupte

Se produit lorsque le coureur de test interrompt IDT. Par exemple, en appuyant surCtrl+C.

Étant donné que les terminaux propagent des signaux à tous les processus enfants, vous pouvez simplement configurer un gestionnaire de signaux dans vos cas de test pour détecter les signaux d'interruption.

Vous pouvez également interroger périodiquement l'API pour vérifier la valeur de l'API`CancellationRequested`booléen dans le`PollForNotifications`Réponse de l'API. Lorsque IDT reçoit un signal d'interruption, il définit la valeur du paramètre`CancellationRequested`booléen`true`.

Arrêtez lors du premier échec

Se produit lorsqu'un cas de test exécuté en parallel avec le cas de test actuel échoue et que le lanceur de test a utilisé le paramètre`stop-on-first-failure`pour spécifier qu'IDT doit s'arrêter lorsqu'il rencontre une défaillance.

Pour détecter cet événement, vous pouvez interroger périodiquement l'API afin de vérifier la valeur de l'API`CancellationRequested`booléen dans le`PollForNotifications`Réponse de l'API. Lorsque IDT rencontre une défaillance et est configuré pour s'arrêter lors de la première défaillance, il définit la valeur de la`CancellationRequested`booléen`true`.

Lorsque l'un de ces événements se produit, IDT attend pendant 5 minutes que tous les cas de test en cours d'exécution soient terminés. Si tous les cas de test en cours ne se terminent pas dans les 5 minutes, IDT force chacun de ses processus à s'arrêter. Si IDT n'a pas reçu de résultats de test avant la fin des processus, il marquera les cas de test comme ayant expiré. En tant que meilleure pratique, vous devez vous assurer que vos cas de test effectuent les actions suivantes lorsqu'ils rencontrent l'un des événements :

1. Arrêtez d'exécuter une logique de test normale.
2. Nettoyez toutes les ressources temporaires, telles que les artefacts de test sur l'appareil testé.
3. Signalez un résultat de test à IDT, tel qu'un échec du test ou une erreur.
4. Quitter.

Utiliser le contexte IDT

Lorsque IDT exécute une suite de tests, la suite de tests peut accéder à un ensemble de données qui peuvent être utilisées pour déterminer comment chaque test s'exécute. Ces données sont appelées contexte IDT. Par exemple, la configuration des données utilisateur fournie par les exécuteurs de tests dans un`userdata.json`est mis à la disposition des suites de tests dans le contexte IDT.

Le contexte IDT peut être considéré comme un document JSON en lecture seule. Les suites de tests peuvent extraire des données du contexte et les écrire à l'aide de types de données JSON standard tels que des objets, des tableaux, des nombres, etc.

Schéma contextuel

Le contexte IDT utilise le format suivant :

```
{
    "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`

Informations provenant du`config.json`fichier. Le`config`contient également le champ supplémentaire suivant :

`config.timeoutMultiplier`

Multiplicateur pour toute valeur de délai d'attente utilisée par la suite de tests. Cette valeur est spécifiée par le lanceur de test à partir de l'interface de ligne de commande IDT. La valeur par défaut est 1.

`device`

Informations sur le périphérique sélectionné pour le test. Ces informations sont équivalentes à l'information`devices`élément de tableau dans le`device.json`fichierpour l'appareil sélectionné.

`devicePool`

Informations sur le pool de périphériques sélectionné pour le test. Ces informations sont équivalentes à l'élément de tableau de pool de périphériques de niveau supérieur défini dans le`device.json`pour le pool de périphériques sélectionné.

`resource`

Informations sur les périphériques de ressources provenant du`resource.json`dans le fichier.

`resource.devices`

Ces informations sont équivalentes à l'information`devices`tableau défini dans le tableau`resource.json`dans le fichier. EACH`devices`inclut le champ supplémentaire suivant :

`resource.device.name`

Nom du périphérique de ressources. Cette valeur est définie sur le paramètre`requiredResource.name`valeur dans le module`test.json`dans le fichier.

`testData.awsCredentials`

LeAWSinformations d'identification utilisées par le test pour se connecter auAWScloud. Ces informations sont obtenues auprès du`config.json`dans le fichier.

`testData.logFilePath`

Chemin d'accès au fichier journal dans lequel le scénario de test écrit des messages de journal. S'il n'existe pas, la suite de tests crée ce fichier.

`userData`

Informations fournies par le coureur de test dans le`userdata.json`fichier.

Accédez aux données dans le contexte

Vous pouvez interroger le contexte à l'aide de la notation JSONPath à partir de vos fichiers JSON et de votre exécutable de texte avec le`GetContextValue`et`GetContextString`API. La syntaxe des chaînes JSONPath pour accéder au contexte IDT varie comme suit :

- Dans`suite.jsonettest.json`, vous utilisez`{{query}}`. En d'autres termes, n'utilisez pas l'élément racine`$.`pour commencer votre expression.
- Dans`statemachine.json`, vous utilisez`{{$.query}}`.
- Dans les commandes API, vous utilisez`query`ou`{{$.query}}`, selon la commande. Pour de plus amples informations, veuillez consulter la documentation en ligne dans les kits SDK.

Le tableau suivant décrit les opérateurs dans une expression JSONPath typique :

Operator	Description
`$`	The root element. Because the top-level context value for IDT is an object, you will typically use `$.` to start your queries.
`.nom d'enfant`	Accesses the child element with name `Nom de l'enfant` from an object. If applied to an array, yields a new array with this operator applied to each element. The element name is case sensitive. For example, the query to access the `awsRegion` value in the `config` object is `Région $.config.awsRegion` .
`[début : fin]`	Filters elements from an array, retrieving items beginning from the `démarrer` index and going up to the `fin` index, both inclusive.
`[index1, index2,..., IndexN]`	Filters elements from an array, retrieving items from only the specified indices.
`[? (expr)]`	Filters elements from an array using the `expr` expression. This expression must evaluate to a boolean value.

Pour créer des expressions de filtre, utilisez la syntaxe suivante :

```
<jsonpath> | <value> operator <jsonpath> | <value>
```

Dans cette syntaxe :

- `jsonpath`est un JSONPath qui utilise la syntaxe JSON standard.
- `value`est toute valeur personnalisée qui utilise la syntaxe JSON standard.
- `operator`est l'un des opérateurs suivants :
  - <(Inférieur à)
  - <=(Inférieur ou égal à)
  - ==(Egal à)

    Si le JSONPath ou la valeur de votre expression est une valeur de tableau, d'un booléen ou d'un objet, il s'agit du seul opérateur binaire pris en charge que vous pouvez utiliser.
  - >=(Supérieur ou égal à)
  - >(Supérieur à)
  - =~(correspondance d'expression régulière). Pour utiliser cet opérateur dans une expression de filtre, le JSONPath ou la valeur située sur le côté gauche de votre expression doit être évalué en chaîne et le côté droit doit être une valeur de modèle qui suit leSyntaxe RE2.

Vous pouvez utiliser des requêtes JSONPath sous la forme {{*`requête`*}} sous forme de chaînes d'espace réservé dans le`args`et`environmentVariables`champs dans`test.json`et dans le dossier`environmentVariables`champs dans`suite.json`fichiers suivants. IDT effectue une recherche de contexte et remplit les champs avec la valeur évaluée de la requête. Par exemple, dans le`suite.json`, vous pouvez utiliser des chaînes d'espace réservé pour spécifier des valeurs de variables d'environnement qui changent avec chaque cas de test et IDT renseignera les variables d'environnement avec la valeur correcte pour chaque cas de test. Toutefois, lorsque vous utilisez des chaînes d'espace réservé dans`test.json`et`suite.json`, les considérations suivantes s'appliquent à vos requêtes :

- Vous devez chaque occurrence du`devicePool`dans votre requête en minuscules. C'est-à-dire, utilisez`devicepool`.
- Pour les tableaux, vous ne pouvez utiliser que des tableaux de chaînes. De plus, les baies utilisent un système non standard`item1, item2,...,itemN`. Si le tableau ne contient qu'un seul élément, il est sérialisé comme suit :`item`, ce qui le rend indiscernable d'un champ de chaîne.
- Vous ne pouvez pas utiliser d'espaces réservés pour récupérer des objets du contexte.

En raison de ces considérations, nous recommandons que, dans la mesure du possible, vous utilisiez l'API pour accéder au contexte de votre logique de test plutôt que des chaînes d'espace réservé dans`test.json`et`suite.json`fichiers suivants. Cependant, dans certains cas, il peut être plus pratique d'utiliser des espaces réservés JSONPath pour récupérer des chaînes uniques à définir comme variables d'environnement.

Configuration des paramètres pour les coureurs de tests

Pour exécuter des suites de tests personnalisées, les exécuteurs de tests doivent configurer leurs paramètres en fonction de la suite de tests qu'ils souhaitent exécuter. Les paramètres sont spécifiés en fonction des modèles de fichiers de configuration JSON situés dans le*<device-tester-extract-location>*`/configs/`folder. Si nécessaire, les coureurs d'essai doivent également configurerAWSinformations d'identification qu'IDT utilisera pour se connecter auAWScloud.

En tant que rédacteur de tests, vous devez configurer ces fichiers pourdéboguer votre suite de tests. Vous devez fournir des instructions pour tester les coureurs afin qu'ils puissent configurer les paramètres suivants au besoin pour exécuter vos suites de tests.

Configurer device.json

Le`device.json`contient des informations sur les appareils sur lesquels les tests sont effectués (par exemple, l'adresse IP, les informations de connexion, le système d'exploitation et l'architecture d'UC).

Les coureurs de tests peuvent fournir ces informations via le modèle suivant.`device.json`situé dans le fichier*<device-tester-extract-location>*`/configs/`folder.

```
[
    {
        "id": "<pool-id>",
        "sku": "<pool-sku>",
        "features": [
            {
                "name": "<feature-name>",
                "value": "<feature-value>",
                "configs": [
                    {
                        "name": "<config-name>",
                        "value": "<config-value>"
                    }
                ],
```

```
            }
        ],
        "devices": [
            {
                "id": "<device-id>",
                "connectivity": {
                    "protocol": "ssh | uart | docker",
                    // ssh
                    "ip": "<ip-address>",
                    "port": <port-number>,
                    "auth": {
                        "method": "pki | password",
                        "credentials": {
                            "user": "<user-name>",
                            // pki
                            "privKeyPath": "/path/to/private/key",

                            // password
                            "password": "<password>",
                        }
                    },

                    // uart
                    "serialPort": "<serial-port>",

                    // docker
                    "containerId": "<container-id>",
                    "containerUser": "<container-user-name>",
                }
            }
        ]
    }
]
```

Tous les champs qui contiennent des valeurs sont requis, comme indiqué ici :

`id`

ID alphanumérique défini par l'utilisateur qui identifie de façon unique un ensemble d'appareils appelé un groupe d'appareils. Le matériel doit être identique pour les appareils d'un même groupe. Lorsque vous exécutez une suite de tests, les appareils du groupe sont utilisés pour paralléliser la charge de travail. Plusieurs appareils sont utilisés pour exécuter différents tests.

`sku`

Valeur alphanumérique qui identifie de façon unique l'appareil que vous testez. Le SKU est utilisé pour effectuer le suivi des appareils qualifiés.

> **Note**
>
> Si vous souhaitez ajouter votre carte dans le catalogue d'appareils AWS Partner, la référence que vous indiquez ici doit correspondre à celle indiquée pendant le processus d'élaboration de la liste.

`features`

Facultatif. Un tableau contenant les fonctions prises en charge de l'appareil. Les fonctionnalités de l'appareil sont des valeurs définies par l'utilisateur que vous configurez dans votre suite de tests. Vous devez fournir à vos exécuteurs de tests des informations sur les noms et les valeurs des entités à inclure dans le`device.json`dans le fichier. Par exemple, si vous souhaitez tester un appareil qui fonctionne comme un serveur MQTT pour d'autres appareils, vous pouvez configurer votre logique de test pour valider des niveaux pris en charge spécifiques pour une fonctionnalité nommée`MQTT_QOS`. Les exécuteurs de test fournissent ce nom de fonctionnalité et définissent la valeur de la fonction sur les niveaux de QOS pris en charge par leur appareil. Vous pouvez récupérer les informations fournies à partir duContexte IDTavec le`devicePool.features`ou à partir de la requêtecontexte de machine d'étatavec le`pool.features`requête.

`features.name`

Nom de la fonction.

`features.value`

Les valeurs des entités prises en charge.

`features.configs`

Les paramètres de configuration, le cas échéant, pour la fonctionnalité.

`features.config.name`

Nom du paramètre de configuration.

`features.config.value`

Les valeurs de réglage prises en charge.

`devices`

Un ensemble d'appareils dans le pool à tester. Au moins un appareil est requis.

`devices.id`

Un identificateur unique défini par l'utilisateur pour l'appareil testé.

`connectivity.protocol`

Le protocole de communication utilisé pour communiquer avec cet appareil. Chaque appareil d'un pool doit utiliser le même protocole.

Actuellement, les seules valeurs prises en charge sont.`ssh`et`uart`pour les appareils physiques, et`docker`pour conteneurs Docker.

`connectivity.ip`

L'adresse IP de l'appareil testé.

Cette propriété s'applique uniquement si `connectivity.protocol` est défini sur `ssh`.

`connectivity.port`

Facultatif. Numéro de port à utiliser pour les connexions SSH.

La valeur par défaut est 22.

Cette propriété s'applique uniquement si `connectivity.protocol` est défini sur `ssh`.

`connectivity.auth`

Informations d'authentification pour la connexion.

Cette propriété s'applique uniquement si `connectivity.protocol` est défini sur `ssh`.

`connectivity.auth.method`

Méthode d'authentification utilisée pour accéder à un appareil sur le protocole de connectivité donné.

Les valeurs prises en charge sont :

- `pki`
- `password`

`connectivity.auth.credentials`

Informations d'identification utilisées pour l'authentification.

`connectivity.auth.credentials.password`

Mot de passe utilisé pour se connecter à l'appareil à tester.

Cette valeur s'applique uniquement si `connectivity.auth.method` est défini sur `password`.

`connectivity.auth.credentials.privKeyPath`

Chemin complet de la clé privée utilisée pour se connecter à l'appareil testé.

Cette valeur s'applique uniquement si `connectivity.auth.method` est défini sur `pki`.

`connectivity.auth.credentials.user`

Nom d'utilisateur pour la connexion à l'appareil testé.

`connectivity.serialPort`

Facultatif. Port série auquel l'appareil est connecté.

Cette propriété s'applique uniquement si `connectivity.protocol` est défini sur `uart`.

`connectivity.containerId`

ID de conteneur ou nom du conteneur Docker en cours de test.

Cette propriété s'applique uniquement si `connectivity.protocol` est défini sur `docker`.

`connectivity.containerUser`

Facultatif. Nom de l'utilisateur à l'utilisateur à l'intérieur du conteneur. La valeur par défaut est celle fournie par l'utilisateur dans Dockerfile.

La valeur par défaut est 22.

Cette propriété s'applique uniquement si `connectivity.protocol` est défini sur `docker`.

> **Note**
>
> Pour vérifier si les exécuteurs de tests configurent la connexion de périphérique incorrecte pour un test, vous pouvez

récupérerpool.Devices[0].Connectivity.Protocoldepuis le contexte de la machine d'état et comparez-le à la valeur attendue dans unChoiceétat. Si un protocole incorrect est utilisé, imprimez un message à l'aide de la commandeLogMessageétat et transition vers leFailétat.
Vous pouvez également utiliser un code de gestion des erreurs pour signaler un échec de test pour des types de périphériques incorrects.

(Facultatif) Configuration de userdata.json

Leuserdata.jsoncontient toutes les informations supplémentaires requises par une suite de tests, mais qui ne sont pas spécifiées dans ledevice.jsondans le fichier. Le format de ce fichier dépend de lauserdata_scheme.jsonfichierdéfini dans la suite de tests. Si vous êtes un rédacteur de tests, assurez-vous de fournir ces informations aux utilisateurs qui exécuteront les suites de tests que vous écrivez.

(Facultatif) Configuration de resource.json

Leresource.jsoncontient des informations sur tous les appareils qui seront utilisés comme périphériques de ressources. Les périphériques de ressources sont des appareils nécessaires pour tester certaines fonctionnalités d'un appareil en cours de test. Par exemple, pour tester la capacité Bluetooth d'un appareil, vous pouvez utiliser un périphérique de ressources pour vérifier si votre appareil peut s'y connecter correctement. Les périphériques de ressources sont facultatifs et vous pouvez avoir besoin d'autant d'appareils de ressources que nécessaire. En tant que rédacteur de tests, vous utilisez lefichier test.jsonpour définir les fonctionnalités du périphérique de ressources requises pour un test. Les coureurs de test utilisent ensuite leresource.jsonpour fournir un pool de périphériques de ressources dotés des fonctionnalités requises. Assurez-vous de fournir ces informations aux utilisateurs qui exécuteront les suites de tests que vous écrivez.

Les coureurs de tests peuvent fournir ces informations via le modèle suivant.resource.jsonsitué dans le fichier*<device-tester-extract-location>*/configs/folder.

```
[
    {
        "id": "<pool-id>",
        "features": [
            {
                "name": "<feature-name>",
                "version": "<feature-value>",
```

```
                "jobSlots": <job-slots>
            }
        ],
        "devices": [
            {
                "id": "<device-id>",
                "connectivity": {
                    "protocol": "ssh | uart | docker",
                    // ssh
                    "ip": "<ip-address>",
                    "port": <port-number>,
                    "auth": {
                        "method": "pki | password",
                        "credentials": {
                            "user": "<user-name>",
                            // pki
                            "privKeyPath": "/path/to/private/key",

                            // password
                            "password": "<password>",
                        }
                    },

                    // uart
                    "serialPort": "<serial-port>",

                    // docker
                    "containerId": "<container-id>",
                    "containerUser": "<container-user-name>",
                }
            }
        ]
    }
]
```

Tous les champs qui contiennent des valeurs sont requis, comme indiqué ici :

`id`

ID alphanumérique défini par l'utilisateur qui identifie de façon unique un ensemble d'appareils appelé un groupe d'appareils. Le matériel doit être identique pour les appareils d'un même groupe. Lorsque vous exécutez une suite de tests, les appareils du groupe sont utilisés pour paralléliser la charge de travail. Plusieurs appareils sont utilisés pour exécuter différents tests.

`features`

Facultatif. Un tableau contenant les fonctions prises en charge de l'appareil. Les informations requises dans ce champ sont définies dans lefichiers test.jsondans la suite de tests et détermine quels tests exécuter et comment exécuter ces tests. Si la suite de tests ne nécessite aucune fonctionnalité, ce champ n'est pas obligatoire.

`features.name`

Nom de la fonction.

`features.version`

La version des fonctionnalités.

`features.jobSlots`

Paramètre pour indiquer le nombre de tests pouvant utiliser simultanément l'appareil. La valeur par défaut est 1.

`devices`

Un ensemble d'appareils dans le pool à tester. Au moins un appareil est requis.

`devices.id`

Un identificateur unique défini par l'utilisateur pour l'appareil testé.

`connectivity.protocol`

Le protocole de communication utilisé pour communiquer avec cet appareil. Chaque appareil d'un pool doit utiliser le même protocole.

Actuellement, les seules valeurs prises en charge sont.`ssh`et`uart`pour les appareils physiques, et`docker`pour conteneurs Docker.

`connectivity.ip`

L'adresse IP de l'appareil testé.

Cette propriété s'applique uniquement si `connectivity.protocol` est défini sur `ssh`.

`connectivity.port`

Facultatif. Numéro de port à utiliser pour les connexions SSH.

La valeur par défaut est 22.

Cette propriété s'applique uniquement si `connectivity.protocol` est défini sur `ssh`.

`connectivity.auth`

Informations d'authentification pour la connexion.

Cette propriété s'applique uniquement si `connectivity.protocol` est défini sur `ssh`.

`connectivity.auth.method`

Méthode d'authentification utilisée pour accéder à un appareil sur le protocole de connectivité donné.

Les valeurs prises en charge sont :

- `pki`
- `password`

`connectivity.auth.credentials`

Informations d'identification utilisées pour l'authentification.

`connectivity.auth.credentials.password`

Mot de passe utilisé pour se connecter à l'appareil à tester.

Cette valeur s'applique uniquement si `connectivity.auth.method` est défini sur `password`.

`connectivity.auth.credentials.privKeyPath`

Chemin complet de la clé privée utilisée pour se connecter à l'appareil testé.

Cette valeur s'applique uniquement si `connectivity.auth.method` est défini sur `pki`.

`connectivity.auth.credentials.user`

Nom d'utilisateur pour la connexion à l'appareil testé.

`connectivity.serialPort`

Facultatif. Port série auquel l'appareil est connecté.

Cette propriété s'applique uniquement si `connectivity.protocol` est défini sur `uart`.

`connectivity.containerId`

ID de conteneur ou nom du conteneur Docker en cours de test.

Cette propriété s'applique uniquement si `connectivity.protocol` est défini sur `docker`.

`connectivity.containerUser`

Facultatif. Nom de l'utilisateur à l'utilisateur à l'intérieur du conteneur. La valeur par défaut est celle fournie par l'utilisateur dans Dockerfile.

La valeur par défaut est 22.

Cette propriété s'applique uniquement si `connectivity.protocol` est défini sur `docker`.

(Facultatif) Configuration de config.json

Le`config.json`contient des informations de configuration pour IDT. En règle générale, les exécuteurs de tests n'auront pas besoin de modifier ce fichier, sauf pour fournir leurAWSinformations d'identification utilisateur pour IDT, et éventuellement, unAWSrégion. SiAWSles informations d'identification avec les autorisations requises sont fourniesAWS IoTDevice Tester collecte et soumet des mesures d'utilisation àAWS. Il s'agit d'une fonctionnalité d'option facultative utilisée pour améliorer la fonctionnalité IDT. Pour plus d'informations, consultez Métriques d'utilisation de l'IDT.

Les coureurs de test peuvent configurer leurAWSinformations d'identification de l'une des manières suivantes :

- Fichier d'informations d'identification.

  IDT utilise le même fichier d'informations d'identification que l'AWS CLI. Pour de plus amples informations, veuillez consulter Fichiers de configuration et d'informations d'identification.

  L'emplacement du fichier d'informations d'identification varie en fonction du système d'exploitation que vous utilisez :

  - macOS, Linux : `~/.aws/credentials`
  - Windows: `C:\Users\`*`UserName`*`\.aws\credentials`
- Variables d'environnement

  Les variables d'environnement sont des variables gérées par le système d'exploitation et utilisées par les commandes du système. Les variables définies pendant une session SSH ne sont pas disponibles après la fermeture de cette session. IDT peut utiliser le`AWS_ACCESS_KEY_ID`et`AWS_SECRET_ACCESS_KEY`variables d'environnement à stockerAWSinformations d'identification

  Pour définir ces variables sous Linux, macOS ou Unix, utilisez export:

```
export AWS_ACCESS_KEY_ID=<your_access_key_id>
export AWS_SECRET_ACCESS_KEY=<your_secret_access_key>
```

Pour définir ces variables sous Windows, utilisez set :

```
set AWS_ACCESS_KEY_ID=<your_access_key_id>
set AWS_SECRET_ACCESS_KEY=<your_secret_access_key>
```

Pour configurerAWSinformations d'identification pour IDT, les coureurs de test modifient leauthdans la sectionconfig.jsonsitué dans le fichier*<device-tester-extract-location>*/configs/folder.

```
{
    "log": {
        "location": "logs"
    },
    "configFiles": {
        "root": "configs",
        "device": "configs/device.json"
    },
    "testPath": "tests",
    "reportPath": "results",
    "awsRegion": "<region>",
    "auth": {
        "method": "file | environment",
        "credentials": {
            "profile": "<profile-name>"
        }
    }
}
]
```

Tous les champs qui contiennent des valeurs sont requis, comme indiqué ici :

> **Note**
>
> Tous les chemins d'accès de ce fichier sont définis par rapport au*<device-tester-extract-location>*.

`log.location`

Chemin d'accès au dossier de journaux dans le*<device-tester-extract-location>*.

`configFiles.root`

Chemin d'accès au dossier contenant les fichiers de configuration.

`configFiles.device`

Chemin vers la`device.json`dans le fichier.

`testPath`

Chemin d'accès au dossier contenant des suites de tests.

`reportPath`

Chemin d'accès au dossier qui contiendra les résultats des tests après l'exécution d'une suite de tests par IDT.

`awsRegion`

Facultatif. LeAWSrégion que les suites de tests utiliseront. Si elle n'est pas définie, les suites de tests utiliseront la région par défaut spécifiée dans chaque suite de tests.

`auth.method`

La méthode utilisée par IDT pour récupérerAWSInformations d'identification . Les valeurs prises en charge sont`file`pour récupérer les informations d'identification depuis un fichier d'informations d'identification, et`environment`pour récupérer les informations d'identification à l'aide de variables d'environnement.

`auth.credentials.profile`

Profil d'informations d'identification à utiliser depuis le fichier d'informations d'identification. Cette propriété s'applique uniquement si `auth.method` est défini sur `file`.

Déboguer et exécuter des suites de tests personnalisées

Après leconfiguration requiseest défini, IDT peut exécuter votre suite de tests. L'exécution de l'ensemble de la suite de tests dépend du matériel et de la composition de la suite de tests. Pour référence, il faut environ 30 minutes pour terminer l'exécution de l'ensembleAWS IoT Greengrasssuite de tests de qualification sur un Raspberry Pi 3B.

Lorsque vous écrivez votre suite de tests, vous pouvez utiliser IDT pour exécuter la suite de tests en mode débogage afin de vérifier votre code avant de l'exécuter ou de le fournir aux coureurs de tests.

Exécution d'IDT en mode de débogage

Étant donné que les suites de tests dépendent d'IDT pour interagir avec les appareils, fournir le contexte et recevoir des résultats, vous ne pouvez pas simplement déboguer vos suites de tests dans un IDE sans aucune interaction IDT. Pour ce faire, l'interface de ligne de commande IDT fournit le`debug-test-suite`qui vous permet d'exécuter IDT en mode débogage. Exécutez la commande suivante pour afficher les options disponibles pour`debug-test-suite` :

```
devicetester_[linux | mac | win_x86-64] debug-test-suite -h
```

Lorsque vous exécutez IDT en mode débogage, IDT ne lance pas réellement la suite de tests ni n'exécute la machine d'état. Au lieu de cela, il interagit avec votre EDI pour répondre aux demandes émanant de la suite de tests exécutée dans l'EDI et imprime les journaux sur la console. IDT n'excède pas le délai d'expiration et attend de quitter jusqu'à ce qu'il soit interrompu manuellement. En mode débogage, IDT n'exécute pas non plus la machine d'état et ne génère aucun fichier de rapport. Pour déboguer votre suite de tests, vous devez utiliser votre EDI pour fournir certaines informations qu'IDT obtient généralement à partir des fichiers JSON de configuration. Veillez à spécifier les informations suivantes :

- Variables et arguments d'environnement pour chaque test. IDT ne lira pas ces informations depuis`test.json`ou`suite.json`.
- Arguments permettant de sélectionner des périphériques de ressources. IDT ne lira pas ces informations depuis`test.json`.

Pour déboguer vos suites de tests, procédez comme suit :

1. Créez les fichiers de configuration nécessaires à l'exécution de la suite de tests. Par exemple, si votre suite de tests nécessite le`device.json`,`resource.json`, et`user data.json`, assurez-vous de les configurer tous au besoin.
2. Exécutez la commande suivante pour placer IDT en mode débogage et sélectionner tous les périphériques nécessaires à l'exécution du test.

   ```
   devicetester_[linux | mac | win_x86-64] debug-test-suite [options]
   ```

Après avoir exécuté cette commande, IDT attend les demandes de la suite de tests, puis y répond. IDT génère également les variables d'environnement requises pour le processus de requête du SDK client IDT.

3. Dans votre EDI, utilisez le`runoudebug`pour effectuer les opérations suivantes :

   a. Définissez les valeurs des variables d'environnement générées par IDT.

   b. Définissez la valeur de toutes les variables ou arguments d'environnement que vous avez spécifiés dans votre`test.json`et`suite.json`dans le fichier.

   c. Définissez des points d'arrêt si nécessaire.

4. Exécutez la suite de tests dans votre EDI.

   Vous pouvez déboguer et réexécuter la suite de tests autant de fois que nécessaire. IDT n'excède pas de temps en mode de débogage.

5. Une fois le débogage terminé, interrompre IDT pour quitter le mode de débogage.

Commandes IDT CLI pour exécuter des tests

La section suivante décrit les commandes de l'interface de ligne de commande IDT :

IDT v4.0.0

`help`

Répertorie les informations sur la commande spécifiée.

`list-groups`

Répertorie les groupes dans une suite de tests donnée.

`list-suites`

Répertorie les suites de tests disponibles.

`list-supported-products`

Répertorie les produits pris en charge pour votre version d'IDT, dans ce casAWS IoT Greengrassversions, etAWS IoT Greengrassversions de suite de tests de qualification disponibles pour la version IDT actuelle.

`list-test-cases`

Répertorie les cas de tests d'un groupe de tests donné. L'option suivante est prise en charge :

- `group-id`. Groupe de tests à rechercher. Cette option est obligatoire et doit spécifier un groupe unique.

`run-suite`

Exécute une suite de tests sur un groupe d'appareils. Voici quelques options couramment utilisées :

- `suite-id`. Version de la suite de tests à exécuter. Si celle-ci n'est pas spécifiée, IDT utilise la dernière version dans le dossier `tests`.
- `group-id`. Les groupes de tests à exécuter, sous la forme d'une liste séparée par des virgules. Si cette option n'est pas spécifiée, IDT exécute tous les groupes de tests de la suite de tests.
- `test-id`. Cas de test à exécuter, sous la forme d'une liste séparée par des virgules. Lorsqu'il est spécifié, `group-id` doit spécifier un seul groupe.
- `pool-id`. Le groupe de périphériques à tester. Les exécuteurs de tests doivent spécifier un groupe s'ils ont plusieurs groupes de périphériques définis dans votre`device.json`dans le fichier.
- `timeout-multiplier`. Configure IDT pour modifier le délai d'exécution du test spécifié dans le`test.json`fichier pour un test avec un multiplicateur défini par l'utilisateur.
- `stop-on-first-failure`. Configure IDT pour arrêter l'exécution à la première défaillance. Cette option doit être utilisée avec `group-id` pour déboguer les groupes de tests spécifiés.
- `userdata`. Définit le fichier contenant les informations de données utilisateur nécessaires à l'exécution de la suite de tests. Cette option est obligatoire uniquement si`userdataRequired`est réglé sur true dans le`suite.json`pour la suite de tests.

Pour de plus amples informations sur les options `run-suite`, utilisez l' option `help` suivante :

```
devicetester_[linux | mac | win_x86-64] run-suite -h
```

`debug-test-suite`

Exécutez la suite de tests en mode de débogage. Pour plus d'informations, consultez Exécution d'IDT en mode de débogage.

Examiner les résultats des tests IDT et les journaux

Cette section décrit le format dans lequel IDT génère des journaux de console et des rapports de test.

Format des messages de la console

AWS IoTDevice Tester utilise un format standard pour imprimer des messages sur la console lorsqu'il démarre une suite de tests. L'extrait suivant présente un exemple de message de console généré par 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 plupart des messages de la console se composent des champs suivants :

`time`

Un horodatage ISO 8601 complet pour l'événement enregistré.

`level`

Niveau de message pour l'événement enregistré. En règle générale, le niveau de message enregistré est l'un des`info`,`warn`, ou`error`. IDT émet un`fatal`ou`panic`message s'il rencontre un événement attendu qui provoque sa fermeture anticipée.

`msg`

Le message enregistré.

`executionId`

Chaîne d'identification unique pour le processus IDT actuel. Cet ID est utilisé pour différencier les exécutions IDT individuelles.

Les messages de console générés à partir d'une suite de tests fournissent des informations supplémentaires sur le périphérique testé et sur la suite de tests, le groupe de tests et les cas de test exécutés par IDT. L'extrait suivant illustre un exemple de message de console généré à partir d'une suite de tests.

```
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 partie spécifique de la suite de tests du message de la console contient les champs suivants :

`suiteId`

Nom de la suite de tests en cours d'exécution.

`groupId`

ID du groupe de test en cours d'exécution.

`testCaseId`

L'ID du scénario de test en cours d'exécution.

`deviceId`

Identifiant de l'appareil testé utilisé par le cas de test actuel.

Pour imprimer un résumé de test sur la console lorsqu'un IDT a terminé d'exécuter un test, vous devez inclure un`Reportétat`dans votre machine d'état. Le récapitulatif des tests contient des informations sur la suite de tests, les résultats des tests pour chaque groupe exécuté et les emplacements des journaux et des fichiers de rapport générés. L'exemple suivant montre un message de récapitulatif des tests.

```
========== 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 IoTSchéma de rapport Device Tester

`awsiotdevicetester_report.xml`est un rapport signé contenant les informations suivantes :

- La version IDT.
- Version de la suite de tests.
- Signature et clé du rapport utilisées pour signer le rapport.
- La référence de l'appareil et le nom du groupe d'appareils spécifiés dans la`device.json`dans le fichier.
- La version du produit et les fonctionnalités de l'appareil testées.
- Le récapitulatif des résultats des tests. Ces informations sont les mêmes que celles contenues dans le*`suite-name`*`_report.xml`dans le fichier.

```
<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>
```

Le fichier `awsiotdevicetester_report.xml` contient une balise `<awsproduct>` qui contient des informations relatives au produit testé et les caractéristiques du produit qui ont été validées par une suite de tests.

Attributs utilisés dans la balise **`<awsproduct>`**

`name`

Nom du produit testé.

`version`

Version du produit testé.

`features`

Caractéristiques validées. Caractéristiques marquées comme`required`sont nécessaires pour que la suite de tests puisse valider le périphérique. L'extrait de code suivant montre comment ces informations apparaissent dans le fichier `awsiotdevicetester_report.xml`.

```
<feature name="ssh" value="supported" type="required"></feature>
```

Caractéristiques marquées comme`optional`ne sont pas nécessaires à la validation. Les extraits suivants illustrent des fonctions facultatives.

```
<feature name="hsi" value="supported" type="optional"></feature>

<feature name="mqtt" value="not-supported" type="optional"></feature>
```

Schéma de rapport de suite de tests

Le rapport *`suite-name`*`_Result.xml` est au format JUnit XML. Vous pouvez intégrer des plateformes de déploiement/d'intégration continues tels que Jenkins, Bamboo, etc. Le rapport contient un récapitulatif des résultats des tests.

```
<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 section Rapport dans les deux`awsiotdevicetester_report.xml`ou*suite-name*`_report.xml`répertorie les tests qui ont été exécutés et leurs résultats.

La première balise XML `<testsuites>` contient le résumé de l'exécution des tests. Par exemple :

```
<testsuites name="MyTestSuite results" time="2299" tests="28" failures="0" errors="0"
 disabled="0">
```

Attributs utilisés dans la balise **`<testsuites>`**

`name`

Nom de la suite de tests.

`time`

Durée, en secondes, nécessaire pour exécuter la suite de tests.

`tests`

Nombre de tests exécutés.

`failures`

Nombre de tests exécutés mais dont le résultat n'est pas probant.

`errors`

Nombre de tests qu'IDT n'a pas pu exécuter.

`disabled`

Cet attribut n'est pas utilisé et peut être ignoré.

En cas d'erreurs ou d'échecs de tests, vous pouvez identifier les tests concernés à l'aide des balises XML `<testsuites>`. Les balises XML `<testsuite>` au sein de la balise `<testsuites>` montrent le récapitulatif des résultats d'un groupe de tests. Par exemple :

```
<testsuite name="combination" package="" tests="1" failures="0" time="161" disabled="0"
 errors="0" skipped="0">
```

Le format est similaire à la balise `<testsuites>`, mais avec un attribut appelé `skipped` qui n'est pas utilisé et qui ne peut pas être ignoré. Chaque balise XML `<testsuite>` inclut des balises `<testcase>` pour chaque test exécuté pour un groupe de tests. Par exemple :

```
<testcase classname="Security Test" name="IP Change Tests" attempts="1"></testcase>>
```

Attributs utilisés dans la balise **`<testcase>`**

`name`

Nom du test.

`attempts`

Nombre de fois où IDT a exécuté le test.

Lorsqu'un test échoue ou qu'une erreur se produit, les balises `<failure>` ou `<error>` sont ajoutées à la balise `<testcase>` avec des informations relatives au dépannage. Par exemple :

```
<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>
```

Métriques d'utilisation de l'IDT

Si vous fournissez des AWS informations d'identification avec les autorisations requises, AWS IoT Device Tester collecte et envoie des statistiques d'utilisation à AWS. Il s'agit d'une fonctionnalité opt-in qui est utilisée pour améliorer la fonctionnalité IDT. IDT collecte des informations telles que les suivantes :

- L' Compte AWS ID utilisé pour exécuter IDT
- Les commandes IDT CLI utilisées pour exécuter des tests
- La suite de tests exécutée
- Les suites de tests dans le dossier *< device-tester-extract-location >*
- Le nombre d'appareils configurés dans le pool de périphériques
- Noms des scénarios de test et durées d'exécution
- Informations sur les résultats des tests, par exemple si les tests ont été réussis, ont échoué, ont rencontré des erreurs ou ont été ignorés
- Caractéristiques du produit testées
- Comportement de sortie IDT, tel que les sorties inattendues ou anticipées

Toutes les informations envoyées par IDT sont également enregistrées dans un `metrics.log` fichier du *<device-tester-extract-location>*`/results/`*<execution-id>*`/` dossier. Vous pouvez consulter le fichier journal pour voir les informations collectées lors d'un test. Ce fichier est généré uniquement si vous choisissez de collecter des statistiques d'utilisation.

Pour désactiver la collecte des métriques, il n'est pas nécessaire de prendre d'autres mesures. Ne stockez simplement pas vos AWS informations d'identification et, si vous en avez, ne configurez pas le fichier `config.jso` n pour y accéder. AWS

Configurez vos AWS informations d'identification

Si vous n'en avez pas déjà un Compte AWS, vous devez en créer un. Si vous en avez déjà un Compte AWS, il vous suffit de configurer les autorisations requises pour votre compte qui permettent à IDT d'envoyer des statistiques d'utilisation en votre AWS nom.

Étape 1 : Création d'un Compte AWS

Au cours de cette étape, créez et configurez un Compte AWS. Si vous en avez déjà un Compte AWS, passez directement àthe section called “Étape 2 : Configurer les autorisations pour IDT”.

Inscrivez-vous pour un Compte AWS

Si vous n'en avez pas Compte AWS, procédez comme suit pour en créer un.

Pour vous inscrire à un Compte AWS

1. Ouvrez https://portal.aws.amazon.com/billing/signup.
2. Suivez les instructions en ligne.

   Dans le cadre de la procédure d‘inscription, vous recevrez un appel téléphonique et vous saisirez un code de vérification en utilisant le clavier numérique du téléphone.

   Lorsque vous vous inscrivez à un Compte AWS, un Utilisateur racine d'un compte AWSest créé. Par défaut, seul l‘utilisateur racine a accès à l‘ensemble des Services AWS et des ressources de ce compte. La meilleure pratique en matière de sécurité consiste à attribuer un accès administratif à un utilisateur et à n'utiliser que l'utilisateur root pour effectuer les tâches nécessitant un accès utilisateur root.

AWS vous envoie un e-mail de confirmation une fois le processus d'inscription terminé. Vous pouvez afficher l‘activité en cours de votre compte et gérer votre compte à tout moment en accédant à https://aws.amazon.com/ et en choisissant Mon compte.

Création d'un utilisateur doté d'un accès administratif

Une fois que vous vous êtes inscrit à un utilisateur administratif Compte AWS, que vous Utilisateur racine d'un compte AWS l'avez sécurisé AWS IAM Identity Center, que vous l'avez activé et que vous en avez créé un, afin de ne pas utiliser l'utilisateur root pour les tâches quotidiennes.

Sécurisez votre Utilisateur racine d'un compte AWS

1. Connectez-vous en AWS Management Consoletant que propriétaire du compte en choisissant Utilisateur root et en saisissant votre adresse Compte AWS e-mail. Sur la page suivante, saisissez votre mot de passe.

   Pour obtenir de l'aide pour vous connecter en utilisant l'utilisateur racine, consultez Connexion en tant qu'utilisateur racine dans le Guide de l'utilisateur Connexion à AWS .
2. Activez l'authentification multifactorielle (MFA) pour votre utilisateur racine.

   Pour obtenir des instructions, voir Activer un périphérique MFA virtuel pour votre utilisateur Compte AWS root (console) dans le guide de l'utilisateur IAM.

Création d'un utilisateur doté d'un accès administratif

1. Activez IAM Identity Center.

   Pour obtenir des instructions, consultez Activation d' AWS IAM Identity Center dans le Guide de l'utilisateur AWS IAM Identity Center .
2. Dans IAM Identity Center, accordez un accès administratif à un utilisateur.

   Pour un didacticiel sur l'utilisation du Répertoire IAM Identity Center comme source d'identité, voir Configurer l'accès utilisateur par défaut Répertoire IAM Identity Center dans le Guide de AWS IAM Identity Center l'utilisateur.

Connectez-vous en tant qu'utilisateur disposant d'un accès administratif

- Pour vous connecter avec votre utilisateur IAM Identity Center, utilisez l'URL de connexion qui a été envoyée à votre adresse e-mail lorsque vous avez créé l'utilisateur IAM Identity Center.

  Pour obtenir de l'aide pour vous connecter en utilisant un utilisateur d'IAM Identity Center, consultez la section Connexion au portail AWS d'accès dans le guide de l'Connexion à AWS utilisateur.

Attribuer l'accès à des utilisateurs supplémentaires

1. Dans IAM Identity Center, créez un ensemble d'autorisations conforme aux meilleures pratiques en matière d'application des autorisations du moindre privilège.

Pour obtenir des instructions, voir Création d'un ensemble d'autorisations dans le guide de AWS IAM Identity Center l'utilisateur.

2. Affectez des utilisateurs à un groupe, puis attribuez un accès d'authentification unique au groupe.

   Pour obtenir des instructions, consultez la section Ajouter des groupes dans le guide de AWS IAM Identity Center l'utilisateur.

Étape 2 : Configurer les autorisations pour IDT

Au cours de cette étape, configurez les autorisations utilisées par IDT pour exécuter des tests et collecter les données d'utilisation de l'IDT. Vous pouvez utiliser le AWS Management Console ou AWS Command Line Interface (AWS CLI) pour créer une stratégie IAM et un utilisateur pour IDT, puis associer des politiques à l'utilisateur.

- Pour configurer des autorisations pour IDT (console)
- Pour configurer des autorisations pour IDT (AWS CLI)

Pour configurer des autorisations pour IDT (console)

Procédez comme suit pour utiliser la console afin de configurer les autorisations pour IDT pour AWS IoT Greengrass.

1. Connectez-vous à la console IAM.
2. Créez une stratégie gérée par le client qui accorde des autorisations de création des rôles avec des autorisations spécifiques.

   a. Dans le volet de navigation, sélectionnez Politiques, puis Créer une politique.

   b. Sur l'onglet JSON, remplacez le contenu de l'espace réservé par la stratégie suivante.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot-device-tester:SendMetrics"
            ],
```

```
            "Resource": "*"
        }
    ]
}
```

c. Choisissez Suivant : Balises.

d. Choisissez Suivant : Vérification.

e. Pour Name (Nom), saisissez **`IDTUsageMetricsIAMPermissions`**. Sous Résumé, vérifiez les autorisations accordées par votre stratégie.

f. Choisissez Créer une politique.

3. Créez un utilisateur IAM et associez des autorisations à cet utilisateur.

   a. Créez un utilisateur IAM. Suivez les étapes 1 à 5 de la section Création d'utilisateurs IAM (console) dans le guide de l'utilisateur IAM. Si vous avez déjà créé un utilisateur IAM, passez à l'étape suivante.

   b. Associez les autorisations à votre utilisateur IAM :

      i. Sur la page Définir les autorisations, choisissez Joindre directement les politiques existantes.

      ii. Recherchez la politique IDT UsageMetrics IAMPermissions que vous avez créée à l'étape précédente. Activez la case à cocher.

   c. Choisissez Suivant : Balises.

   d. Choisissez Suivant : Réviser pour afficher un résumé de vos choix.

   e. Choisissez Create user (Créer un utilisateur).

   f. Pour afficher les clés d'accès de l'utilisateur (ID de clé d'accès et clés d'accès secrètes), choisissez Afficher en regard du mot de passe et de la clé d'accès. Pour enregistrer les clés d'accès, choisissez Télécharger .csv, puis enregistrez le fichier dans un emplacement sécurisé sur votre ordinateur. Vous utiliserez ces informations ultérieurement pour configurer votre fichier AWS d'informations d'identification.

Pour configurer des autorisations pour IDT (AWS CLI)

Procédez comme suit pour utiliser le AWS CLI afin de configurer les autorisations pour IDT pour AWS IoT Greengrass. Si vous avez déjà configuré des autorisations dans la console, passez à the section

called “Configurer votre appareil afin d'exécuter des tests IDT” ou the section called “Facultatif : Configuration de votre conteneur Docker”.

1. Sur votre ordinateur, installez et configurez le AWS CLI s'il n'est pas déjà installé. Suivez les étapes décrites AWS CLI dans la section Installation du guide de AWS Command Line Interface l'utilisateur.

   > **Note**
   >
   > AWS CLI Il s'agit d'un outil open source que vous pouvez utiliser pour interagir avec les AWS services à partir de votre shell de ligne de commande.

2. Créez la politique gérée par le client suivante qui accorde les autorisations nécessaires à la gestion de l'IDT et AWS IoT Greengrass des rôles.

   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**
>
> Cette étape inclut un exemple d'invite de commande Windows car elle utilise une syntaxe JSON différente de celle des commandes de terminal Linux, macOS ou Unix.

3. Créez un utilisateur IAM et associez les autorisations requises par IDT pour. AWS IoT Greengrass

   a. Créez un utilisateur IAM.

   ```
   aws iam create-user --user-name user-name
   ```

   b. Attachez la `IDTUsageMetricsIAMPermissions` politique que vous avez créée à votre utilisateur IAM. Remplacez le *nom d'utilisateur par* votre nom d'utilisateur IAM et <account-id>dans la commande par l'ID de votre. Compte AWS

   ```
   aws iam attach-user-policy --user-name user-name --policy-arn
    arn:aws:iam::<account-id>:policy/IDTGreengrassIAMPermissions
   ```

4. Créez une clé d'accès secrète pour l'utilisateur.

   ```
   aws iam create-access-key --user-name user-name
   ```

   Stockez la sortie dans un emplacement sécurisé. Vous utiliserez ces informations ultérieurement pour configurer votre fichier AWS d'informations d'identification.

Fournir des AWS informations d'identification à IDT

Pour autoriser IDT à accéder à vos AWS informations d'identification et à envoyer des métriques à AWS celles-ci, procédez comme suit :

1. Stockez les AWS informations d'identification de votre utilisateur IAM sous forme de variables d'environnement ou dans un fichier d'informations d'identification :

   a. Pour utiliser des variables d'environnement, exécutez la commande suivante :

   ```
   AWS_ACCESS_KEY_ID=access-key
   AWS_SECRET_ACCESS_KEY=secret-access-key
   ```

b. Pour utiliser le fichier d'informations d'identification, ajoutez les informations suivantes au `.aws/credentials file:`

```
[profile-name]
aws_access_key_id=access-key
aws_secret_access_key=secret-access-key
```

2. Configurez la `auth` section du `config.json` fichier. Pour plus d'informations, voir (Facultatif) Configuration de config.json.

Résolution des problèmes liés à IDT pour AWS IoT Greengrass

IDT pour AWS IoT Greengrass écrit ces erreurs dans différents emplacements, en fonction du type d'erreurs. Les erreurs sont écrites dans la console, les fichiers journaux et les rapports de tests.

Codes d'erreur

Le tableau suivant répertorie les codes d'erreur générés par IDT pour AWS IoT Greengrass.

Code d'erreur	Nom du code d'erreur	Cause racine possible	Dépannage
101	InternalError	Une erreur interne s'est produite.	Vérifiez les journaux du répertoire *<device-tester-extract-location>* `/results`. Si vous ne pouvez pas résoudre le problème, contactezAWSDeveloper Support.
102	TimeoutError	Le test ne peut pas être réalisé dans une plage de temps limitée. Ceci peut se produire si :	• Vérifiez la connexion et le débit réseau. • Vérifiez que vous n'avez pas modifié

Code d'erreur	Nom du code d'erreur	Cause racine possible	Dépannage
		• Il y a une connexion réseau lente entre la machine de test et l'appareil (par exemple, si vous utilisez un réseau VPN). • Une connexion réseau lente retarde la communication entre l'appareil et le cloud. • Le champ `timeout` dans les fichiers de configuration test (`test.json`) a été modifié par erreur.	de fichier dans le répertoire `/test`. • Essayez d'exécuter manuellement le groupe de tests en échec avec l'indicateur `"--group-id"` . • Essayez d'exécuter la suite de tests en augmentant les délais d'attente des tests. Pour plus d'informations, consultez Erreurs de délai d'attente.

Code d'erreur	Nom du code d'erreur	Cause racine possible	Dépannage
103	PlatformNotSupportError	Combinaison Système d'exploitation/Architecture incorrecte spécifiée dans `device.json` .	Remplacez votre configuration par l'une des combinaisons prises en charge : • Linux, x86_64 • Linux, ARMv6l • Linux, ARMv7l • Linux, AArch64 • Ubuntu, x86_64 • OpenWRT, ARMv7l • OpenWRT, AArch64 Pour plus d'informations, consultez Configurer device.json.

Code d'erreur	Nom du code d'erreur	Cause racine possible	Dépannage
104	VersionNotSupportError	La version du logiciel AWS IoT Greengrass Core n'est pas prise en charge par la version d'IDT que vous utilisez.	Utilisez la commande device_tester_bin version pour identifier la version prise en charge du logiciel AWS IoT Greengrass Core. Par exemple, si vous utilisez macOS, utilisez : ./devicetester_mac_x86_64 version. Pour rechercher la version du logiciel AWS IoT Greengrass Core que vous utilisez : • Si vous exécutez des tests avec préinstallésAWS IoT Greengrass sLogiciel principal, utilisez SSH pour vous connecter à vos recettesAWS IoT Greengrass spériphérique principal et exécution***<path-to-preinstalled-greengrass-location>*** /greengra

Code d'erreur	Nom du code d'erreur	Cause racine possible	Dépannage
			ss/ggc/core/greengrassd --version • Si vous exécutez des tests avec une autre version du logiciel AWS IoT Greengrass Core, accédez au répertoire `devicetester_greengrass_<os>/products/greengrass/gcc`. La version du logiciel AWS IoT Greengrass Core fait partie du nom du fichier .zip. Vous pouvez tester une autre version du logiciel AWS IoT Greengrass Core. Pour plus d'informations, consultez Commencer avec AWS IoT Greengrass.

Code d'erreur	Nom du code d'erreur	Cause racine possible	Dépannage
105	LanguageNotSupportError	IDT prend uniquement en charge Python pour les kits SDK et bibliothèques AWS IoT Greengrass.	Vérifiez les éléments suivants : • Le package du kit SDK sous `devicetester_greengrass_ <os>/products/greengrass/ggsdk` est le kit SDK Python. • Le contenu du dossier `bin` sous `devicetester_greengrass_ <os>/tests/GGQ_1.0.0/suite/resources/runtimearm/bin` n'a pas été modifié.

Code d'erreur	Nom du code d'erreur	Cause racine possible	Dépannage
106	ValidationError	Certains champs de `device.json` ou de `config.json` sont non valides.	Vérifiez le message d'erreur situé à droite du code d'erreur dans le rapport. • Type d'authentification non valide pour l'appareil : Spécifiez la méthode appropriée pour se connecter à votre appareil. Pour plus d'informations, consultez the section called “Configurer device.json”. • Chemin de clé privée non valide : Spécifiez le chemin correct vers votre clé privée. Pour plus d'informations, consultez Configurer device.json. • Non valideRégion AWS : Spécifiez un code valideRégion AWSdans vos recettes`config.json` dans le fichier. Pour plus d'informa

Code d'erreur	Nom du code d'erreur	Cause racine possible	Dépannage
			tions, consultez Points de terminaison du service AWS. • AWSinformations d'identification : Set valideAWSinformations d'identification de votre machine de test (en utilisant des variables d'environnement ou le `credentials` fichier). Vérifiez que le champ `auth` est correctement configuré. Pour plus d'informations, consultez the section called “Créez et configurez un Compte AWS”. • Entrée HSM non valide : Vérifier vos recettes `pkcs11Provider` ,`privateKeyLabel` ,`slotLabel` ,`slotUserPin` , et `openSSLEngine` champs dans `device.json` .

Code d'erreur	Nom du code d'erreur	Cause racine possible	Dépannage
107	SSHConnectionFailed	La machine de test ne peut pas se connecter à l'appareil configuré.	Vérifiez que les champs suivants sont corrects dans votre fichier `device.json` : • `ip` • `user` • `privKeyPath` • `password` Pour plus d'informations, consultez Configurer device.json.

Code d'erreur	Nom du code d'erreur	Cause racine possible	Dépannage
108	RunCommandError	Un test n'a pas réussi à exécuter une commande sur l'appareil testé.	Vérifiez que l'accès racine est autorisé pour l'utilisateur configuré dans `device.json` . Un mot de passe est requis par certains appareils lors de l'exécuti on de commandes avec accès racine. Assurez-vous que l'accès racine est autorisé sans mot de passe. Pour de plus amples informati ons, veuillez consulter la documentation relative à votre appareil. Essayez d'exécute r manuellement la commande qui a échoué sur votre appareil pour vérifier si une erreur se produit.
109	PermissionDeniedEr ror	Aucun accès racine.	Définissez l'accès racine pour l'utilisa teur configuré sur votre appareil.

Code d'erreur	Nom du code d'erreur	Cause racine possible	Dépannage
110	CreateFileError	Impossible de créer un fichier.	Vérifiez l'espace disque de votre appareil et les autorisations sur les répertoires.
111	CreateDirError	Impossible de créer un répertoire.	Vérifiez l'espace disque de votre appareil et les autorisations sur les répertoires.
112	InvalidPathError	Le chemin vers le logiciel AWS IoT Greengrass Core est incorrect.	Vérifiez que le chemin indiqué dans le message d'erreur est valide. Ne modifiez aucun fichier sous le répertoire `devicetester_greengrass_` *<os>*.
113	InvalidFileError	Un fichier n'est pas valide.	Vérifiez que le fichier indiqué dans le message d'erreur est valide.

Code d'erreur	Nom du code d'erreur	Cause racine possible	Dépannage
114	ReadFileError	Le fichier spécifié ne peut pas être lu.	Vérifiez les paramètres suivants : • Les autorisations de fichiers sont correctes. • `limits.config` autorise l'ouverture d'un nombre de fichiers suffisant. • Le chemin spécifié dans le message d'erreur existe et est valide. Si vous procédez aux tests sur Mac OS, augmentez le nombre limite de fichiers ouverts. La limite par défaut est 256, ce qui est suffisant pour les tests.

Code d'erreur	Nom du code d'erreur	Cause racine possible	Dépannage
115	FileNotFoundError	Un fichier requis est introuvable.	Vérifiez les paramètres suivants : • Un fichier Greengrass compressé existe sous `devicetester_greengrass_<os>/products/greengrass/ggc`. Vous pouvez télécharger leAWS IoT Greengrass Fichier tar de base à partir du fichierAWS IoT GreengrassLogiciel Corepage des téléchargements. • Le package du kit SDK figure sous `devicetester_greengrass_<os>/products/greengrass/ggsdk`. • Les fichiers sous `devicetester_greengrass_<os>/`

Code d'erreur	Nom du code d'erreur	Cause racine possible	Dépannage
			`tests` n'ont pas été modifiés.
116	OpenFileFailed	Impossible d'ouvrir le fichier spécifié.	Vérifiez les paramètres suivants : • Le chemin spécifié dans le message d'erreur existe et est valide. • `limits.config` autorise l'ouverture d'un nombre de fichiers suffisant. Si vous procédez aux tests sur Mac OS, augmentez le nombre limite de fichiers ouverts. La limite par défaut est 256, ce qui est suffisant pour les tests.
117	WriteFileFailed	Échec de l'écriture du fichier (peut être l'appareil testé ou la machine de test).	Vérifiez que le répertoire spécifié dans le message d'erreur existe et que vous disposez d'une autorisation en écriture.

Code d'erreur	Nom du code d'erreur	Cause racine possible	Dépannage
118	FileCleanUpError	Pendant le test, le fichier ou le répertoire spécifié n'a pas pu être supprimé, ou le fichier spécifié n'a pas pu être démonté de l'appareil distant.	Si le fichier binaire est toujours en cours d'exécution, il est possible que le fichier soit verrouillé. Terminez le processus et supprimez le fichier spécifié.
119	InvalidInputError	Configuration non valide.	Vérifiez que votre fichier `suite.json` est valide.

Code d'erreur	Nom du code d'erreur	Cause racine possible	Dépannage
120	InvalidCredentialError	Non valideAWS Informations d'identification .	• Vérifier vos recettesAWSInformations d'identification . Pour plus d'informations, consultez the section called “Configuration de vos informations d'identification pour l'AWS”. • Vérifiez votre connexion réseau et réexécutez le groupe de tests. Les problèmes de réseau peuvent également occasionner cette erreur.
121	AWSSessionError	Impossible de créer une session AWS.	Cette erreur peut se produire siAWSLes informations d'identification ne sont pas valides ou la connexion Internet est instable. Essayez d'utiliser l'AWS CLI pour appeler une opération d'API AWS.

Code d'erreur	Nom du code d'erreur	Cause racine possible	Dépannage
122	AWSApiCallError	Une erreur d'API AWS s'est produite.	Cette erreur peut être due à un problème de réseau. Vérifiez votre réseau avant de relancer le groupe de tests.
123	IpNotExistError	L'adresse IP ne figure pas dans les informations de connectivité.	Vérifiez votre connexion Internet. Vous pouvez utiliser le pluginAWS IoT Greengrasspour vérifier les informations de connectivité duAWS IoT Greengrassélément essentiel utilisé par le test. Si 10 points de terminaison figurent dans les informations de connectivité, vous pouvez en supprimer certains ou tous les supprimer, puis relancer le test. Pour de plus amples informations, veuillez consulter Informations de connectivité.
124	OTAJobNotCompleteError	Une tâche OTA ne s'est pas terminée.	Vérifiez votre connexion réseau et réexécutez le groupe de tests OTA.

Code d'erreur	Nom du code d'erreur	Cause racine possible	Dépannage
125	CreateGreengrassServiceRoleError	L'une des erreurs suivantes s'est produite : • Une erreur s'est produite lors de la création d'un rôle. • Une erreur s'est produite lors de l'attachement d'une stratégie au rôle de service AWS IoT Greengrass. • La stratégie associée au rôle de service n'est pas valide. • Une erreur s'est produite lors de l'association d'un rôle à unCompte AWS.	Configurez le rôle du service AWS IoT Greengrass. Pour plus d'informations, consultez the section called “Rôle de service Greengrass”.

Code d'erreur	Nom du code d'erreur	Cause racine possible	Dépannage
126	DependenciesNotPresentError	Une ou plusieurs dépendances requises pour le test spécifique ne sont pas présentes sur l'appareil.	Consultez le journal de test pour voir quelles dépendances manquent sur votre appareil : *<device-tester-extract-location>* `/results/` *<execution-id>*`/logs/`*<test-case-name.log>* .
127	InvalidHSMConfiguration	La configuration HSM/PKCS fournie est incorrecte.	Dans votre fichier `device.json` , fournissez la configuration requise pour interagir avec HSM en utilisant PKCS#11.

Code d'erreur	Nom du code d'erreur	Cause racine possible	Dépannage
128	OTAJobNot SuccededError	La tâche OTA n'a pas abouti.	• Si vous avez exécuté le groupe de test `ota` individuellement, exécutez le groupe de test `ggcdependencies` pour vérifier que toutes les dépendances (telles que `wget`) sont présentes. Essayez ensuite à nouveau le groupe de test `ota`. • Consultez les journaux détaillés sous *`<device-tester-extract-location>`* `/results/` *`<execution-id>`* `/logs/` pour obtenir des informations relatives à la résolution des problèmes et aux erreurs. Plus précisément, vérifiez les journaux suivants : •

Code d'erreur	Nom du code d'erreur	Cause racine possible	Dépannage
			Journal de la console (`test_manager.log`) • Journal des cas de test OTA (`ota_test.log`) • Journal du démon GGC (`ota_test_ggc_logs.tar.gz`) • Journaux de l'agent OTA (`ota_test_ota_logs.tar.gz`) • Vérifiez votre connexion réseau et exécutez à nouveau le groupe de tests `ota`. • Si le problème persiste, contactez AWSDeveloper Support.

Code d'erreur	Nom du code d'erreur	Cause racine possible	Dépannage
129	NoConnectivityError	L'agent hôte ne parvient pas à se connecter à Internet.	Vérifiez vos paramètres de connexion réseau et de pare-feu. Réessayez le groupe de test une fois le problème de connectivité résolu.
130	NoPermissionError	L'utilisateur IAM que vous utilisez pour exécuter IDTAWS IoT Greengrassn'est pas autorisé à créer le fichierAWSressourc es nécessaires à l'exécution d'IDT.	Veuillez consulter Modèle de stratégie d'autorisations pour connaître le modèle de stratégie qui accorde les autorisat ions requises pour exécuter IDT pour AWS IoT Greengrass.

Code d'erreur	Nom du code d'erreur	Cause racine possible	Dépannage
131	LeftoverAgentExist Error	Votre appareil exécute les processus AWS IoT Greengrass lorsque vous essayez de démarrer IDT pour AWS IoT Greengrass.	Assurez-vous qu'aucun démon Greengrass n'est en cours d'exécution sur votre appareil. • Vous pouvez utiliser cette commande pour arrêter le démon : sudo ./***<absolute-path-to-greengrass-daemon>*** / greengrassd stop. • Vous pouvez également mettre fin au démon Greengrass par PID. **Note** Si vous utilisez une installation existante de AWS IoT Greengrass configurée pour démarrer automatiquement après

Code d'erreur	Nom du code d'erreur	Cause racine possible	Dépannage
			le redémarrage, vous devez arrêter le démon après le redémarrage et avant d'exécuter la suite de tests.
132	DeviceTimeOffsetError	L'heure de l'appareil est incorrecte.	Réglez votre appareil sur l'heure correcte.
133	InvalidMLConfiguration	La configuration ML fournie est incorrecte.	Dans votre fichier `device.json` , indiquez la configuration requise pour exécuter les tests d'inférence ML. Pour plus d'informations, consultez the section called “Facultatif : Configuration de votre appareil pour la qualification ML”.

Résolution des erreurs liées à IDT pour AWS IoT Greengrass

Lorsque vous utilisez IDT, vous devez obtenir les fichiers de configuration appropriés en place avant d'exécuter IDT AWS IoT Greengrass. Si vous obtenez des erreurs d'analyse et de configuration, la première étape consiste à localiser et à utiliser un modèle de configuration approprié pour votre environnement.

Si le problème persiste, consultez les processus de débogage suivants.

Rubriques

Où rechercher des erreurs ?

Les erreurs de haut niveau sont affichées sur la console pendant l'exécution et un récapitulatif des tests ayant échoué avec indication de l'erreur s'affiche lorsque tous les tests sont terminés. `awsiotdevicetester_report.xml` contient un récapitulatif de toutes les erreurs qui provoquaient l'échec d'un test. Les fichiers journaux pour chaque série de tests sont stockés dans un répertoire dont le nom comporte un UUID relatif à l'exécution du test, affiché sur la console pendant la série de tests.

Le répertoire des journaux de test est situé dans *`<device-tester-extract-location>`*`/results/`*`<execution-id>`*`/logs/`. Ce répertoire contient les fichiers suivants, qui sont utiles pour le débogage.

Fichier	Description
`test_manager.log`	Contient tous les journaux qui ont été écrits dans la console lors de l'exécution du test. Un récapitulatif des résultats est situé à la fin de ce fichier, et comprend une liste des tests qui ont échoué.

Fichier	Description
	Les journaux des erreurs et des avertissements de ce fichier peuvent vous donner des informations sur les défaillances.
<test-group-id> __*<test-name>* .log	Journaux détaillés pour le test spécifique.
<nom-test> _ggc_logs.tar.gz	Collection compressée de tous les journaux duAWS IoT Greengrassle démon noyau généré au cours du test. Pour plus d'informations, consultez Dépannage de AWS IoT Greengrass.
<test-name> _ota_logs.tar.gz	Il s'agit d'une collection compressée de journaux générés par l'agent OTA AWS IoT Greengrass pendant le test. Pour les tests OTA uniquement.
<test-name> _basic_assertion_publisher_ggad_logs.tar.gz	Collection compressée de journaux générés par l'appareil de l'éditeur AWS IoT pendant le test.
<test-name> _basic_assertion_subscriber_ggad_logs.tar.gz	Ensemble compressé de journaux générés par l'appareil de l'abonné AWS IoT pendant le test.

Erreurs d'analyse

Parfois, une faute de frappe dans une configuration JSON peut entraîner des erreurs d'analyse. La plupart du temps, le problème est lié à l'absence d'une virgule, d'une apostrophe ou d'un crochet dans le fichier JSON. IDT effectue la validation JSON et imprime les informations de débogage. Il imprime la ligne dans laquelle l'erreur s'est produite, le numéro de ligne et le numéro de colonne de l'erreur de syntaxe. Ces informations devraient suffire pour vous aider à résoudre l'erreur, mais si vous ne parvenez toujours pas à localiser cette dernière, vous pouvez effectuer la validation manuellement dans un environnement de développement intégré, un éditeur de texte tel qu'Atom ou Sublime, ou via un outil en ligne comme JSONLint.

Erreur liée à un paramètre obligatoire manquant

Les fichiers de configuration peuvent être modifiés pour refléter les nouvelles fonctions ajoutées régulièrement à IDT. L'utilisation d'un ancien fichier de configuration peut corrompre votre configuration. Si tel est le cas, le fichier *<test_case_id>*`.log` sous `/results/`*<execution-id>*`/logs` répertorie explicitement tous les paramètres manquants. IDT vérifie également les schémas de votre fichier de configuration JSON pour s'assurer que la dernière version prise en charge a bien été utilisée.

Erreur indiquant l'impossibilité de démarrer le test

Vous pouvez rencontrer des erreurs qui renvoient à une défaillance lors du démarrage du test. Il y a plusieurs causes possibles. Par conséquent, procédez de la façon suivante :

- Assurez-vous que le nom du groupe que vous avez inclus dans votre commande d'exécution existe réellement. Le nom du groupe est référencé directement à partir du fichier `device.json`.
- Assurez-vous que les appareils du groupe ont des paramètres de configuration corrects.

Erreur d'absence d'autorisation d’accès à la ressource

Vous pouvez voir le message d’erreur `<user or role> is not authorized to access this resource` dans la sortie du terminal ou dans le fichier `test_manager.log` sous `/results/`*<execution-id>*`/logs`. Pour résoudre ce problème, attachez la stratégie gérée `AWSIoTDeviceTesterForGreengrassFullAccess` à votre utilisateur de test. Pour plus d'informations, consultez the section called “Créez et configurez un Compte AWS”.

Erreurs de type « Autorisation refusée »

IDT effectue des opérations sur différents répertoires et fichiers d'un appareil testé. Certaines de ces opérations nécessitent un accès racine. Pour automatiser ces opérations, IDT doit être en mesure d'exécuter des commandes avec la commande sudo sans avoir à saisir un mot de passe.

Suivez les étapes ci-après pour autoriser l'accès sudo sans saisie de mot de passe.

> **Note**
>
> `user` et `username` font référence à l'utilisateur SSH utilisé par IDT pour accéder à l'appareil testé.

1. Utilisez sudo usermod -aG sudo ***<ssh-username>*** pour ajouter l'utilisateur SSH au groupe sudo.
2. Déconnectez-vous, puis reconnectez-vous pour que les modifications entrent en vigueur.
3. Ouvrez le fichier `/etc/sudoers`, puis ajoutez la ligne suivante à la fin du fichier : *<ssh-username>* `ALL=(ALL) NOPASSWD: ALL`

> **Note**
>
> En tant que bonne pratique, nous vous recommandons d'utiliser sudo visudo lorsque vous modifiez `/etc/sudoers`.

Erreurs de connexion SSH

Lorsqu'IDT ne parvient pas à se connecter à un appareil testé, les échecs de connexion sont consignés dans `/results/`*<execution-id>*`/logs/`*<test-case-id>*`.log`. Les messages d'échec SSH apparaissent en haut de ce fichier journal, car la connexion à un appareil testé est l'une des premières opérations effectuées par IDT.

La plupart des configurations Windows utilisent l'application de terminal PuTTY pour se connecter aux hôtes Linux. Cette application nécessite que les fichiers de clé privée PEM standard soient convertis dans un format propriétaire Windows appelé PPK. Lorsqu'IDT est configuré dans votre fichier `device.json`, utilisez les fichiers PEM uniquement. Si vous utilisez un fichier PPK, IDT ne peut pas créer de connexion SSH avec l'appareil AWS IoT Greengrass et ne peut donc pas exécuter les tests.

Erreurs de délai d'attente

Vous pouvez augmenter le délai d'attente pour chaque test en spécifiant un multiplicateur de délai d'attente, qui sera appliqué à la valeur par défaut du délai d'attente de chaque test. Toute valeur configurée pour cet indicateur doit être supérieure ou égale à 1.0.

Pour utiliser le multiplicateur de délai d'attente, utilisez l'indicateur `--timeout-multiplier` lors de l'exécution de tests. Par exemple :

```
./devicetester_linux run-suite --suite-id GGQ_1.0.0 --pool-id DevicePool1 --timeout-
multiplier 2.5
```

Pour de plus amples informations, exécutez `run-suite --help`.

Erreurs liées à des commandes introuvables lors des tests

Une ancienne version de la bibliothèque OpenSSL (libssl1.0.0) est requise pour l'exécution des tests OTA sur les appareils AWS IoT Greengrass. La plupart des distributions Linux récentes utilisent libssl 1.0.2 ou version ultérieure (v1.1.0).

Par exemple, sur un Raspberry Pi, exécutez les commandes suivantes pour installer la version requise de 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
   ```

Exception de sécurité sur macOS

Lorsque vous exécutez IDT sur une machine hôte utilisant macOS 10.15, le ticket de notarisation d'IDT n'est pas correctement détecté et IDT est bloqué. Pour exécuter IDT, vous devez accorder une exception de sécurité au`devicetester_mac_x86-64`exécutable.

Pour accorder une exception de sécurité à l'exécutable IDT

1. Lancement d'Préférences systèmedepuis le menu Apple.
2. ChoisissezSécurité et confidentialité, puis sur leGénéral, cliquez sur l'icône représentant un verrou pour modifier les paramètres de sécurité.
3. Recherchez le message`"devicetester_mac_x86-64" was blocked from use because it is not from an identified developer.`et choisissezAutoriser quand même.
4. Acceptez l'avertissement de sécurité.

Si vous avez des questions sur la politique de support IDT, contactezAWSsupport client.

Politique de prise en charge d’AWS IoT Device Tester pour AWS IoT Greengrass V1

AWS IoTDevice Tester pourAWS IoT Greengrassest un framework de test téléchargeable qui vous permet de valider etqualifiervotreAWS IoT Greengrassdispositifs à inclure dans leAWS

PartnerCatalogue d'appareils. Nous vous recommandons d'utiliser la version la plus récente deAWS IoT Greengrasset IDT pour tester ou qualifier vos appareils. Pour de plus amples informations, veuillez consulterVersions d'IDT prises en charge pourAWS IoT Greengrass V2dans leAWS IoT Greengrass Version 2Manuel du développeur.

Vous pouvez également toutes les versions prises en charge deAWS IoT Greengrasset IDT pour tester ou qualifier vos appareils. Bien que vous puissiez continuer à utiliserversions d'IDT non prises en charge, ces versions ne reçoivent pas de corrections de bogues ou de mises à jour.

> **Important**
>
> Au 4 avril 2022,AWS IoTDevice Tester pourAWS IoT Greengrass V1ne génère plus de rapports de qualification signés. Vous ne pouvez plus être qualifié pour un nouveauAWS IoT Greengrass V1appareils à mettre en vente dans leAWS PartnerCatalogue d'appareilsvia leAWSProgramme de qualification d'appareils. Bien que vous ne puissiez pas qualifier les appareils Greengrass V1, vous pouvez continuer à utiliser IDT pourAWS IoT Greengrass V1pour tester vos appareils Greengrass V1. Nous vous recommandons d'utiliserIDT pourAWS IoT Greengrass V2pour qualifier et répertorier les appareils Greengrass dans leAWS PartnerCatalogue d'appareils.

Si vous avez des questions sur la politique de support, contactez le support client AWS.

Résolution des problèmes de AWS IoT Greengrass

Cette section fournit des informations de dépannage et des solutions possibles pour vous aider à résoudre les problèmes liés à AWS IoT Greengrass.

Pour de plus amples informations sur les quotas (limites) AWS IoT Greengrass, veuillez consulter Quotas de service dans la Référence générale d'Amazon Web Services.

Problèmes liés à AWS IoT Greengrass Core

Si le logiciel AWS IoT Greengrass Core ne démarre pas, essayez les étapes générales de résolution des problèmes suivantes :

- Veillez à installer les fichiers binaires appropriés pour votre architecture. Pour plus d'informations, consultez Logiciel AWS IoT Greengrass Core.
- Assurez-vous que votre appareil principal dispose d'un stockage local disponible. Pour plus d’informations, consultez the section called “Résolution des problèmes de stockage”.
- Recherchez les éventuels messages d'erreur dans `runtime.log` et `crash.log`. Pour plus d’informations, consultez the section called “Résolution des problèmes liés aux journaux”.

Effectuez une recherche dans les symptômes et erreurs suivants pour trouver des informations qui vous aideront à résoudre les problèmes liés à un AWS IoT Greengrass noyau.

Problèmes

- Erreur : le fichier de configuration ne contient pas CaPath le CertPath ou KeyPath. Le processus de démon Greengrass avec [pid = <pid>] a expiré.
- Erreur : impossible d'analyser /<greengrass-root>/config/config.json.
- Erreur : Une erreur s'est produite lors de la génération de la configuration TLS : ErrUnknown UriScheme
- Erreur : échec de démarrage de Runtime : impossible de démarrer les composants Worker : le test de conteneur a expiré.
- <address>Erreur : échec de l'appel PutLogEvents sur Cloudwatch local, LogGroup :GreengrassSystem//connection_manager, erreur : échec de l'envoi de la demande causé par RequestError : Post http :<path>///cloudwatch/logs/ : dial tcp : getsockopt : connexion refusée, réponse : {}.

- <region><account-id><function-name><version><file-name>Erreur : Impossible de créer le serveur en raison de : échec du chargement du groupe : chmod/<greengrass-root>/ggc/deployment/lambda/arn:aws:lambda : ::function : :/: aucun fichier ou répertoire de ce type.
- Le logiciel AWS IoT Greengrass Core ne démarre pas une fois que vous êtes passé d'une exécution sans conteneurisation à une exécution dans un conteneur Greengrass.
- Erreur : la taille du dossier spool doit être d'au moins 262 144 octets.
- Erreur : [ERREUR] - Erreur de messagerie cloud : une erreur s'est produite lors de la tentative de publication d'un message. {"errorString": "operation timed out"}
- Erreur : container_linux.go:344: starting container process caused "process_linux.go:424: container init caused \"rootfs_linux.go:64: mounting \\\"/greengrass/ggc/socket/greengrass_ipc.sock\\\" to rootfs \\\"/greengrass/ggc/packages/<version>/rootfs/merged\\\" at \\\"/greengrass_ipc.sock\\\" caused \\\"stat /greengrass/ggc/socket/greengrass_ipc.sock: permission denied\\\"\"".
- Erreur : exécution du démon Greengrass avec le PID : <id-processus>. Certains composants du système n'ont pas pu démarrer. Recherchez les erreurs dans runtime.log.
- Le shadow de l’appareil n'est pas synchronisé avec le cloud.
- ERREUR : impossible d'accepter la connexion TCP. accept tcp [::]:8000: accept4: trop de fichiers ouverts.
- Erreur : erreur d'exécution de Runtime : impossible de démarrer le conteneur lambda. container_linux.go:259: starting container process caused "process_linux.go:345: container init caused \"rootfs_linux.go:50: preparing rootfs caused \\\"permission denied\\\"\"".
- Avertissement : [WARN] - [5] GK Remote : Erreur lors de la récupération des données de clé publique ErrPrincipalNotConfigured : la clé privée pour n' MqttCertificate est pas définie.
- <account-id><role-name><region>Erreur : autorisation refusée lors de la tentative d'utilisation du rôle arn:aws:iam : :role/ pour accéder à l'URL s3 https ://-greengrass-updates.s3.<region><architecture><distribution-version>.amazonaws.com/core/ /greengrass-core- .tar.gz.
- Le AWS IoT Greengrass cœur est configuré pour utiliser un proxy réseau et votre fonction Lambda ne peut pas établir de connexions sortantes.
- Le noyau se trouve dans une boucle de connexion-déconnexion infinie. Le fichier runtime.log contient une série continue d'entrées de connexion et de déconnexion.
- 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 \\\"proc\\\" to rootfs \\\"
- [ERREUR] -erreur d'exécution du runtime : impossible de démarrer le conteneur Lambda. <ggc-path>{"ErrorString » : « Impossible d'initialiser le montage des conteneurs : impossible de masquer

la racine de Greengrass dans le répertoire supérieur de superposition : échec de création du périphérique de masque dans le répertoire : le fichier existe »}

- [ERREUR] - Le déploiement a échoué. {"DeploymentID » : <deployment-id>"«, « errorString » : « <pid>échec du processus de test du conteneur avec pid : état du processus du conteneur : état du processus du conteneur : état de sortie 1"}
- Error: [ERROR]-runtime execution error: unable to start lambda container. {"errorString": "failed to initialize container mounts: failed to create overlay fs for container: mounting overlay at /greengrass/ggc/packages/<ggc-version>/rootfs/merged failed: failed to mount with args source=\"no_source\" dest=\"/greengrass/ggc/packages/<ggc-version>/rootfs/merged\" fstype=\"overlay\" flags=\"0\" data=\"lowerdir=/greengrass/ggc/packages/<ggc-version>/dns:/,upperdir=/greengrass/ggc/packages/<ggc-version>/rootfs/upper,workdir=/greengrass/ggc/packages/<ggc-version>/rootfs/work\": too many levels of symbolic links"}
- Error: [DEBUG]-Failed to get routes. Discarding message.
- Error: [Errno 24] Too many open <lambda-function>,[Errno 24] Too many open files
- Erreur : le serveur DS n'a pas pu démarrer l'écoute du socket : listen unix <ggc-path>/ggc/socket/greengrass_ipc.sock : bind : argument non valide
- [INFO] (Photocopieur) aws.greengrass. StreamManager: stdout. Causé par : com.fasterxml.jackson.databind. JsonMappingException: l'instant dépasse l'instant minimum ou maximum
- GPG error: https://dnw9lb6lzp2d8.cloudfront.net stable InRelease: The following signatures were invalid: EXPKEYSIG 68D644ABD2327D47 AWS Greengrass Master Key

Erreur : le fichier de configuration ne contient pas CaPath le CertPath ou KeyPath. Le processus de démon Greengrass avec [pid = <pid>] a expiré.

Solution : vous pouvez voir cette erreur dans `crash.log` lorsque le logiciel AWS IoT Greengrass Core ne démarre pas. Cela peut se produire si vous exécutez la version 1.6 ou une version antérieure. Effectuez l'une des actions suivantes :

- Effectuez une mise à niveau vers la version 1.7 ou ultérieure. Nous vous recommandons de toujours exécuter la dernière version du logiciel AWS IoT Greengrass Core. Pour obtenir des informations sur le téléchargement, consultez Logiciel AWS IoT Greengrass Core.

- Utilisez le format `config.json` correct pour votre version du logiciel AWS IoT Greengrass Core. Pour plus d'informations, consultez the section called "Fichier de configuration de AWS IoT Greengrass Core".

> **Note**
>
> Pour connaître la version du logiciel AWS IoT Greengrass Core installée sur l'appareil principal, exécutez les commandes suivantes dans le terminal de votre appareil.
>
> ```
> cd /greengrass-root/ggc/core/
> sudo ./greengrassd --version
> ```

Erreur : impossible d'analyser /<greengrass-root>/config/config.json.

Solution : vous pouvez voir cette erreur lorsque le logiciel AWS IoT Greengrass Core ne démarre pas. Assurez-vous que le fichier de configuration Greengrass utilise un format JSON valide.

Ouvrez `config.json` (situé dans */greengrass-root*`/config`) et vérifiez le format JSON. Par exemple, assurez-vous que les virgules sont utilisées correctement.

Erreur : Une erreur s'est produite lors de la génération de la configuration TLS : ErrUnknown UriScheme

Solution : vous pouvez voir cette erreur lorsque le logiciel AWS IoT Greengrass Core ne démarre pas. Assurez-vous que les propriétés de la section crypto du fichier de configuration Greengrass sont valides. Le message d'erreur doit fournir plus d'informations.

Ouvrez `config.json` (situé dans */greengrass-root*`/config`) et vérifiez la section `crypto`. Par exemple, les chemins de certificat et de clé doivent utiliser le format d'URI correct et pointer vers l'emplacement correct.

Erreur : échec de démarrage de Runtime : impossible de démarrer les composants Worker : le test de conteneur a expiré.

Solution : vous pouvez voir cette erreur lorsque le logiciel AWS IoT Greengrass Core ne démarre pas. Définissez la propriété `postStartHealthCheckTimeout` dans le fichier de configuration Greengrass. Cette propriété facultative configure la durée (en millisecondes) pendant laquelle le démon Greengrass attend la fin de la vérification de l'état post-démarrage. La valeur par défaut est de 30 secondes (30000 ms).

Ouvrez `config.json` (situé dans */greengrass-root*`/config`). Dans l'objet `runtime`, ajoutez la propriété `postStartHealthCheckTimeout` et définissez la valeur sur un nombre supérieur à 30000. Ajoutez une virgule si nécessaire pour créer un fichier JSON valide. Par exemple :

```
...
"runtime" : {
  "cgroup" : {
    "useSystemd" : "yes"
  },
  "postStartHealthCheckTimeout" : 40000
},
...
```

<address>Erreur : échec de l'appel PutLogEvents sur Cloudwatch local, LogGroup :GreengrassSystem//connection_manager, erreur : échec de l'envoi de la demande causé par RequestError : Post http :<path>/// cloudwatch/logs/ : dial tcp : getsockopt : connexion refusée, réponse : {}.

Solution : vous pouvez voir cette erreur lorsque le logiciel AWS IoT Greengrass Core ne démarre pas. Cela peut se produire si vous exécutez AWS IoT Greengrass sur un appareil Raspberry Pi et que la configuration de la mémoire requise n'a pas été effectuée. Pour plus d'informations, consultez cette étape.

<region><account-id><function-name><version><file-name>Erreur : Impossible de créer le serveur en raison de : échec du chargement du groupe : chmod/<greengrass-root>/ggc/deployment/lambda/ arn:aws:lambda : ::function : :/: aucun fichier ou répertoire de ce type.

Solution : vous pouvez voir cette erreur lorsque le logiciel AWS IoT Greengrass Core ne démarre pas. Si vous avez déployé un exécutable Lambda sur le noyau, vérifiez les propriétés de la fonction dans le `group.json` fichier (`Handler`situé dans */greengrass-root /ggc/deployment/ group*). Si le gestionnaire n'a pas le même nom que le fichier exécutable compilé, remplacez le contenu du fichier `group.json` par un objet JSON vide (`{}`) et exécutez les commandes suivantes pour démarrer AWS IoT Greengrass :

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

Ensuite, utilisez l'API AWS Lambda pour mettre à jour le paramètre `handler` de la configuration de la fonction, publier une nouvelle version de la fonction, et mettre à jour l'alias. Pour de plus amples informations, veuillez consulter Versions et alias des fonctions AWS Lambda.

En supposant que vous ayez ajouté par alias la fonction à votre groupe Greengrass (recommandé), vous pouvez maintenant redéployer votre groupe. (Si ce n'est pas le cas, vous devez pointer vers la nouvelle version ou le nouvel alias de la fonction dans votre définition de groupe et les abonnements avant de déployer le groupe.)

Le logiciel AWS IoT Greengrass Core ne démarre pas une fois que vous êtes passé d'une exécution sans conteneurisation à une exécution dans un conteneur Greengrass.

Solution : vérifiez qu'aucune dépendance de conteneur ne manque.

Erreur : la taille du dossier spool doit être d'au moins 262 144 octets.

Solution : vous pouvez voir cette erreur lorsque le logiciel AWS IoT Greengrass Core ne démarre pas. Ouvrez le fichier `group.json` (situé dans */greengrass-root*`/ggc/deployment/group`),

remplacez le contenu du fichier par un objet JSON vide ({}) et exécutez les commandes suivantes pour démarrer AWS IoT Greengrass :

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

Ensuite, suivez les étapes de la procédure the section called “Pour mettre en cache des messages dans le stockage local”. Pour la fonction `GGCloudSpooler`, assurez-vous de spécifier une valeur `GG_CONFIG_MAX_SIZE_BYTES` supérieure ou égale à 262 144.

Erreur : [ERREUR] - Erreur de messagerie cloud : une erreur s'est produite lors de la tentative de publication d'un message. {"errorString": "operation timed out"}

Solution : Cette erreur peut s'afficher dans `GGCloudSpooler.log` lorsque le noyau Greengrass ne peut pas envoyer de messages MQTT à AWS IoT Core. Cela peut se produire si l'environnement principal a une bande passante limitée et une latence élevée. Si vous exécutez AWS IoT Greengrass version 1.10.2 ou ultérieure, essayez d'augmenter la valeur de `mqttOperationTimeout` dans le fichier config.json. Si la propriété n'est pas présente, ajoutez-la à l'objet `coreThing`. Par exemple :

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

La valeur par défaut est 5 et la valeur minimale est 5.

Erreur : container_linux.go:344: starting container process caused "process_linux.go:424: container init caused \"rootfs_linux.go:64: mounting \\\"/greengrass/ggc/socket/greengrass_ipc.sock\\\" to rootfs \\\"/greengrass/ggc/packages/<version>/rootfs/merged\\\" at \\\"/greengrass_ipc.sock\\\" caused \\\"stat /greengrass/ggc/socket/greengrass_ipc.sock: permission denied\\\"\"".

Solution : vous pouvez voir cette erreur dans `runtime.log` lorsque le logiciel AWS IoT Greengrass Core ne démarre pas. Cela se produit si la valeur de `umask` est supérieure à `0022`. Actuellement, pour résoudre ce problème, vous devez affecter à `umask` la valeur `0022` ou une valeur inférieure. La valeur `0022` accorde à tout le monde l'autorisation de lecture sur les nouveaux fichiers par défaut.

Erreur : exécution du démon Greengrass avec le PID : <id-processus>. Certains composants du système n'ont pas pu démarrer. Recherchez les erreurs dans runtime.log.

Solution : vous pouvez voir cette erreur lorsque le logiciel AWS IoT Greengrass Core ne démarre pas. Recherchez des informations spécifiques sur l'erreur dans `runtime.log` et `crash.log`. Pour plus d'informations, consultez the section called “Résolution des problèmes liés aux journaux”.

Le shadow de l'appareil n'est pas synchronisé avec le cloud.

Solution : assurez-vous qu'AWS IoT Greengrass possède les autorisations d'exécution des actions `iot:UpdateThingShadow` et `iot:GetThingShadow` et dans le rôle de service Greengrass. Si le rôle de service utilise la stratégie gérée `AWSGreengrassResourceAccessRolePolicy`, ces autorisations sont incluses par défaut.

veuillez consulter Dépannage des problèmes de temporisation de la synchronisation shadow.

ERREUR : impossible d'accepter la connexion TCP. accept tcp [::]:8000: accept4: trop de fichiers ouverts.

Solution : vous pouvez voir cette erreur dans la sortie du script `greengrassd`. Cela peut se produire si la limite de descripteur de fichier du logiciel AWS IoT Greengrass Core a atteint le seuil défini et doit être augmentée.

Utilisez la commande suivante et redémarrez le logiciel AWS IoT Greengrass Core.

```
ulimit -n 2048
```

> **Note**
>
> Dans cet exemple, la limite est augmentée à 2 048. Choisissez une valeur adaptée à votre cas d'utilisation.

Erreur : erreur d'exécution de Runtime : impossible de démarrer le conteneur lambda. container_linux.go:259: starting container process caused "process_linux.go:345: container init caused \"rootfs_linux.go:50: preparing rootfs caused \\\"permission denied\\\"\"".

Solution : installez AWS IoT Greengrass directement dans le répertoire racine, ou assurez-vous que le répertoire où le logiciel AWS IoT Greengrass Core est installé et ses répertoires parents disposent d'autorisations `execute` pour tout le monde.

Avertissement : [WARN] - [5] GK Remote : Erreur lors de la récupération des données de clé publique ErrPrincipalNotConfigured : la clé privée pour n' MqttCertificate est pas définie.

Solution : AWS IoT Greengrass utilise un gestionnaire courant pour valider les propriétés de tous les mandataires de sécurité. Cet avertissement dans `runtime.log` est prévu, sauf si vous avez spécifié

une clé privée personnalisée pour le serveur MQTT local. Pour plus d'informations, consultez the section called "Mandataires de sécurité".

<account-id><role-name><region>Erreur : autorisation refusée lors de la tentative d'utilisation du rôle arn:aws:iam : :role/ pour accéder à l'URL s3 https ://-greengrass-updates.s3. <region><architecture><distribution-version>.amazonaws.com/core/ /greengrass-core- .tar.gz.

Solution : cette erreur peut s'afficher en cas d'échec d'une mise à jour over-the-air (OTA). Dans la politique de rôle du signataire, ajoutez la cible Région AWS en tant que`Resource`. Le rôle signataire est utilisé pour pré-signer l'URL S3 pour la mise à jour du logiciel AWS IoT Greengrass. Pour plus d'informations, consultez Rôle de signataire d'URL S3.

Le AWS IoT Greengrass cœur est configuré pour utiliser un proxy réseau et votre fonction Lambda ne peut pas établir de connexions sortantes.

Solution : En fonction de votre environnement d'exécution et des exécutables utilisés par la fonction Lambda pour créer des connexions, vous pouvez également recevoir des erreurs de temporisation de connexion. Assurez-vous que vos fonctions Lambda utilisent la configuration de proxy appropriée pour se connecter via le proxy réseau. AWS IoT Greengrasstransmet la configuration du proxy aux fonctions Lambda définies par l'utilisateur via `http_proxy` les variables d'`https_proxy`environnement, `no_proxy` et. On peut y accéder comme illustré dans l'extrait de code Python suivant.

```
import os
print(os.environ['http_proxy'])
```

Utilisez la même casse que la variable définie dans votre environnement, par exemple, `http_proxy` en minuscules ou `HTTP_PROXY` en majuscules. Pour ces variables, AWS IoT Greengrass prend en charge les deux possibilités.

> **Note**
>
> Les bibliothèques courantes utilisées pour établir des connexions (par exemple, boto3 ou cURL et les packages `requests` Python) se servent de ces variables d'environnement par défaut.

Le noyau se trouve dans une boucle de connexion-déconnexion infinie. Le fichier runtime.log contient une série continue d'entrées de connexion et de déconnexion.

Solution : cela peut se produire lorsqu'un autre appareil est codé de manière irréversible pour utiliser le nom d'objet principal comme ID client pour les connexions MQTT à AWS IoT. Les connexions simultanées sont identiques Région AWS et Compte AWS doivent utiliser des identifiants clients uniques. Par défaut, le cœur utilise le nom d'objet principal comme ID client pour ces connexions.

Pour résoudre ce problème, vous pouvez modifier l'ID client utilisé par l'autre périphérique pour la connexion (recommandé) ou remplacer la valeur par défaut pour le cœur.

Pour remplacer l'ID client par défaut pour le périphérique principal

1. Exécutez la commande suivante pour arrêter le daemon Greengrass :

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

2. Ouvrez *greengrass-root*`/config/config.json` pour le modifier en tant qu'utilisateur su.
3. Dans l'objet `coreThing`, ajoutez la propriété `coreClientId` et définissez la valeur sur votre ID client personnalisé. La valeur doit comprendre entre 1 et 128 caractères. Il doit être unique dans le courant Région AWS pour leCompte AWS.

   ```
   "coreClientId": "MyCustomClientId"
   ```

4. Lancez le démon.

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

```
sudo ./greengrassd start
```

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 \\\"proc\\\" to rootfs \\\"

Solution : Sur certaines plateformes, cette erreur peut s'afficher `runtime.log` lorsque vous essayez de AWS IoT Greengrass monter le système de `/proc` fichiers pour créer un conteneur Lambda. Vous pouvez également voir des erreurs similaires, telles que `operation not permitted` ou `EPERM`. Ces erreurs peuvent se produire même si les tests exécutés sur la plateforme par le script du vérificateur de dépendance aboutissent.

Essayez l'une des solutions suivantes :

- Activez l'option `CONFIG_DEVPTS_MULTIPLE_INSTANCES` dans le noyau Linux.
- Définissez les options de montage `/proc` sur l'hôte sur `rw,relatim` uniquement.
- Mettez à niveau le noyau Linux vers la version 4.9 ou ultérieure.

> **Note**
>
> Ce problème n'est pas lié au montage `/proc` pour l'accès aux ressources locales.

[ERREUR] -erreur d'exécution du runtime : impossible de démarrer le conteneur Lambda. <ggc-path>{"ErrorString » : « Impossible d'initialiser le montage des conteneurs : impossible de masquer la racine de Greengrass dans le répertoire supérieur de superposition : échec de création du périphérique de masque dans le répertoire : le fichier existe »}

Solution : cette erreur peut s'afficher dans le fichier runtime.log en cas d'échec du déploiement. Cette erreur se produit si une fonction Lambda du AWS IoT Greengrass groupe ne peut pas accéder au `/usr` répertoire dans le système de fichiers du noyau.

Pour résoudre ce problème, ajoutez une ressource de volume local au groupe, puis déployez le groupe. Cette ressource doit :

- Spécifier `/usr` comme chemin source et chemin de destination
- Ajouter automatiquement les autorisations de groupe de système d'exploitation du groupe Linux qui possède la ressource
- Affiliez-vous à la fonction Lambda et autorisez l'accès en lecture seule.

[ERREUR] - Le déploiement a échoué. {"DeploymentID » : <deployment-id>"«, « errorString » : « <pid>échec du processus de test du conteneur avec pid : état du processus du conteneur : état du processus du conteneur : état de sortie 1"}

Solution : cette erreur peut s'afficher dans le fichier runtime.log en cas d'échec du déploiement. Cette erreur se produit si une fonction Lambda du AWS IoT Greengrass groupe ne peut pas accéder au `/usr` répertoire dans le système de fichiers du noyau.

Vous pouvez confirmer que tel est le cas en vérifiant s'il n'y a `GGCanary.log` pas d'autres erreurs. Si la fonction Lambda ne peut pas accéder au `/usr` répertoire, il `GGCanary.log` contiendra l'erreur suivante :

```
[ERROR]-standard_init_linux.go:207: exec user process caused "no such file or
 directory"
```

Pour résoudre ce problème, ajoutez une ressource de volume local au groupe, puis déployez le groupe. Cette ressource doit :

- Spécifier `/usr` comme chemin source et chemin de destination
- Ajouter automatiquement les autorisations de groupe de système d'exploitation du groupe Linux qui possède la ressource
- Affiliez-vous à la fonction Lambda et autorisez l'accès en lecture seule.

Error: [ERROR]-runtime execution error: unable to start lambda container. {"errorString": "failed to initialize container mounts: failed to create overlay fs for container: mounting overlay at /greengrass/ggc/packages/<ggc-version>/rootfs/merged failed: failed to mount with args source=\"no_source\" dest=\"/greengrass/ggc/packages/<ggc-version>/rootfs/merged\" fstype=\"overlay\" flags=\"0\" data=\"lowerdir=/greengrass/ggc/packages/<ggc-version>/dns:/,upperdir=/greengrass/ggc/packages/<ggc-version>/rootfs/upper,workdir=/greengrass/ggc/packages/<ggc-version>/rootfs/work\": too many levels of symbolic links"}

Solution : cette erreur peut s'afficher dans le `runtime.log` fichier lorsque le logiciel AWS IoT Greengrass Core ne démarre pas. Ce problème peut être plus fréquent sur les systèmes d'exploitation Debian.

Pour résoudre ce problème, procédez comme suit :

1. Mettez à niveau le logiciel AWS IoT Greengrass Core vers la version v1.9.3 ou ultérieure. Cela devrait résoudre automatiquement ce problème.
2. Si cette erreur persiste après la mise à niveau du logiciel AWS IoT Greengrass Core, définissez la `system.useOverlayWithTmpfs` propriété sur `true` dans le fichier config.json.

   Example Exemple

   ```
   {
     "system": {
   ```

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

> **Note**
>
> Votre version du logiciel AWS IoT Greengrass Core s'affiche dans le message d'erreur. Pour rechercher la version de votre noyau Linux, exécutez `uname -r`.

Error: [DEBUG]-Failed to get routes. Discarding message.

Solution : vérifiez les abonnements dans votre groupe et assurez-vous que l'abonnement répertorié dans le message `[DEBUG]` existe.

Error: [Errno 24] Too many open <lambda-function>,[Errno 24] Too many open files

Solution : cette erreur peut s'afficher dans le fichier journal de votre fonction Lambda si la fonction est instanciée `StreamManagerClient` dans le gestionnaire de fonctions. Nous vous recommandons de créer le client en dehors du gestionnaire. Pour plus d'informations, consultez the section called "Utiliser StreamManagerClient pour travailler avec des flux".

Erreur : le serveur DS n'a pas pu démarrer l'écoute du socket : listen unix <ggc-path>/ggc/socket/greengrass_ipc.sock : bind : argument non valide

Solution : cette erreur peut s'afficher lorsque le logiciel AWS IoT Greengrass Core ne démarre pas. Cette erreur se produit lorsque le logiciel AWS IoT Greengrass Core est installé dans un dossier dont le chemin de fichier est long. Réinstallez le logiciel AWS IoT Greengrass Core dans un dossier dont le chemin de fichier contient moins de 79 octets, si vous n'utilisez pas de répertoire d'écriture, ou 83 octets, si vous utilisez un répertoire d'écriture.

[INFO] (Photocopieur) aws.greengrass. StreamManager: stdout. Causé par : com.fasterxml.jackson.databind. JsonMappingException: l'instant dépasse l'instant minimum ou maximum

Lorsque vous mettez à niveau le logiciel AWS IoT Greengrass principal vers la version v1.11.3, l'erreur suivante peut s'afficher dans les journaux du gestionnaire de flux si le gestionnaire de flux ne démarre pas.

```
2021-07-16T00:54:58.568Z [INFO] (Copier) aws.greengrass.StreamManager:
 stdout. Caused by: com.fasterxml.jackson.databind.JsonMappingException:
 Instant exceeds minimum or maximum instant (through reference chain:
 com.amazonaws.iot.greengrass.streammanager.export.PersistedSuccessExportStatesV1["lastExportTi
 {scriptName=services.aws.greengrass.StreamManager.lifecycle.startup.script,
 serviceName=aws.greengrass.StreamManager, currentState=STARTING}
2021-07-16T00:54:58.579Z [INFO] (Copier) aws.greengrass.StreamManager: stdout.
 Caused by: java.time.DateTimeException: Instant exceeds minimum or maximum instant.
 {scriptName=services.aws.greengrass.StreamManager.lifecycle.startup.script,
 serviceName=aws.greengrass.StreamManager, currentState=STARTING}
```

Si vous utilisez une version du logiciel AWS IoT Greengrass principal antérieure à la version v1.11.3 et que vous souhaitez passer à une version ultérieure, utilisez une mise à jour OTA pour passer à la version v1.11.4.

GPG error: https://dnw9lb6lzp2d8.cloudfront.net stable InRelease: The following signatures were invalid: EXPKEYSIG 68D644ABD2327D47 AWS Greengrass Master Key

Lorsque vous l'exécutez `apt update` sur un appareil sur lequel vous avez installé le logiciel AWS IoT Greengrass principal à partir d'un dépôt APT, l'erreur suivante peut s'afficher.

```
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
```

Cette erreur se produit car il AWS IoT Greengrass n'est plus possible d'installer ou de mettre à jour le logiciel AWS IoT Greengrass principal à partir du référentiel APT. Pour une exécution réussie`apt update`, supprimez le AWS IoT Greengrass référentiel de la liste des sources de l'appareil.

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

Problèmes de déploiement

Utilisez les informations suivantes pour résoudre les problèmes de déploiement.

Problèmes

- Votre déploiement actuel ne fonctionne pas et vous souhaitez revenir à un déploiement opérationnel précédent.
- L'erreur 403 Interdit s'affiche dans les journaux lors du déploiement.
- Une ConcurrentDeployment erreur se produit lorsque vous exécutez la commande create-deployment pour la première fois.
- Erreur : Greengrass n'est pas autorisé à assumer le rôle de service associé à ce compte, ou erreur : Échec : le rôle de service TES n'est pas associé à ce compte.
- Erreur : impossible d'exécuter l'étape de téléchargement dans le déploiement. erreur lors du téléchargement : erreur lors du téléchargement du fichier de définition de groupe :... x509 : le certificat a expiré ou n'est pas encore valide
- Le déploiement n'est pas terminé.
- Erreur : Impossible de trouver les exécutables Java ou Java8, ou erreur : <deployment-id><group-id>échec du déploiement du type NewDeployment pour le groupe Erreur : le travailleur n'a <worker-id>pas pu s'initialiser avec raison La version Java installée doit être supérieure ou égale à 8
- Le déploiement n'est pas terminé et runtime.log contient plusieurs entrées « attendre 1 s que le conteneur s'arrête ».

- Le déploiement ne se termine pas et runtime.log contient « [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>Erreur : <deployment-id><group-id>échec du déploiement du type NewDeployment pour le groupe Erreur : erreur lors du traitement. la configuration du groupe n'est pas valide : 112 ou [1190] n'ont pas l'autorisation rw sur le fichier :.
- Erreur : < list-of-function-arns > sont configurés pour s'exécuter en tant que root, mais Greengrass n'est pas configuré pour exécuter les fonctions Lambda avec les autorisations root.
- Erreur : <deployment-id><group-id>échec du déploiement du type NewDeployment pour le groupe Erreur : erreur de déploiement de Greengrass : impossible d'exécuter l'étape de téléchargement lors du déploiement. erreur lors du traitement : impossible de charger le fichier de groupe téléchargé : UID introuvable sur la base du nom d'utilisateur, UserName : ggc_user : user : unknown user ggc_user.
- Error: [ERROR]-runtime execution error: unable to start lambda container. {"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"}
- Erreur : échec <deployment-id>du déploiement du type NewDeployment pour le groupe <group-id>Erreur : échec du démarrage du processus : container_linux.go:259 : le démarrage du processus conteneur a provoqué « process_linux.go:250 : running exec setns process for init caused \" wait : no child processes \ "».
- <host-prefix>Erreur : [WARN] -MQTT [client] compose tcp : lookup -ats.iot.<region>.amazonaws.com : aucun hôte de ce type... [ERREUR] -Erreur de déploiement de Greengrass : impossible de signaler l'état du déploiement au cloud... net/http : demande annulée en attendant la connexion (Client.Timeout dépassé pendant l'attente des en-têtes)

Votre déploiement actuel ne fonctionne pas et vous souhaitez revenir à un déploiement opérationnel précédent.

Solution : utilisez la AWS IoT console ou AWS IoT Greengrass l'API pour redéployer un déploiement fonctionnel précédent. Cela déploie la version de groupe correspondante sur votre appareil principal.

Pour redéployer un déploiement (console)

1. Sur la page de configuration du groupe, choisissez l'onglet Déploiements. Cette page affiche l'historique de déploiement du groupe, y compris la date et l'heure, la version du groupe et l'état de chaque tentative de déploiement.
2. Recherchez la ligne qui contient le déploiement que vous souhaitez redéployer. Sélectionnez le déploiement que vous souhaitez redéployer, puis choisissez Redéployer.

Pour redéployer un déploiement (interface de ligne de commande)

1. ListDeploymentsUtilisez-le pour trouver l'ID du déploiement que vous souhaitez redéployer. Par exemple :

```
aws greengrass list-deployments --group-id 74d0b623-c2f2-4cad-9acc-ef92f61fcaf7
```

La commande renvoie la liste des déploiements pour le groupe.

```
{
    "Deployments": [
        {
            "DeploymentId": "8d179428-f617-4a77-8a0c-3d61fb8446a6",
            "DeploymentType": "NewDeployment",
            "GroupArn": "arn:aws:greengrass:us-west-2:123456789012:/greengrass/
groups/74d0b623-c2f2-4cad-9acc-ef92f61fcaf7/versions/8dd1d899-4ac9-4f5d-
afe4-22de086efc62",
            "CreatedAt": "2019-07-01T20:56:49.641Z"
        },
        {
            "DeploymentId": "f8e4c455-8ac4-453a-8252-512dc3e9c596",
            "DeploymentType": "NewDeployment",
```

```
            "GroupArn": "arn:aws:greengrass:us-west-2::123456789012:/greengrass/
groups/74d0b623-c2f2-4cad-9acc-ef92f61fcaf7/versions/4ad66e5d-3808-446b-940a-
b1a788898382",
            "CreatedAt": "2019-07-01T20:41:47.048Z"
        },
        {
            "DeploymentId": "e4aca044-bbd8-41b4-b697-930ca7c40f3e",
            "DeploymentType": "NewDeployment",
            "GroupArn": "arn:aws:greengrass:us-west-2::123456789012:/greengrass/
groups/74d0b623-c2f2-4cad-9acc-ef92f61fcaf7/versions/1f3870b6-850e-4c97-8018-
c872e17b235b",
            "CreatedAt": "2019-06-18T15:16:02.965Z"
        }
    ]
}
```

> **Note**
>
> Ces commandes AWS CLI utilisent des exemples de valeurs pour le groupe et l'ID de déploiement. Lorsque vous exécutez les commandes, veillez à remplacer les exemples de valeurs.

2. CreateDeploymentÀ utiliser pour redéployer le déploiement cible. Définissez le type de déploiement sur `Redeployment`. Par exemple :

```
aws greengrass create-deployment --deployment-type Redeployment \
  --group-id 74d0b623-c2f2-4cad-9acc-ef92f61fcaf7 \
  --deployment-id f8e4c455-8ac4-453a-8252-512dc3e9c596
```

La commande renvoie l'ARN et l'ID du nouveau déploiement.

```
{
    "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. GetDeploymentStatusUtilisez-le pour obtenir l'état du déploiement.

L'erreur 403 Interdit s'affiche dans les journaux lors du déploiement.

Solution : assurez-vous que la politique du AWS IoT Greengrass noyau dans le cloud inclut `"greengrass:*"` une action autorisée.

Une ConcurrentDeployment erreur se produit lorsque vous exécutez la commande create-deployment pour la première fois.

Solution : un déploiement est peut-être en cours. Vous pouvez exécuter get-deployment-status pour voir si un déploiement a été créé. Si ce n'est pas le cas, essayez de recréer le déploiement.

Erreur : Greengrass n'est pas autorisé à assumer le rôle de service associé à ce compte, ou erreur : Échec : le rôle de service TES n'est pas associé à ce compte.

Solution : vous pouvez voir cette erreur lorsque le déploiement échoue. Vérifiez qu'un rôle de service Greengrass est actuellement associé à votre rôle Compte AWS dans le service. Région AWS Pour plus d'informations, consultez the section called "Gestion du rôle de service (interface de ligne de commande)" ou the section called "Gestion du rôle de service (console)".

Erreur : impossible d'exécuter l'étape de téléchargement dans le déploiement. erreur lors du téléchargement : erreur lors du téléchargement du fichier de définition de groupe :... x509 : le certificat a expiré ou n'est pas encore valide

Solution : vous pouvez voir cette erreur dans `runtime.log` lorsque le déploiement échoue. Si vous recevez une erreur `Deployment failed` contenant le message `x509: certificate has expired or is not yet valid`, vérifiez l'horloge de l'appareil. Les certificats TLS et X.509

fournissent une base sécurisée pour la construction de systèmes IoT, mais ils nécessitent des temps précis sur les serveurs et les clients. Les appareils IoT doivent avoir l'heure correcte (dans les 15 minutes) avant de tenter de se connecter à AWS IoT Greengrass ou à d'autres services TLS qui utilisent des certificats de serveur. Pour plus d'informations, consultez la section Utilisation de l'heure de l'appareil pour valider les certificats de AWS IoT serveur sur l'Internet des objets sur le blog AWS officiel.

Le déploiement n'est pas terminé.

Solution : effectuez les opérations suivantes :

- Assurez-vous que le démon AWS IoT Greengrass est en cours d'exécution sur votre appareil principal. Dans le terminal de votre appareil principal, exécutez les commandes suivantes pour vérifier si le démon est en cours d'exécution et démarrez-le, si nécessaire.

  1. Pour vérifier si le démon est en cours d'exécution :

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

     Si la sortie contient une entrée `root` pour `/greengrass/ggc/packages/1.11.6/bin/daemon`, le démon est en cours d'exécution.

     La version du chemin d'accès dépend de la version du logiciel AWS IoT Greengrass Core installée sur votre appareil principal.

  2. Pour démarrer le daemon, procédez comme suit :

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

- Assurez-vous que votre appareil principal est connecté et que les points de terminaison de connexion à l'appareil principal sont configurés correctement.

Erreur : Impossible de trouver les exécutables Java ou Java8, ou erreur : <deployment-id><group-id>échec du déploiement du type NewDeployment pour le groupe Erreur : le travailleur n'a <worker-id>pas pu s'initialiser avec raison La version Java installée doit être supérieure ou égale à 8

Solution : Si le gestionnaire de flux est activé pour le AWS IoT Greengrass noyau, vous devez installer le runtime Java 8 sur le périphérique principal avant de déployer le groupe. Pour de plus amples informations, veuillez consulter la configuration requise pour le gestionnaire de flux. Le gestionnaire de flux est activé par défaut lorsque vous utilisez le flux de travail de création de groupes par défaut dans la AWS IoT console pour créer un groupe.

Ou, désactivez le gestionnaire de flux, puis déployez le groupe. Pour plus d'informations, consultez the section called "Configurer les paramètres (console)".

Le déploiement n'est pas terminé et runtime.log contient plusieurs entrées « attendre 1 s que le conteneur s'arrête ».

Solution : exécutez les commandes suivantes sur votre terminal principal pour redémarrer le démon AWS IoT Greengrass.

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

Le déploiement ne se termine pas et **`runtime.log`** contient « [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"} ».

Solution : Cette erreur peut s'afficher dans `runtime.log` lorsque le noyau Greengrass est configuré pour utiliser une connexion proxy HTTPS et que la chaîne de certificats du serveur proxy n'est pas

approuvée sur le système. Pour essayer de résoudre ce problème, ajoutez la chaîne de certificats au certificat d'autorité de certification racine. Le noyau Greengrass ajoute les certificats de ce fichier dans le groupe de certificats utilisé pour l'authentification TLS dans les connexions HTTPS et MQTT avec AWS IoT Greengrass.

L'exemple suivant présente un certificat d'autorité de certification de serveur proxy ajouté au fichier de certificat d'autorité de certification racine :

```
# My proxy CA
-----BEGIN CERTIFICATE-----
MIIEFTCCAv2gAwIQWgIVAMHSAzWG/5YVRYtRQOxXUTEpHuEmApzGCSqGSIb3DQEK
\nCwUAhuL9MQswCQwJVUzEPMAVUzEYMBYGA1UECgwP1hem9uLmNvbSBJbmMuMRww
... content of proxy CA certificate ...
+vHIRlt0e5JAm5\noTIZGoFbK82A0/nO7f/t5PSIDAim9V3Gc3pSXxCCAQoFYnui
GaPUlGk1gCE84a0X\n7Rp/lND/PuMZ/s8YjlkY2NmYmNjMCAXDTE5MTEyN2cM216
gJMIADggEPADf2/m45hzEXAMPLE=
-----END CERTIFICATE-----

# Amazon Root CA 1
-----BEGIN CERTIFICATE-----
MIIDQTCCAimgF6AwIBAgITBmyfz/5mjAo54vB4ikPmljZKyjANJmApzyMZFo6qBg
ADA5MQswCQYDVQQGEwJVUzEPMA0tMVT8QtPHRh8jrdkGA1UEChMGDV3QQDExBBKW
... content of root CA certificate ...
o/ufQJQWUCyziar1hem9uMRkwFwYVPSHCb2XV4cdFyQzR1KldZwgJcIQ6XUDgHaa
5MsI+yMRQ+hDaXJiobldXgjUka642M4UwtBV8oK2xJNDd2ZhwLnoQdeXeGADKkpy
rqXRfKoQnoZsG4q5WTP46EXAMPLE
-----END CERTIFICATE-----
```

Par défaut, le fichier de certificat d'autorité de certification racine se trouve dans */greengrass-root*`/certs/root.ca.pem`. Pour rechercher l'emplacement sur votre appareil noyau, vérifiez la propriété `crypto.caPath` dans config.json.

> **Note**
>
> *greengrass-root* indique le chemin d'installation du logiciel AWS IoT Greengrass Core sur votre appareil. Généralement, il s'agit du répertoire `/greengrass`.

<path>Erreur : <deployment-id><group-id>échec du déploiement du type NewDeployment pour le groupe Erreur : erreur lors du traitement. la configuration du groupe n'est pas valide : 112 ou [119 0] n'ont pas l'autorisation rw sur le fichier :.

Solution : assurez-vous que le groupe propriétaire du répertoire *<chemin>* dispose des autorisations de lecture et d'écriture sur le répertoire.

Erreur : < list-of-function-arns > sont configurés pour s'exécuter en tant que root, mais Greengrass n'est pas configuré pour exécuter les fonctions Lambda avec les autorisations root.

Solution : vous pouvez voir cette erreur dans `runtime.log` lorsque le déploiement échoue. Assurez-vous que vous avez configuré AWS IoT Greengrass pour autoriser les fonctions Lambda à s'exécuter avec les autorisations root. Modifiez la valeur de `allowFunctionsToRunAsRoot` in `greengrass_root/config/config.json` to `yes` ou modifiez la fonction Lambda pour qu'elle s'exécute en tant qu'autre utilisateur/groupe. Pour plus d'informations, consultez the section called "Exécution d'une fonction Lambda en tant que root".

Erreur : <deployment-id><group-id>échec du déploiement du type NewDeployment pour le groupe Erreur : erreur de déploiement de Greengrass : impossible d'exécuter l'étape de téléchargement lors du déploiement. erreur lors du traitement : impossible de charger le fichier de groupe téléchargé : UID introuvable sur la base du nom d'utilisateur, UserName : ggc_user : user : unknown user ggc_user.

Solution : Si l'identité d'accès par défaut du AWS IoT Greengrass groupe utilise les comptes système standard, l'`ggc_user`utilisateur et le `ggc_group` groupe doivent être présents sur l'appareil. Pour obtenir des instructions qui expliquent comment ajouter l'utilisateur et le groupe, consultez cette étape. Assurez-vous d'entrer les noms exactement comme indiqué.

Error: [ERROR]-runtime execution error: unable to start lambda container. {"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"}

Solution : vous pouvez voir cette erreur dans `runtime.log` lorsque le déploiement échoue. Cette erreur se produit si une fonction Lambda du groupe Greengrass ne peut pas accéder au `/usr` répertoire dans le système de fichiers du noyau. Pour résoudre ce problème, ajoutez une ressource de volume local au groupe, puis déployez le groupe. La ressource doit :

- Spécifier `/usr` comme chemin source et chemin de destination
- Ajouter automatiquement les autorisations de groupe de système d'exploitation du groupe Linux qui possède la ressource
- Affiliez-vous à la fonction Lambda et autorisez l'accès en lecture seule.

Erreur : échec <deployment-id>du déploiement du type NewDeployment pour le groupe <group-id>Erreur : échec du démarrage du processus : container_linux.go:259 : le démarrage du processus conteneur a provoqué « process_linux.go:250 : running exec setns process for init caused \" wait : no child processes \ "».

Solution : vous pouvez voir cette erreur lorsque le déploiement échoue. Retentez le déploiement.

<host-prefix>Erreur : [WARN] -MQTT [client] compose tcp : lookup -ats.iot. <region>.amazonaws.com : aucun hôte de ce type... [ERREUR] -Erreur de déploiement de Greengrass : impossible de signaler l'état du déploiement au cloud... net/http : demande annulée en attendant la connexion (Client.Timeout dépassé pendant l'attente des en-têtes)

Solution : vous pouvez voir cette erreur si vous utilisez `systemd-resolved`, ce qui active le paramètre `DNSSEC` par défaut. Par conséquent, de nombreux domaines publics ne sont pas reconnus. Les tentatives d'atteindre le point de terminaison AWS IoT Greengrass ne parviennent pas à trouver l'hôte. Vos déploiements restent donc à l'état `In Progress`.

Vous pouvez utiliser les commandes et la sortie suivantes pour tester ce problème. Remplacez l'espace réservé à *`la région`* dans les points de terminaison par votre. Région 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
```

Une solution possible consiste à désactiver `DNSSEC`. Lorsque `DNSSEC` a pour valeur `false`, les recherches DNS ne sont pas validées par `DNSSEC`. Pour plus d'informations, consultez ce problème connu pour `systemd`.

1. Ajoutez `DNSSEC=false` à `/etc/systemd/resolved.conf`.
2. Redémarrez `systemd-resolved`.

Pour plus d'informations sur `resolved.conf` et `DNSSEC`, exécutez `man resolved.conf` dans votre terminal.

Problèmes de création de groupe ou de fonction

Utilisez les informations suivantes pour résoudre les problèmes liés à la création d'un AWS IoT Greengrass groupe ou d'une fonction Greengrass Lambda.

Problèmes

Erreur : votre configuration « IsolationMode » pour le groupe n'est pas valide.

Solution : cette erreur se produit lorsque la valeur `IsolationMode` dans la section `DefaultConfig` de `function-definition-version` n'est pas prise en charge. Les valeurs prises en charge sont `GreengrassContainer` et `NoContainer`.

Erreur : votre configuration « IsolationMode » pour la fonction avec arn <function-arn>n'est pas valide.

Solution : cette erreur se produit lorsque la valeur `IsolationMode` de la section <arn-fonction> de `function-definition-version` n'est pas prise en charge. Les valeurs prises en charge sont `GreengrassContainer` et `NoContainer`.

Erreur : MemorySize la configuration de la fonction avec arn <function-arn>n'est pas autorisée dans IsolationMode =NoContainer.

Solution : Cette erreur se produit lorsque vous spécifiez une `MemorySize` valeur et que vous choisissez de l'exécuter sans conteneurisation. Les fonctions Lambda exécutées sans conteneurisation ne peuvent pas être soumises à des limites de mémoire. Vous pouvez soit supprimer la limite, soit modifier la fonction Lambda pour qu'elle s'exécute dans un AWS IoT Greengrass conteneur.

Erreur : la configuration Access Sysfs pour la fonction avec arn <function-arn>n'est pas autorisée dans IsolationMode =. NoContainer

Solution : Cette erreur se produit lorsque vous spécifiez `true` pour `AccessSysfs` et que vous choisissez de l'exécuter sans conteneurisation. Les fonctions Lambda exécutées sans conteneurisation doivent avoir leur code mis à jour pour accéder directement au système de fichiers et ne peuvent pas être utilisées. `AccessSysfs` Vous pouvez soit spécifier la valeur `false` for, `AccessSysfs` soit modifier la fonction Lambda pour qu'elle s'exécute dans un AWS IoT Greengrass conteneur.

Erreur : MemorySize la configuration de la fonction avec arn <function-arn>est requise dans IsolationMode =GreengrassContainer.

Solution : Cette erreur se produit parce que vous n'avez pas spécifié de `MemorySize` limite pour une fonction Lambda que vous exécutez dans un AWS IoT Greengrass conteneur. Spécifiez une valeur `MemorySize` pour résoudre l'erreur.

Erreur : <function-arn>La fonction fait référence à une ressource <resource-type>dont le type n'est pas autorisé dans IsolationMode =NoContainer.

Solution : Vous ne pouvez pas accéder aux types de `S3_Object.Generic_Archive` ressources `Local.Device` `Local.Volume` `ML_Model.SageMaker.JobML_Model.S3_Object`,, ou lorsque vous exécutez une fonction Lambda sans conteneurisation. Si vous avez besoin de ces

types de ressources, vous devez les exécuter dans un conteneur AWS IoT Greengrass. Vous pouvez également accéder directement aux appareils locaux lorsque vous les exécutez sans conteneurisation en modifiant le code de votre fonction Lambda.

Erreur : la configuration d'Execution pour la fonction ayant l'arn <arn-fonction> n'est pas autorisée.

Solution : Cette erreur se produit lorsque vous créez une fonction Lambda système avec `GGIPDetector` ou `GGCloudSpooler` que vous avez spécifié `IsolationMode` ou `RunAs` configuré. Vous devez omettre les `Execution` paramètres de cette fonction Lambda du système.

Problèmes de détection

Utilisez les informations suivantes pour résoudre les problèmes liés au service de découverte AWS IoT Greengrass.

Problèmes

- Erreur : l'appareil appartient à un trop grand nombre de groupes, les appareils ne peuvent pas être dans plus de 10 groupes

Erreur : l'appareil appartient à un trop grand nombre de groupes, les appareils ne peuvent pas être dans plus de 10 groupes

Solution : il s'agit d'une limitation connue. Un appareil client peut appartenir à un maximum de 10 groupes.

Problèmes liés aux ressources de machine learning

Utilisez les informations suivantes pour résoudre les problèmes liés aux ressources de Machine Learning.

Problèmes

- InvalidML ModelOwner - GroupOwnerSetting est fourni dans la ressource du modèle ML, mais n' GroupOwner GroupPermission est pas présent
- NoContainer La fonction ne peut pas configurer les autorisations lors de l'attachement de ressources Machine Learning. <function-arn>fait référence à une ressource d'apprentissage automatique <resource-id>avec autorisation <ro/rw> dans la politique d'accès aux ressources.
- <function-arn>La fonction fait référence à une ressource Machine Learning dont l'<resource-id>autorisation est manquante à la fois dans la ressource ResourceAccessPolicy et dans la ressource OwnerSetting.
- <function-arn>La fonction fait référence à la ressource Machine Learning <resource-id>avec l'autorisation \ "rw \ », tandis que le paramètre du propriétaire de la ressource autorise GroupPermission uniquement \ "ro \ ».
- NoContainer La fonction <function-arn>fait référence aux ressources du chemin de destination imbriqué.
- La fonction Lambda <function-arn> accède à la ressource <resource-id> en partageant le même identifiant de propriétaire de groupe

InvalidML ModelOwner - GroupOwnerSetting est fourni dans la ressource du modèle ML, mais n' GroupOwner GroupPermission est pas présent

Solution : vous recevez cette erreur si une ressource d'apprentissage automatique contient l'ResourceDownloadOwnerSettingobjet mais que la `GroupPermission` propriété `GroupOwner` ou la propriété requise n'est pas définie. Pour résoudre ce problème, définissez la propriété manquante.

NoContainer La fonction ne peut pas configurer les autorisations lors de l'attachement de ressources Machine Learning. <function-arn>fait référence à une ressource d'apprentissage automatique <resource-id>avec autorisation <ro/rw> dans la politique d'accès aux ressources.

Solution : vous recevez cette erreur si une fonction Lambda non conteneurisée spécifie des autorisations au niveau de la fonction pour une ressource de machine learning. Les fonctions non

conteneurisées doivent hériter des autorisations du propriétaire de la ressource définies sur la ressource de Machine Learning. Pour résoudre ce problème, choisissez d'hériter des autorisations du propriétaire des ressources (console) ou de supprimer les autorisations de la politique d'accès aux ressources (API) de la fonction Lambda.

<function-arn>La fonction fait référence à une ressource Machine Learning dont l'<resource-id>autorisation est manquante à la fois dans la ressource ResourceAccessPolicy et dans la ressource OwnerSetting.

Solution : vous recevez cette erreur si les autorisations d'accès à la ressource d'apprentissage automatique ne sont pas configurées pour la fonction Lambda attachée ou pour la ressource. Pour résoudre ce problème, configurez les autorisations dans la ResourceAccessPolicypropriété de la fonction Lambda ou dans la OwnerSettingpropriété de la ressource.

<function-arn>La fonction fait référence à la ressource Machine Learning <resource-id>avec l'autorisation \ "rw \ », tandis que le paramètre du propriétaire de la ressource autorise GroupPermission uniquement \ "ro \ ».

Solution : vous recevez cette erreur si les autorisations d'accès définies pour la fonction Lambda associée dépassent les autorisations du propriétaire de la ressource définies pour la ressource d'apprentissage automatique. Pour résoudre ce problème, définissez des autorisations plus restrictives pour la fonction Lambda ou des autorisations moins restrictives pour le propriétaire de la ressource.

NoContainer La fonction <function-arn>fait référence aux ressources du chemin de destination imbriqué.

Solution : vous recevez cette erreur si plusieurs ressources de machine learning associées à une fonction Lambda non conteneurisée utilisent le même chemin de destination ou un chemin de destination imbriqué. Pour résoudre ce problème, spécifiez des chemins de destination distincts pour les ressources.

La fonction Lambda <function-arn> accède à la ressource <resource-id> en partageant le même identifiant de propriétaire de groupe

Solution : vous recevez cette erreur `runtime.log` si le même groupe de système d'exploitation est spécifié comme identité Run as de la fonction Lambda et propriétaire de la ressource pour une ressource d'apprentissage automatique, mais que la ressource n'est pas attachée à la fonction Lambda. Cette configuration donne à la fonction Lambda des autorisations implicites qu'elle peut utiliser pour accéder à la ressource sans AWS IoT Greengrass autorisation.

Pour résoudre ce problème, utilisez un autre groupe de systèmes d'exploitation pour l'une des propriétés ou associez la ressource d'apprentissage automatique à la fonction Lambda.

Problèmes AWS IoT Greengrass Core dans Docker

Utilisez les informations suivantes pour résoudre les problèmes liés à l'exécution d'un AWS IoT Greengrass noyau dans un conteneur Docker.

Problèmes

- Erreur : options inconnues : -no-include-email.
- Avertissement : IPv4 est désactivé. La mise en réseau ne fonctionnera pas.
- Erreur : Un pare-feu bloque le partage de fichiers entre les fenêtres et les conteneurs.
- Erreur : une erreur s'est produite (AccessDeniedException) lors de l'appel de l' GetAuthorizationToken opération : L'utilisateur : arn:aws:iam : ::user/ <account-id><user-name>n'est pas autorisé à effectuer : ecr : on resource : * GetAuthorizationToken
- Erreur : Impossible de créer un conteneur pour le service greengrass : conflit. Le nom du conteneur «/aws-iot-greengrass» est déjà utilisé.
- Erreur : [FATAL] - Échec de la réinitialisation de l'espace de noms de montage du thread en raison d'une erreur inattendue : « opération non autorisée ». Pour maintenir la cohérence, GGC tombe en panne et doit être redémarré manuellement.

Erreur : options inconnues : -no-include-email.

Solution : cette erreur peut se produire lorsque vous exécutez la commande `aws ecr get-login`. Assurez-vous que vous disposez de la dernière version de l'interface de ligne de commande AWS CLI installée (par exemple, exécutez : `pip install awscli --upgrade --user`). Si vous utilisez Windows et avez installé l'interface de ligne de commande à l'aide du programme d'installation MSI, vous devez répéter le processus d'installation. Pour plus d'informations, consultez la section Installation du AWS Command Line Interface sous Microsoft Windows dans le Guide de AWS Command Line Interface l'utilisateur.

Avertissement : IPv4 est désactivé. La mise en réseau ne fonctionnera pas.

Solution : vous pouvez recevoir cet avertissement ou un message similaire lors de l'exécution d'AWS IoT Greengrass sur un ordinateur Linux. Activez le réacheminement réseau IPv4 comme décrit dans cette étape. Le déploiement cloud AWS IoT Greengrass et les communications MQTT ne fonctionnent pas lorsque le réacheminement IPv4 n'est pas activé. Pour plus d'informations, consultez Configurer les paramètres du noyau dans un espace de noms (sysctls) lors de l'exécution dans la documentation Docker.

Erreur : Un pare-feu bloque le partage de fichiers entre les fenêtres et les conteneurs.

Solution : vous pouvez recevoir cette erreur ou un message `Firewall Detected` lors de l'exécution de Docker sur un ordinateur Windows. Ce problème peut également survenir si vous êtes connecté à un réseau privé virtuel (VPN) et que vos paramètres réseau empêchent le montage du lecteur partagé. Dans ce cas, désactivez le VPN et réexécutez le conteneur Docker.

Erreur : une erreur s'est produite (AccessDeniedException) lors de l'appel de l' GetAuthorizationToken opération : L'utilisateur : arn:aws:iam : ::user/ <account-id><user-name>n'est pas autorisé à effectuer : ecr : on resource : * GetAuthorizationToken

Vous pouvez recevoir cette erreur lors de l'exécution de la `aws ecr get-login-password` commande si vous ne disposez pas des autorisations suffisantes pour accéder à un référentiel Amazon ECR. Pour plus d'informations, consultez les exemples de politiques relatives aux référentiels Amazon ECR et l'accès à un référentiel Amazon ECR dans le guide de l'utilisateur Amazon ECR.

Erreur : Impossible de créer un conteneur pour le service greengrass : conflit. Le nom du conteneur «/aws-iot-greengrass» est déjà utilisé.

Solution : cela peut se produire lorsque le nom du conteneur est utilisé par un conteneur plus ancien. Pour résoudre ce problème, exécutez la commande suivante afin de supprimer l'ancien conteneur Docker :

```
docker rm -f $(docker ps -a -q -f "name=aws-iot-greengrass")
```

Erreur : [FATAL] - Échec de la réinitialisation de l'espace de noms de montage du thread en raison d'une erreur inattendue : « opération non autorisée ». Pour maintenir la cohérence, GGC tombe en panne et doit être redémarré manuellement.

Solution : Cette erreur `runtime.log` peut se produire lorsque vous essayez de déployer une fonction `GreengrassContainer` Lambda sur un AWS IoT Greengrass cœur exécuté dans un conteneur Docker. Actuellement, seules les fonctions `NoContainer` Lambda peuvent être déployées sur un conteneur Greengrass Docker.

Pour résoudre ce problème, assurez-vous que toutes les fonctions Lambda sont en `NoContainer` mode et lancez un nouveau déploiement. Ensuite, lors du démarrage du conteneur, ne montez pas

le `deployment` répertoire existant sur le conteneur Docker AWS IoT Greengrass principal. Au lieu de cela, créez un répertoire `deployment` vide à sa place et liez-le ou montez-le dans le conteneur Docker. Cela permet au nouveau conteneur Docker de recevoir le dernier déploiement avec des fonctions Lambda exécutées en `NoContainer` mode.

Pour plus d'informations, consultez the section called "Exécuter AWS IoT Greengrass dans un conteneur Docker".

Résolution des problèmes liés aux journaux

Vous pouvez configurer les paramètres de journalisation pour un groupe Greengrass, par exemple s'il faut envoyer les CloudWatch journaux à Logs, les stocker sur le système de fichiers local, ou les deux. Pour obtenir des informations détaillées lors de la résolution des problèmes, vous pouvez temporairement définir le niveau de journalisation sur `DEBUG`. Les modifications apportées aux paramètres de journalisation prennent effet lorsque vous déployez le groupe. Pour plus d'informations, consultez the section called "Configurer la journalisation pour AWS IoT Greengrass".

Sur le système de fichiers local, AWS IoT Greengrass stocke les journaux dans les emplacements suivants. La lecture des journaux sur le système de fichiers requiert des autorisations racine.

greengrass-root`/ggc/var/log/crash.log`

Affiche les messages générés lorsqu'un AWS IoT Greengrass noyau tombe en panne.

greengrass-root`/ggc/var/log/system/runtime.log`

Affiche les messages sur le composant qui est en échec.

greengrass-root`/ggc/var/log/system/`

Contient tous les journaux des composants système AWS IoT Greengrass, tels que le gestionnaire de certificats et le gestionnaire de connexions. En utilisant les messages affichés dans `ggc/var/log/system/` et `ggc/var/log/system/runtime.log`, vous devez être en mesure d'identifier les erreurs qui se sont produites dans les composants du système AWS IoT Greengrass.

greengrass-root`/ggc/var/log/system/localwatch/`

Contient les journaux du AWS IoT Greengrass composant qui gère le téléchargement des journaux Greengrass vers Logs CloudWatch . Si vous ne pouvez pas consulter les connexions à Greengrass CloudWatch, vous pouvez utiliser ces journaux pour résoudre les problèmes.

greengrass-root/ggc/var/log/user/

Contient tous les journaux des fonctions Lambda définies par l'utilisateur. Consultez ce dossier pour trouver les messages d'erreur provenant de vos fonctions Lambda locales.

> **Note**
>
> Par défaut, *greengrass_root* est le répertoire /greengrass. Si un répertoire en écriture est configuré, les journaux se trouvent dans ce répertoire.

Si les journaux sont configurés pour être stockés dans le cloud, utilisez CloudWatch Logs pour afficher les messages des journaux. crash.logse trouve uniquement dans les journaux du système de fichiers sur le périphérique AWS IoT Greengrass principal.

S'il AWS IoT est configuré pour écrire des journaux dans CloudWatch, vérifiez-les si des erreurs de connexion se produisent lorsque des composants du système tentent de se connecter àAWS IoT.

Pour plus d'informations sur la journalisation AWS IoT Greengrass, consultez the section called “Surveillance avec les journaux AWS IoT Greengrass”.

> **Note**
>
> Les journaux relatifs aux logiciels AWS IoT Greengrass Core v1.0 sont stockés sous le répertoire *greengrass-root*/var/log.

Résolution des problèmes de stockage

Lorsque le stockage de fichiers local est plein, il se peut que certains composants commencent à tomber en panne :

- Les mises à jour locales de Shadow ne se produisent pas.
- Les nouveaux certificats de serveur MQTT AWS IoT Greengrass principaux ne peuvent pas être téléchargés localement.
- Les déploiements échouent.

Vous devez toujours connaître la quantité d'espace libre disponible en local. Vous pouvez calculer l'espace libre en fonction de la taille des fonctions Lambda déployées, de la configuration de journalisation (voirthe section called “Résolution des problèmes liés aux journaux”) et du nombre d'ombres stockées localement.

Messages de résolution des problèmes

Tous les messages envoyés localement dans AWS IoT Greengrass sont envoyés avec QoS 0. Par défaut, AWS IoT Greengrass stocke les messages dans une file d'attente en mémoire. Par conséquent, les messages non traités sont perdus lorsque le noyau Greengrass redémarre (par exemple après un déploiement de groupe ou un redémarrage de l'appareil). Cependant, vous pouvez configurer AWS IoT Greengrass (v1.6 ou version ultérieure) pour mettre les messages en cache dans le système de fichiers afin qu'ils persistent pendant les redémarrages principaux. Vous pouvez également configurer la taille de la file d'attente. Si vous configurez une taille de file d'attente, assurez-vous qu'elle est supérieure ou égale à 262 144 octets (256 Ko). Sinon, AWS IoT Greengrass risque de ne pas démarrer correctement. Pour plus d’informations, consultez the section called “File d'attente de messages MQTT”.

> **Note**
>
> Lorsque vous utilisez la file d'attente en mémoire par défaut, nous vous recommandons de déployer des groupes ou de redémarrer l'appareil lorsque l'interruption de service est la moins importante.

Vous pouvez également configurer l'appareil principal (noyau) pour établir des sessions persistantes avec AWS IoT. Cela permet au noyau de recevoir des messages envoyés AWS Cloud alors que le noyau est hors ligne. Pour plus d’informations, consultez the section called “Sessions persistantes MQTT avec AWS IoT Core”.

Dépannage des problèmes de temporisation de la synchronisation shadow

En cas de retard important dans la communication entre un appareil principal Greengrass et le cloud, la synchronisation shadow peut échouer en raison d'un délai d'expiration. Dans ce cas, vous devriez voir des entrées de journal similaires à ce qui suit :

```
[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 []}"
```

Pour remédier à ce problème, vous pouvez configurer la durée pendant laquelle votre appareil principal attend une réponse de l'hôte. Ouvrez le fichier config.json dans *greengrass-root*/config et ajoutez un champ `system.shadowSyncTimeout` avec une valeur de délai d'attente en secondes. Par exemple :

```
{
  "system": {
    "shadowSyncTimeout": 10
  },
  "coreThing": {
    "caPath": "root-ca.pem",
    "certPath": "cloud.pem.crt",
    "keyPath": "cloud.pem.key",
    ...
  },
  ...
}
```

La valeur par défaut est de 5 secondes si aucune valeur `shadowSyncTimeout` n'est spécifiée dans `config.json`.

> **Note**
>
> Pour AWS IoT Greengrass Core v1.6 et versions antérieures, la valeur par défaut de l'élément `shadowSyncTimeout` est de 1 seconde.

Consultez AWS re:Post

Si vous ne parvenez pas à résoudre votre problème à l'aide des informations de résolution de problèmes contenues dans cette rubrique, vous pouvez rechercher des problèmes connexes Résolution des problèmes ou consulter le AWS IoT Greengrasstag sur AWS Re:post ou publier une nouvelle question. Les membres de l'AWS IoT Greengrasséquipe surveillent activement AWS Re:post.

Historique du document pour AWS IoT Greengrass

Le tableau suivant décrit les modifications importantes apportées au guide du AWS IoT Greengrass développeur après juin 2018. Pour recevoir les notifications de mise à jour de cette documentation, abonnez-vous à un flux RSS.

Modification	Description	Date
Mise à jour indiquant la fin du support pour Snap v1.11.x	Mise à jour des informations de fin de support pour AWS IoT Greengrass Core v 1.11.x Snap sur snapcraft.io.	22 septembre 2023
Fin du support pour Snap v1.11.x	Ajout d'informations de fin de support pour AWS IoT Greengrass Core v 1.11.x Snap sur snapcraft.io.	19 septembre 2023
Images Docker pour v1.11.6 AWS IoT Greengrass	Les images Docker pour le logiciel AWS IoT Greengrass Core v1.11.6 sont disponibles sur Amazon Elastic Container Registry (Amazon ECR) et Docker Hub. Nous vous recommandons de toujours utiliser la dernière version.	12 avril 2022
AWS IoT Device Tester (IDT) pour AWS IoT Greengrass V1 la dépréciation	IDT pour AWS IoT Greengras s V1 ne générera plus de rapports de qualification signés.	4 avril 2022
Support mis à jour pour AWS IoT Device Tester pour AWS IoT Greengrass	IDT pour la AWS IoT Greengrass version 4.4.1 prend désormais en charge l'utilisation de la version logicielle AWS IoT Greengras	24 mars 2022

	s principale v1.11.6 pour la qualification des appareils.	
AWS IoT Greengrass version 1.11.6 publiée	La version 1.11.6 du logiciel AWS IoT Greengrass Core est disponible. Cette version contient des améliorations de performances et des correctif s de bogues. Nous vous recommandons de toujours utiliser la dernière version.	24 mars 2022
Sortie de la version 12 du SiteWise connecteur IoT	La version 12 du SiteWise connecteur IoT est disponibl e. Cette version contient des corrections de bogues.	23 décembre 2021
Images Docker pour AWS IoT Greengrass v1.11.5 et v1.10.5	Les images Docker pour les logiciels AWS IoT Greengras s Core v1.11.5 et v1.10.5 sont disponibles sur Amazon Elastic Container Registry (Amazon ECR) et Docker Hub. Nous vous recommandons de toujours utiliser la dernière version.	22 décembre 2021
AWS IoT Greengrass V1 politique de maintenance	La politique de AWS IoT Greengrass V1 maintenance définit les différents niveaux de maintenance et de mise à jour pour le AWS IoT Greengras s V1 service et le logiciel de AWS IoT Greengrass base v1.x.	20 décembre 2021

AWS IoT Sortie de la version 4.4.1 de Device Tester	IDT pour la AWS IoT Greengrass version 4.4.1 est désormais disponible. Cette version inclut la suite de AWS IoT Greengrass qualification (GGQ) v1.3.1 et prend en charge l'utilisation des versions logicielles de AWS IoT Greengrass base v1.11.5 et v1.10.5 pour la qualification des appareils.	20 décembre 2021
AWS IoT Greengrass versions 1.11.5 et 1.10.5 publiées	Les versions 1.11.5 et 1.10.5 du logiciel AWS IoT Greengrass Core sont disponibles. Ces versions contiennent des améliorations de performances et des corrections de bogues. Nous vous recommandons de toujours utiliser la dernière version.	12 décembre 2021

Images Docker AWS IoT Greengrass v1.11.4 et v1.10.4 republiées	Les images Docker des versions 1.11.4 et 1.10.4 du logiciel AWS IoT Greengras s Core ont été republiées sur Amazon Elastic Container Registry (Amazon ECR) et Docker Hub afin de corriger les bogues. BusyBox Pour utiliser les dernières images Docker, utilisez les `1.10.4-1` balises `1.11.4-1` or. Pour plus d'informations sur les balises disponibles, consultez amazon/ aws-iot-greengrass dans Docker Hub.	8 décembre 2021
CloudWatch Le connecteur Metrics prend en charge les horodatages dupliqués dans les données d'entrée	Vous pouvez désormais envoyer des données d'entrée avec des horodatages dupliqués à ce connecteur.	19 novembre 2021
Mise à jour de la prévention du problème de l'adjoint confus entre services	AWS IoT Greengrass prend en charge l'utilisa tion des clés de contexte de condition `aws:SourceAccount` globale `aws:SourceArn` et des clés de contexte dans les politiques de ressources IAM afin d'éviter le problème de confusion des adjoints.	1er novembre 2021

Images Docker pour v1.11.4 AWS IoT Greengrass	Les images Docker pour le logiciel AWS IoT Greengrass Core v1.11.4 sont disponibles sur Amazon Elastic Container Registry (Amazon ECR) et Docker Hub. Nous vous recommandons de toujours utiliser la dernière version.	24 août 2021
AWS IoT Greengrass Snap v1.11.4 publié	La version 1.11.4 du AWS IoT Greengrass snap est disponible. Nous vous recommandons de toujours utiliser la dernière version.	20 août 2021
Support mis à jour pour AWS IoT Device Tester pour AWS IoT Greengrass	IDT pour la AWS IoT Greengrass version 4.1.0 prend désormais en charge l'utilisation de la version logicielle AWS IoT Greengrass principale v1.11.4 pour la qualification des appareils.	18 août 2021
AWS IoT Greengrass version 1.11.4 publiée	La version 1.11.4 du logiciel AWS IoT Greengrass Core est disponible. Cette version corrige un problème lié au gestionnaire de flux qui empêchait les mises à niveau vers la v1.11.3 à partir d'une version antérieure du logiciel AWS IoT Greengrass Core. Nous vous recommandons de toujours utiliser la dernière version.	17 août 2021

Points de terminaison VPC () AWS PrivateLink	AWS IoT Greengrass prend désormais en charge les points de terminaison VPC d'interface (AWS PrivateLink) pour le AWS IoT Greengrass plan de contrôle. Vous pouvez établir une connexion privée entre votre VPC et le plan de AWS IoT Greengrass contrôle.	16 août 2021
AWS IoT Sortie de la version 4.1.0 de Device Tester	La version 4.1.0 de AWS IoT Device Tester for AWS IoT Greengrass est disponible. Cette version prend en charge l'utilisation des versions logicielles AWS IoT Greengras s principales 1.11.3 et 1.10.4 pour la qualification des appareils.	23 Juin 2021
AWS IoT Greengrass Snap v1.11.3 publié	La version 1.11.3 du AWS IoT Greengrass snap contient des améliorations de performances et des corrections de bugs. Nous vous recommandons de toujours utiliser la dernière version.	15 juin 2021

Publication des images Docker pour AWS IoT Greengrass v1.11.3 et v1.10.4	Les images Docker pour les logiciels AWS IoT Greengrass Core v1.11.3 et v1.10.4 sont disponibles sur Amazon Elastic Container Registry (Amazon ECR) et Docker Hub. Ces versions de AWS IoT Greengrass Core contiennent des améliorations de performances et des corrections de bogues. Nous vous recommandons de toujours utiliser la dernière version.	15 juin 2021
AWS IoT Greengrass version 1.11.3 publiée	La version 1.11.3 du logiciel AWS IoT Greengrass Core est disponible. Cette version contient des améliorations de performances et des correctifs de bogues. Nous vous recommandons de toujours utiliser la dernière version.	14 juin 2021
AWS IoT Greengrass version 1.10.4 publiée	La version 1.10.4 du logiciel AWS IoT Greengrass Core est disponible. Cette version contient des améliorations de performances et des correctifs de bogues. Nous vous recommandons de toujours utiliser la dernière version.	14 juin 2021

Sortie de la version 2 de l'adaptateur de protocole Modbus-TCP	La version 2 du connecteur adaptateur de protocole Modbus-TCP est disponible. Cette version a ajouté la prise en charge des chaînes sources codées en ASCII, UTF8 et ISO8859.	24 mai 2021
Sortie de la version 7 du connecteur de déploiement d'applications Docker	La version 7 du connecteur de déploiement d'applications Greengrass Docker est disponible.	5 avril 2021
AWS IoT Greengrass version 1.11.1 publiée	La version 1.11.1 du logiciel AWS IoT Greengrass Core est disponible. Cette version contient des améliorations de performances et des correctifs de bogues. Nous vous recommandons de toujours utiliser la dernière version.	29 mars 2021
AWS IoT Sortie de la version 4.0.2 de Device Tester	La version 4.0.2 de AWS IoT Device Tester for AWS IoT Greengrass est disponible. Cette version remplace IDT v4.0.0 et ajoute le support pour la version 1.11.1 du logiciel Core. AWS IoT Greengrass Cela résout également un problème en raison duquel IDT masquait les erreurs HSI (Hardware Security Integration).	29 mars 2021

Sortie de la version 11 du SiteWise connecteur IoT	La version 11 du SiteWise connecteur IoT est disponible. Cela lance la prise en charge des chaînes contenant des caractères masqués ou non imprimables. Cette version inclut également des améliorations générales des performances et des corrections de bogues.	24 mars 2021
Snap AWS IoT Greengrass v1.11.0 republié	AWS IoT Greengrass La version 1.11.0 de snap a été republiée sur Snapcraft pour corriger des bogues et corriger un éventuel crash de l'application lors de l'utilisation de l'interpréteur Python. AWS IoT Greengrass ne fournit pas de snaps pour les versions logicielles 1.10 et 1.9.	19 mars 2021
Support mis à jour pour AWS IoT Device Tester pour AWS IoT Greengrass	IDT pour la AWS IoT Greengrass version 4.0.0 prend désormais en charge l'utilisation de la version logicielle AWS IoT Greengrass principale v1.10.3 pour la qualification des appareils.	18 mars 2021
Snap AWS IoT Greengrass v1.8.4 republié	AWS IoT Greengrass La version 1.8.4 de snap a été republiée sur Snapcraft pour corriger des bogues et corriger un éventuel crash de l'application lors de l'utilisation de l'interpréteur Python.	15 mars 2021

Image Docker AWS IoT Greengrass v1.11.0 republiée pour ARMv7L	L'image Docker de la version 1.11.0 du logiciel AWS IoT Greengrass Core pour la plateforme ARMv7L a été republiée sur Amazon Elastic Container Registry (Amazon ECR) et Docker Hub afin de corriger des bogues et d'éviter un éventuel crash de l'application lors de l'utilisation de l'interpréteur Python.	8 mars 2021
AWS IoT Greengrass Images Docker v1.10.3 publiées	Les images Docker pour la version 1.10.3 du logiciel AWS IoT Greengrass Core sont désormais disponibles sur Amazon Elastic Container Registry (Amazon ECR) et Docker Hub.	8 mars 2021
Images Docker republiées dans les AWS IoT Greengrass versions v1.11.0 et v1.9.4	Les images Docker des versions 1.11.0 et 1.9.4 du logiciel AWS IoT Greengrass Core ont été republiées sur Amazon Elastic Container Registry (Amazon ECR) et Docker Hub afin de corriger des bogues et d'éviter un éventuel crash de l'application lors de l'utilisation de l'interpréteur Python. Les images Docker pour ARMv7L n'ont pas été republiées pour le moment.	26 février 2021

AWS IoT Greengrass version 1.10.3 publiée	La version 1.10.3 du logiciel AWS IoT Greengrass Core est disponible. Cette version ajoute la propriété de configuration du noyau `systemComponentAuthTimeout` et contient des améliorations de performances et des corrections de bugs. Nous vous recommandons de toujours utiliser la dernière version.	24 février 2021
Sortie de la version 10 du SiteWise connecteur IoT	La version 10 du SiteWise connecteur IoT est disponible. Cette version résout les problèmes de stabilité du StreamManager client en cas de perte de connexion et améliore la gestion des valeurs OPC-UA en l'absence de `SourceTimestamp` a. Utilisez le SiteWise connecteur IoT pour envoyer les données des appareils et équipements locaux aux propriétés des actifs dans l'IoT SiteWise.	22 janvier 2021

Sortie de la version 9 du SiteWise connecteur IoT	La version 9 du SiteWise connecteur IoT est disponible. Cela permet de prendre en charge les destinations de `StreamManager` streaming Greengrass personnalisées, le deadbanding OPC-UA, le mode de scan personnalisé et le taux de scan personnalisé. Cela inclut également l'amélioration des performances lors des mises à jour de configuration effectuées à partir de la SiteWise passerelle IoT. Utilisez le SiteWise connecteur IoT pour envoyer les données des appareils et équipements locaux aux propriétés des actifs dans l'IoT SiteWise.	15 décembre 2020
AWS IoT Sortie de la version 4.0.0 de Device Tester	La version 4.0.0 de AWS IoT Device Tester for AWS IoT Greengrass est disponible. Cette version vous permet d'utiliser IDT pour développer et exécuter vos suites de tests personnalisées pour la validation des appareils. Cela inclut également les applications IDT signées par code pour macOS et Windows.	15 décembre 2020

AWS IoT Greengrass snap v1.11	La version 1.11.0 du AWS IoT Greengrass snap prend en charge les fonctions Lambda non conteneurisées. Nous vous recommandons de toujours utiliser la dernière version.	6 décembre 2020
Sortie de la version 8 du SiteWise connecteur IoT	La version 8 du SiteWise connecteur IoT est disponible. Cette version améliore la stabilité lorsque le connecteur est confronté à une connectivité réseau intermittente. Utilisez le SiteWise connecteur IoT pour envoyer les données des appareils et équipements locaux aux propriétés des actifs dans l'IoT SiteWise.	19 novembre 2020
Le connecteur Kinesis Firehose prend en charge le mode sans conteneur	Vous pouvez utiliser le `IsolationMode` paramètre pour configurer le mode de conteneurisation du connecteur.	19 octobre 2020
Sortie de la version 6 du connecteur de déploiement d'applications Docker	La version 6 du connecteur de déploiement d'applications Greengrass Docker est disponible.	18 septembre 2020

AWS IoT Greengrass version 1.11.0 publiée	La version 1.11.0 du logiciel AWS IoT Greengrass Core est disponible. Cette version ajoute la fonctionnalité de télémétrie de l'état du système et une API de vérification de l'état locale. Le gestionnaire de flux peut désormais exporter des données vers Amazon Simple Storage Service (Amazon S3) et l'IoT. SiteWise Cette version contient également des améliorations de performances et des corrections de bogues. Nous vous recommandons de toujours utiliser la dernière version.	16 septembre 2020
Sortie de la version 7 du SiteWise connecteur IoT	La version 7 du SiteWise connecteur IoT est disponible. Cette version corrige un problème lié aux métriques de passerelle. Utilisez le SiteWise connecteur IoT pour envoyer les données des appareils et équipements locaux aux propriétés des actifs dans l'IoT SiteWise.	14 août 2020
ServiceNow MetricBase Les connecteurs Integration, Splunk Integration et Twilio Notifications prennent en charge le mode sans conteneur	Vous pouvez utiliser le `IsolationMode` paramètre pour configurer le mode de conteneurisation du connecteur.	30 juillet 2020

Le connecteur SNS prend en charge le mode sans conteneur	Vous pouvez utiliser le `IsolationMode` paramètre pour configurer le mode de conteneurisation du connecteur.	6 juillet 2020
CloudWatch Le connecteur Metrics prend en charge le mode sans conteneur	Vous pouvez utiliser le `IsolationMode` paramètre pour configurer le mode de conteneurisation du connecteur.	17 juin 2020
AWS IoT Greengrass version 1.10.2 publiée	La version 1.10.2 du logiciel AWS IoT Greengrass Core est disponible. Cette version ajoute la propriété de configuration du noyau `mqttOperationTimeout` et contient des améliorations de performances et des corrections de bugs. Nous vous recommandons de toujours utiliser la dernière version.	8 juin 2020
Les installateurs d'apprentissage automatique Tensorflow sont obsolètes	AWS IoT Greengrass Les installateurs d'apprentissage automatique préemballés de Tensorflow sont devenus obsolètes. Les exemples de machine learning ont été mis à niveau vers Python 3.7.	29 mai 2020

Le support du framework Chainer et les installateurs d'apprentissage automatique Greengrass sont obsolètes	AWS IoT Greengrass les programmes d'installation et de téléchargement de machine learning préemballés pour MXnet et DLR sont devenus obsolètes. La prise en charge du framework Chainer et des téléchargements associés est devenue obsolète.	4 mai 2020
Sortie de la version 6 du SiteWise connecteur IoT	La version 6 du SiteWise connecteur IoT est disponibl e. Cette version ajoute la prise en charge des CloudWatch métriques et de la découvert e automatique des nouvelles balises OPC-UA. Cela signifie que vous n'avez pas besoin de redémarrer votre passerell e lorsque les balises changent pour vos sources OPC-UA. Cette version du connecteu r nécessite le gestionnaire de flux et le logiciel AWS IoT Greengrass Core v1.10.0 ou supérieur. Utilisez le SiteWise connecteur IoT pour envoyer les données des appareils et équipements locaux aux propriétés des actifs dans l'IoT SiteWise.	29 avril 2020

Connecteurs mis à niveau vers Python 3.7	Les connecteurs qui prennent en charge l'environnement d'exécution Python ont été mis à niveau vers Python 3.7. Nous vous recommandons de mettre à jour les versions de connecteur de Python 2.7 vers Python 3.7.	29 avril 2020
La configuration de l'appareil Greengrass peut fonctionner en mode silencieux	Vous pouvez exécuter la configuration de l'appareil Greengrass en mode silencieux afin que le script ne vous demande aucune valeur.	27 avril 2020
Nouvelles images de base Docker	Vous pouvez télécharger des images AWS IoT Greengrass Docker basées sur des images de base d'Alpine Linux (x86_64, ARMv7L ou AArch64).	23 avril 2020
AWS IoT Greengrass version 1.10.1 publiée	La version 1.10.1 du logiciel AWS IoT Greengrass Core est disponible. Cette version contient des améliorations de performances et des correctifs de bogues. Nous vous recommandons de toujours utiliser la dernière version.	16 avril 2020
Nouveau chapitre sur la sécurité	AWS IoT Greengrass le contenu de sécurité a été réorganisé et de nouvelles informations ont été ajoutées.	30 mars 2020

Utilisez le gestionnaire de paquets APT pour installer AWS IoT Greengrass	Sur les distributions Linux basées sur Debian prises en charge, vous pouvez l'utiliser `apt` pour installer le logiciel AWS IoT Greengrass Core sur vos appareils.	26 février 2020
Sortie de la version 5 du SiteWise connecteur IoT	La version 5 du SiteWise connecteur IoT est disponible. Cette version corrige un problème de compatibilité avec le logiciel AWS IoT Greengrass Core v1.9.4. Utilisez le SiteWise connecteur IoT pour envoyer les données des appareils et équipements locaux aux propriétés des actifs dans l'IoT SiteWise.	12 février 2020
Nouveau script pour configurer rapidement un appareil principal	Vous pouvez utiliser la configuration de l'appareil Greengrass pour configurer votre appareil principal en quelques minutes. Supporte également AWS IoT Greengrass désormais les fonctions Lambda de Node.js 12.x.	20 décembre 2019

AWS IoT Greengrass version 1.10.0 publiée	La version 1.10.0 du logiciel AWS IoT Greengrass Core est disponible. Les nouvelles fonctionnalités de cette version incluent le gestionnaire de flux, la prise en charge des conteneurs avec le connecteur de déploiement d'applications Docker, les fonctions Lambda non conteneurisées permettant d'accéder aux ressources d'apprentissage automatique, la prise en charge des sessions persistantes MQTT AWS IoT avec et la prise en charge du trafic MQTT local sur un port spécifié.	25 novembre 2019
Support de console pour les notifications de déploiement	Utilisez la EventBridge console Amazon pour créer des règles d'événement qui se déclenchent lorsque les déploiements de votre groupe Greengrass changent d'état.	14 novembre 2019
AWS IoT Greengrass version 1.9.4 publiée	La version 1.9.4 du logiciel AWS IoT Greengrass Core est disponible. Cette version contient des améliorations de performances et des correctifs de bogues. En tant que bonne pratique, nous vous recommandons de toujours exécuter la dernière version.	17 octobre 2019

Support de console pour la gestion du rôle de service Greengrass	Utilisez les fonctionnalités nouvelles et améliorées de la AWS IoT console pour gérer votre rôle de service Greengrass.	4 octobre 2019
Support de console pour la gestion des balises au niveau du groupe	Vous pouvez créer, afficher et gérer les tags de vos groupes Greengrass dans la AWS IoT console.	23 septembre 2019
Nouveaux connecteurs d'apprentissage automatique	Utilisez le connecteur ML Feedback pour publier des entrées de modèle et des prédictions, et le connecteu r ML Object Detection pour exécuter un service local d'inférence de détection d'objet.	19 septembre 2019
AWS IoT Greengrass version 1.9.3 publiée	La version 1.9.3 du logiciel AWS IoT Greengrass Core est disponible. Cette version vous permet d'installer le logiciel AWS IoT Greengras s Core sur les distributions Raspbian sur les architectures ARMv6L, prend en charge les mises à jour OTA sur le port 443 avec ALPN et contient un correctif pour les charges utiles binaires envoyées depuis les fonctions Lambda de Python 2.7 vers d'autres fonctions Lambda.	12 septembre 2019

AWS IoT Greengrass version 1.8.4 publiée	La version 1.8.4 du logiciel AWS IoT Greengrass Core est disponible. Cette version contient des améliorations de performances et des correctif s de bogues. Si vous exécutez la version 1.8.x, nous vous recommandons de procéder à une mise à niveau vers la version 1.8.4 ou 1.9.3. Pour les versions antérieures, nous vous recommandons de procéder à une mise à niveau vers la version 1.9.3.	30 août 2019
AWS IoT Greengrass version 1.9.2 publiée avec le support de OpenWrt	La version 1.9.2 du logiciel AWS IoT Greengrass Core est disponible. Cette version permet d'installer le logiciel AWS IoT Greengrass Core sur des OpenWrt distributions avec des architectures Armv8 (AArch64) et ARMv7L.	20 juin 2019

AWS IoT Greengrass version 1.8.3 publiée	La version 1.8.3 du logiciel AWS IoT Greengrass Core est disponible. Cette version contient des améliorations de performances et des correctif s de bogues généraux. Si vous exécutez la version 1.8.x, nous vous recommandons de procéder à une mise à niveau vers la version 1.8.3 ou 1.9.2. Pour les versions antérieures, nous vous recommandons de procéder à une mise à niveau vers la version 1.9.2.	20 juin 2019
AWS IoT Greengrass version 1.9.1 publiée	La version 1.9.1 du logiciel AWS IoT Greengrass Core est disponible. Cette version contient une correction de bogue pour les messages provenant de messages AWS IoT contenant un caractère générique dans le sujet.	10 mai 2019

AWS IoT Greengrass version 1.8.2 publiée	La version 1.8.2 du logiciel AWS IoT Greengrass Core est disponible. Cette version contient des améliorations de performances et des correctifs de bogues généraux. Si vous exécutez la version 1.8.x, nous vous recommandons de procéder à une mise à niveau vers la version 1.8.2 ou 1.9.0. Pour les versions antérieures, nous vous recommandons de procéder à une mise à niveau vers la version 1.9.0.	2 mai 2019
AWS IoT Greengrass version 1.9.0 publiée	Nouvelles fonctionnalités : support des environnements d'exécution Lambda de Python 3.7 et Node.js 8.10, connexions MQTT optimisées et prise en charge des clés Elliptic Curve (EC) pour le serveur MQTT local.	1er mai 2019
AWS IoT Greengrass version 1.8.1 publiée	La version 1.8.1 du logiciel AWS IoT Greengrass Core est disponible. Cette version contient des améliorations de performances et des correctifs de bogues généraux. En tant que bonne pratique, nous vous recommandons de toujours exécuter la dernière version.	18 avril 2019

AWS IoT Greengrass snap disponible sur Snapcraft	Utilisez l'application AWS IoT Greengrass Snap Store pour concevoir, tester et déployer rapidement des logiciels sur des appareils Linux avec AWS IoT Greengrass.	1 avril 2019
Support pour un meilleur contrôle d'accès à l'aide d'autorisations basées sur des balises	Vous pouvez utiliser des balises dans les politiques AWS Identity and Access Management (IAM) pour contrôler l'accès à vos AWS IoT Greengrass ressources.	29 mars 2019
Lancement du connecteur IoT Analytics	Utilisez le connecteur IoT Analytics pour envoyer les données des appareils locaux aux AWS IoT Analytics canaux.	15 mars 2019
Support par lots dans le connecteur Kinesis Firehose	Le connecteur Kinesis Firehose prend en charge l'envoi d'enregistrements de données par lots à Amazon Data Firehose à un intervalle défini.	15 mars 2019
AWS CloudFormation soutien aux AWS IoT Greengrass ressources	Utilisez AWS CloudFormation des modèles pour créer et gérer AWS IoT Greengrass des ressources.	15 mars 2019

AWS IoT Greengrass version 1.8.0 publiée	Nouvelles fonctionnalités : identité d'accès par défaut configurable pour les fonctions Lambda, prise en charge du trafic HTTPS via le port 443 et identifiants clients nommés de manière prévisible pour les connexions MQTT avec lesquelles. AWS IoT	7 mars 2019
AWS IoT Greengrass versions 1.7.1 et 1.6.1 publiées	Les versions 1.7.1 et 1.6.1 du logiciel AWS IoT Greengras s Core sont disponibles. Ces versions nécessitent un noyau Linux version 3.17 ou ultérieur e. Nous recommandons aux clients qui exécutent n'importe quelle version du logiciel Greengrass Core d'effectuer immédiatement une mise à niveau vers la version 1.7.1.	11 février 2019
SageMaker Runtime d'apprent issage profond Neo	Le moteur d'apprentissage profond SageMaker Neo prend en charge les modèles d'apprentissage automatiq ue optimisés par le compilate ur d'apprentissage profond SageMaker Neo.	28 novembre 2018
Exécuter AWS IoT Greengrass dans un conteneur Docker	Vous pouvez exécuter AWS IoT Greengrass dans un conteneur Docker en configura nt votre groupe Greengras s pour qu'il fonctionne sans conteneurisation.	26 novembre 2018

AWS IoT Greengrass version 1.7.0 publiée	Nouvelles fonctionnalités : connecteurs Greengras s, gestionnaire de secrets locaux, paramètres d'isolati on et d'autorisation pour les fonctions Lambda, sécurité matérielle root of trust, connexion via ALPN ou proxy réseau et support de Raspbian Stretch.	26 novembre 2018
AWS IoT Greengrass téléchargements de logiciels	Les packages AWS IoT Greengrass Core Software, AWS IoT Greengrass Core SDK et AWS IoT Greengras s Machine Learning SDK peuvent être téléchargés via Amazon. CloudFront	26 novembre 2018
AWS IoT Testeur d'appareils pour AWS IoT Greengrass	Utilisez AWS IoT Device Tester AWS IoT Greengras s pour vérifier que l'archite cture de votre processeur, la configuration du noyau et les pilotes fonctionnent avec AWS IoT Greengrass.	26 novembre 2018
AWS CloudTrail journalis ation des appels AWS IoT Greengrass d'API	AWS IoT Greengrass est intégré à AWS CloudTrail un service qui fournit un enregistr ement des actions entrepris es par un utilisateur, un rôle ou un AWS service dans AWS IoT Greengrass.	29 octobre 2018

Support pour la TensorFlow v1.10.1 sur NVIDIA Jetson TX2	La bibliothèque TensorFlow précompilée pour NVIDIA Jetson TX2 qui AWS IoT Greengrass fournit utilise désormais la v1.10.1. TensorFlow Jetpack 3.3 et CUDA Toolkit 9.0 sont donc pris en charge.	18 octobre 2018
Support pour les ressources d'apprentissage automatique MXnet v1.2.1	AWS IoT Greengrass prend en charge les modèles d'apprentissage automatique entraînés à l'aide de MXnet v1.2.1.	29 août 2018
AWS IoT Greengrass version 1.6.0 publiée	Nouvelles fonctionnalités : exécutables Lambda, file d'attente de messages configurable, intervalle de tentatives de reconnexion configurable, ressources de volume sous /proc et répertoire d'écriture configurable.	26 juillet 2018

Mises à jour antérieures

Le tableau suivant décrit les modifications importantes apportées au Guide du AWS IoT Greengrass développeur avant juillet 2018.

Modification	Description	Date
AWS IoT Greengrass Version 1.5.0 publiée	Nouvelles fonctions : • Inférence d'apprentissage automatique locale à l'aide des modèles formés dans le cloud. Pour de plus amples informations, veuillez consulter Exécuter l'inférence de Machine Learning.	29 mars 2018

Modification	Description	Date
	• Les fonctions Lambda de Greengrass prennent en charge les données d'entrée binaires, en plus du JSON. Pour plus d'informations, consultez Versions AWS IoT Greengrass Core.	
AWS IoT Greengrass Version 1.3.0 publiée	Nouvelles fonctions : • Agent de mise à jour O ver-the-air (OTA) capable de gérer les tâches de mise à jour Greengrass déployées dans le cloud. Pour de plus amples informations, veuillez consulter Mises à jour OTA du logiciel AWS IoT Greengrass Core. • Accédez aux périphériques et aux ressources locaux à partir des fonctions de Greengrass Lambda. Pour de plus amples informations, veuillez consulter Accédez à des ressources locales avec des fonctions et des connecteurs Lambda.	27 novembre 2
AWS IoT Greengrass Version 1.1.0 publiée	Nouvelles fonctions : • Réinitialisez AWS IoT Greengrass les groupes déployés. Pour de plus amples informations, veuillez consulter Réinitialiser les déploiements. • Support des environnements d'exécution Lambda de Node.js 6.10 et Java 8, en plus de Python 2.7.	20 septembre 2
AWS IoT Greengrass Version 1.0.0 publiée	AWS IoT Greengrass est généralement disponible.	7 juin 2017