



Guide du développeur, version 2

AWS IoT Greengrass



AWS IoT Greengrass: Guide du développeur, version 2

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

Les marques et la présentation commerciale d'Amazon ne peuvent être utilisées en relation avec un produit ou un service qui n'est pas d'Amazon, d'une manière susceptible de créer une confusion parmi les clients, ou d'une manière qui dénigre ou discrédite Amazon. Toutes les autres marques commerciales qui ne sont pas la propriété d'Amazon appartiennent à leurs propriétaires respectifs, qui peuvent ou non être affiliés ou connectés à Amazon, ou sponsorisés par Amazon.

Table of Contents

Qu'est-ce que c'est AWS IoT Greengrass ?	1
Nouvelles fonctionnalités	1
Pour les nouveaux utilisateurs	2
Pour les utilisateurs existants	2
Comment AWS IoT Greengrass fonctionne	2
Concepts clés	3
Caractéristiques de AWS IoT Greengrass	5
Compatibilité des fonctionnalités de Greengrass	7
Choix de votre environnement d'exécution AWS IoT Greengrass Nucleus	16
Noyau de Greengrass	17
Greengrass Nucleus Lite	17
Nouveautés de la version 2	21
AWS IoT Greengrass Mise à jour logicielle Core v2.16.1	23
Mises à jour des composants publics	24
AWS IoT Greengrass Mise à jour logicielle Core v2.16.0	25
Mises à jour des composants publics	25
AWS IoT Greengrass Mise à jour logicielle Core v2.15.1	27
Mises à jour des composants publics	27
AWS IoT Greengrass Mise à jour logicielle Core v2.15.0	28
Mises à jour des composants publics	28
AWS IoT Greengrass Mise à jour logicielle Core v2.14.3	31
Mises à jour des composants publics	31
AWS IoT Greengrass Mise à jour logicielle Core v2.14.2	33
Mises à jour des composants publics	33
AWS IoT Greengrass Mise à jour logicielle Core v2.14.1	34
Mises à jour des composants publics	34
AWS IoT Greengrass Mise à jour logicielle Core v2.14.0	35
Mises à jour des composants publics	37
AWS IoT Greengrass Mise à jour logicielle Core v2.13.0	40
Mises à jour des composants publics	40
AWS IoT Greengrass Mise à jour logicielle Core v2.12.6	42
Mises à jour des composants publics	42
AWS IoT Greengrass Mise à jour logicielle Core v2.12.5	44
Mises à jour des composants publics	44

AWS IoT Greengrass Mise à jour logicielle Core v2.12.4	45
Mises à jour des composants publics	45
AWS IoT Greengrass Mise à jour logicielle Core v2.12.3	46
Mises à jour des composants publics	46
AWS IoT Greengrass Mise à jour logicielle Core v2.12.2	48
Mises à jour des composants publics	49
AWS IoT Greengrass Mise à jour logicielle Core v2.12.1	50
Mises à jour des composants publics	50
AWS IoT Greengrass Mise à jour logicielle Core v2.12.0	52
Mises à jour des composants publics	52
AWS IoT Greengrass Mise à jour logicielle Core v2.11.3	53
Mises à jour des composants publics	54
AWS IoT Greengrass Mise à jour logicielle Core v2.11.2	55
Mises à jour des composants publics	55
AWS IoT Greengrass Mise à jour logicielle Core v2.11.1	56
Mises à jour des composants publics	57
AWS IoT Greengrass Mise à jour logicielle Core v2.11.0	58
Mises à jour des composants publics	58
AWS IoT Greengrass Mise à jour logicielle Core v2.10.3	60
Mises à jour des composants publics	60
AWS IoT Greengrass Mise à jour logicielle Core v2.10.2	61
Mises à jour des composants publics	61
AWS IoT Greengrass Mise à jour logicielle Core v2.10.1	63
Mises à jour des composants publics	64
AWS IoT Greengrass Mise à jour logicielle Core v2.10.0	65
Mises à jour des composants publics	65
AWS IoT Greengrass Mise à jour logicielle Core v2.9.6	67
Mises à jour des composants publics	67
AWS IoT Greengrass Mise à jour logicielle Core v2.9.5	68
Mises à jour des composants publics	68
AWS IoT Greengrass Mise à jour logicielle Core v2.9.4	70
Mises à jour des composants publics	70
AWS IoT Greengrass Mise à jour logicielle Core v2.9.3	71
Mises à jour des composants publics	71
AWS IoT Greengrass Mise à jour logicielle Core v2.9.2	72
Mises à jour des composants publics	73

AWS IoT Greengrass Mise à jour logicielle Core v2.9.1	73
Mises à jour des composants publics	74
AWS IoT Greengrass Mise à jour logicielle Core v2.9.0	75
Mises à jour des composants publics	76
AWS IoT Greengrass Mise à jour logicielle Core v2.8.1	78
Mises à jour des composants publics	78
AWS IoT Greengrass Mise à jour logicielle Core v2.8.0	80
Mises à jour des composants publics	80
AWS IoT Greengrass Mise à jour logicielle Core v2.7.0	82
Mises à jour des composants publics	83
AWS IoT Greengrass Mise à jour logicielle Core v2.6.0	85
Mises à jour des composants publics	86
AWS IoT Greengrass Mise à jour logicielle Core v2.5.6	90
Mises à jour des composants publics	91
AWS IoT Greengrass Mise à jour logicielle Core v2.5.5	92
Mises à jour des composants publics	92
AWS IoT Greengrass Mise à jour logicielle Core v2.5.4	94
Mises à jour des composants publics	94
AWS IoT Greengrass Mise à jour logicielle Core v2.5.3	95
Mises à jour des composants publics	95
AWS IoT Greengrass Mise à jour logicielle Core v2.5.2	97
Mises à jour des composants publics	97
AWS IoT Greengrass Mise à jour logicielle Core v2.5.1	98
Mises à jour des composants publics	99
AWS IoT Greengrass Mise à jour logicielle Core v2.5.0	100
Mises à jour du support technique	101
Mises à jour des composants publics	101
AWS IoT Greengrass Mise à jour logicielle Core v2.4.0	106
Mises à jour des composants publics	107
AWS IoT Greengrass Mise à jour logicielle Core v2.3.0	110
Mises à jour des composants publics	110
AWS IoT Greengrass Mise à jour logicielle Core v2.2.0	112
Mises à jour des composants publics	112
AWS IoT Greengrass Mise à jour logicielle Core v2.1.0	115
Mises à jour du support technique	116
Mises à jour des composants publics	117

AWS IoT Greengrass Mise à jour logicielle Core v2.0.5	125
Mises à jour des composants publics	125
AWS IoT Greengrass Mise à jour logicielle Core v2.0.4	126
Mises à jour des composants publics	126
Migrer depuis la version 1	128
Présentation de la migration	128
Différences entre V1 et V2	130
Choisissez votre environnement d'exécution	143
Matrice de compatibilité des sources d'événements	143
Flux décisionnel de sélection du temps d'exécution	144
Étapes suivantes	145
Configuration de l'appareil de test	145
Configurer le dispositif de test (Greengrass nucleus)	146
Configuration du dispositif de test (Greengrass nucleus lite)	152
Mise à niveau des appareils principaux V1 vers la V2	221
Étape 1 : Installation du logiciel AWS IoT Greengrass Core v2.x	222
Étape 2 : Déployer les composants Greengrass V2 sur les appareils principaux	225
Premiers pas	228
Prérequis	229
Étape 1 : créer un AWS compte	231
Inscrivez-vous pour un Compte AWS	231
Création d'un utilisateur doté d'un accès administratif	231
Étape 2 : Configuration de votre environnement	233
Étape 3 : Installation du logiciel AWS IoT Greengrass principal	239
Installation du logiciel AWS IoT Greengrass Core (console)	240
Installation du logiciel AWS IoT Greengrass principal (CLI)	247
Exécutez le logiciel Greengrass (Linux)	252
Vérifiez l'installation de Greengrass CLI sur l'appareil	253
Étape 4 : développer et tester un composant sur votre appareil	255
Étape 5 : Créez votre composant dans le AWS IoT Greengrass service	267
Étape 6 : Déployez votre composant	279
Étapes suivantes	284
Configuration des appareils principaux de Greengrass	285
Plateformes prises en charge	285
Exigences relatives aux dispositifs	286
Exigences relatives à la fonction Lambda	286

Configurez un Compte AWS	288
Installation du logiciel AWS IoT Greengrass de base	289
Installation avec provisionnement automatique	292
Installation avec provisionnement manuel	309
Installation avec provisionnement du parc	349
Installation avec provisionnement personnalisé	397
Arguments d'installation	415
Exécutez le logiciel AWS IoT Greengrass Core	421
Vérifiez si le logiciel AWS IoT Greengrass Core fonctionne en tant que service système	421
Exécutez le logiciel AWS IoT Greengrass Core en tant que service système	423
Exécutez le logiciel AWS IoT Greengrass Core sans service système	424
Exécuter AWS IoT Greengrass dans Docker	424
Exigences et plateformes prises en charge	425
Téléchargements de logiciels	426
Choisissez le mode de provisionnement AWS des ressources	426
Construire l' AWS IoT Greengrass image à partir d'un Dockerfile	427
Exécuter AWS IoT Greengrass dans Docker avec provisionnement automatique	433
Exécuter AWS IoT Greengrass dans Docker avec provisionnement manuel	442
Résolution des problèmes AWS IoT Greengrass dans un conteneur Docker	462
Configuration du logiciel AWS IoT Greengrass de base	465
Déployez le composant Greengrass nucleus	466
Configurer le noyau Greengrass en tant que service système	466
Contrôlez l'allocation de mémoire grâce aux options JVM	470
Configurer l'utilisateur qui exécute les composants	472
Configurer les limites des ressources du système	477
Connexion au port 443 ou via un proxy réseau	480
Utiliser un certificat d'appareil signé par une autorité de certification privée	488
Configurer les délais d'expiration et les paramètres de cache du MQTT	489
Configurer Greengrass Nucleus sur le réseau IPv6	489
Mettre à jour le logiciel AWS IoT Greengrass principal (OTA)	490
Prérequis	490
Considérations relatives aux appareils principaux	491
Comportement de mise à jour du noyau Greengrass	491
Effectuer une mise à jour OTA	493
Désinstallez le logiciel AWS IoT Greengrass Core	493
Tutoriels	497

Didacticiels vidéo	497
Vidéos	497
Développer un composant qui reporte les mises à jour des composants	498
Prérequis	499
Étape 1 : Installation de la CLI du kit de développement Greengrass	501
Étape 2 : développer un composant qui reporte les mises à jour	501
Étape 3 : Publier le composant sur le AWS IoT Greengrass service	510
Étape 4 : Déployer et tester le composant sur un appareil principal	514
Interagissez avec des appareils IoT locaux via MQTT	519
Prérequis	520
Étape 1 : révision et mise à jour de la AWS IoT politique de base relative aux appareils	521
Étape 2 : activer la prise en charge des appareils clients	523
Étape 3 : Connecter les appareils clients	529
Étape 4 : développer un composant qui communique avec les appareils clients	532
Étape 5 : développer un composant qui interagit avec les ombres du périphérique client	539
Sécurisez Greengrass Nucleus avec TPM	566
Prérequis	567
Étape 1 : Installation TPM2 des outils et des dépendances	567
Étape 2 : Initialisation du magasin PKCS #11 et création d'un emplacement	568
Étape 3 : Création d'un jeton et d'une clé	569
Étape 4 : générer une demande de signature de certificat (CSR)	572
Étape 5 : Création du certificat d'objet	572
Étape 6 : Importer un certificat d'objet dans le TPM	572
Étape 7 : Capture de l'URL de l'objet PKCS #11	573
Étape 8 : Configuration et installation de Greengrass avec assistance TPM2	574
Étape 9 : vérifier l'installation	576
Résolution de problème	576
Étapes suivantes	577
Sécurisez Greengrass Nucleus Lite avec TPM	578
Prérequis	578
Étape 1 : configurer une instance NitroTPM	579
Étape 2 : Installation et configuration des outils TPM	579
Étape 3 : Configuration du fournisseur OpenSSL TPM2	579
Étape 4 : générer des clés TPM persistantes	580
Étape 5 : générer une demande de signature de certificat (CSR)	581
Étape 6 : Création du certificat à partir du CSR	581

Étape 7 : Configuration de Greengrass Nucleus Lite avec le support TPM	582
Résolution de problème	583
Étapes suivantes	584
Commencez avec SageMaker AI Edge Manager	584
Conditions préalables	585
Configuration dans SageMaker AI Edge Manager	587
Création des exemples de composants	589
Exécuter un exemple d'inférence de classification d'images	590
Effectuer une inférence de classification d'images d'échantillons	594
Prérequis	595
Étape 1 : Abonnez-vous à la rubrique de notifications par défaut	596
Étape 2 : Déploiement du composant de classification d'images TensorFlow Lite	596
Étape 3 : Afficher les résultats de l'inférence	598
Étapes suivantes	600
Effectuer une inférence de classification d'images d'échantillons sur des images provenant d'un appareil photo	601
Prérequis	602
Étape 1 : configurer le module de caméra sur votre appareil	603
Étape 2 : vérifier votre abonnement à la rubrique de notifications par défaut	605
Étape 3 : modifier la configuration du composant de classification d'images TensorFlow Lite et le déployer	605
Étape 4 : Afficher les résultats de l'inférence	608
Étapes suivantes	608
Composants	610
AWS-composants fournis	610
Noyau de Greengrass	630
Greengrass Nucleus Lite	677
Authentification de l'appareil client	683
CloudWatch métriques	762
AWS IoT Device Defender	788
spouleur à disque	806
Gestionnaire d'applications Docker	810
Connecteur Edge pour Kinesis Video Streams	821
Greengrass CLI	829
Détecteur IP	844
Firehose	855

Lanceur Lambda	873
Gestionnaire Lambda	877
Environnements d'exécution (runtimes) Lambda	887
Routeur d'abonnement Legacy	890
Console de débogage locale	902
Gestionnaire de journaux	919
Composants d'apprentissage automatique	970
Adaptateur de protocole Modbus-RTU	1094
Pont MQTT	1126
Courtier MQTT 3.1.1 (Moquette)	1151
Courtier MQTT 5 (EMQX)	1159
Émetteur de télémétrie Nucleus	1177
Fournisseur PKCS #11	1190
Directeur des services secrets	1199
Tunneling sécurisé	1211
Shadow Manager	1224
Amazon SNS	1254
Gestionnaire de flux	1272
Redirecteur de journal du système	1287
Agent Systems Manager	1293
Service d'échange de jetons	1300
Collecteur IoT SiteWise OPC UA	1304
Simulateur de source de données IoT SiteWise OPC UA	1314
SiteWise Éditeur IoT	1317
SiteWise Processeur IoT	1333
Composants pris en charge par l'éditeur	1349
AIShield.Bord	1350
EdgeLabs Capteur AI	1350
Greengrass S3 Inventor	1351
Composantes communautaires	1352
Outils de développement Greengrass	1356
Kit de développement Greengrass CLI	1357
Interface de ligne de commande Greengrass	1389
Utiliser le framework de test Greengrass	1408
Développer des composants	1425
Cycle de vie des composants	1427

Types de composants	1428
Création de composants	1429
Testez les composants avec des déploiements locaux	1442
Publier les composants à déployer	1445
Interagissez avec les AWS services	1451
Exécuter un conteneur Docker	1455
Référence de recette	1479
Variables d'environnement	1512
Déployer des composants sur des appareils	1513
Déploiements d'appareils principaux	1514
Résolution des dépendances entre plateformes	1514
Résolution de dépendance des composants	1514
Supprimer un appareil d'un groupe d'objets	1521
Déploiements	1523
Options de déploiement	1523
Créer des déploiements	1525
Création de sous-déploiements	1545
Réviser les déploiements	1549
Annuler les déploiements	1552
Vérifier l'état du déploiement	1552
Journalisation et surveillance	1557
Outils de surveillance	1557
Surveillez les journaux de Greengrass	1558
Accéder aux journaux du système de fichiers	1559
CloudWatch Journaux d'accès	1562
Accédez aux journaux de service du système	1564
Activer la journalisation dans CloudWatch Logs	1565
Configurer la journalisation pour AWS IoT Greengrass	1567
AWS CloudTrail journaux	1569
Enregistrez les appels d'API avec CloudTrail	1569
AWS IoT Greengrass V2 informations dans CloudTrail	1570
AWS IoT Greengrass événements de données dans CloudTrail	1571
AWS IoT Greengrass événements de gestion dans CloudTrail	1576
Comprendre les entrées du fichier AWS IoT Greengrass V2 journal	1576
Collectez les données de télémétrie relatives à l'état du système	1578
Métriques de télémétrie	1579

Configuration des paramètres de l'agent de télémétrie	1585
Abonnez-vous aux données de télémétrie dans EventBridge	1585
Recevez des notifications relatives au déploiement et à l'état de santé des composants	1594
Événement de modification de l'état du déploiement	1595
Événement de changement d'état du composant	1596
Conditions préalables à la création de règles EventBridge	1599
Configuration des notifications relatives à l'état de santé de l'appareil (console)	1599
Configuration des notifications d'état de l'appareil (CLI)	1601
Configurer les notifications relatives à l'état de l'appareil (CloudFormation)	1602
Consultez aussi	1602
Vérifier l'état de l'appareil principal	1602
Vérifier l'état de santé d'un appareil principal	1603
Vérifier l'état d'un groupe d'appareils principal	1603
Vérifier l'état des composants de l'appareil principal	1604
Exécuter les fonctions Lambda	1605
Prérequis	1606
Configurer le cycle de vie des fonctions Lambda	1606
Configuration de la conteneurisation des fonctions Lambda	1608
Importer une fonction Lambda en tant que composant (console)	1610
Étape 1 : Choisissez une fonction Lambda à importer	1611
Étape 2 : Configuration des paramètres de la fonction Lambda	1611
Étape 3 : (Facultatif) Spécifiez les plateformes prises en charge pour la fonction Lambda ..	1614
Étape 4 : (Facultatif) Spécifiez les dépendances des composants pour la fonction Lambda	1615
Étape 5 : (Facultatif) Exécuter la fonction Lambda dans un conteneur	1616
Étape 6 : Création du composant de fonction Lambda	1618
Importer une fonction Lambda en tant que composant (CLI)	1618
Étape 1 : Définition de la configuration de la fonction Lambda	1618
Étape 2 : Création du composant de fonction Lambda	1639
Communiquer avec le noyau de Greengrass, les autres composants et AWS IoT Core	1642
Versions du client IPC	1643
Soutenu SDKs	1644
Connectez-vous au service AWS IoT Greengrass Core IPC	1644
Autoriser les composants à effectuer des opérations IPC	1650
Des caractères génériques dans les politiques d'autorisation	1652
Variables de recette dans les politiques d'autorisation	1652
Caractères spéciaux dans les politiques d'autorisation	1653

Exemples de politiques d'autorisation	1654
Abonnez-vous aux diffusions d'événements IPC	1657
Définition des gestionnaires d'abonnements	1658
Exemples de gestionnaires d'abonnements	1661
Bonnes pratiques en matière d'IPC	1669
Publier/souscrire des messages locaux	1670
Versions minimales du SDK	1671
Autorisation	1672
PublishToTopic	1675
SubscribeToTopic	1683
Exemples	1696
Publier/souscrire AWS IoT Core des messages MQTT	1718
Versions minimales du SDK	1719
Autorisation	1719
PublishToIoTCore	1724
SubscribeToIoTCore	1734
Exemples	1748
Interagir avec le cycle de vie des composants	1756
Versions minimales du SDK	1757
Autorisation	1757
UpdateState	1759
SubscribeToComponentUpdates	1759
DeferComponentUpdate	1761
PauseComponent	1762
ResumeComponent	1764
Interagir avec la configuration des composants	1765
Versions minimales du SDK	1766
GetConfiguration	1766
UpdateConfiguration	1767
SubscribeToConfigurationUpdate	1768
SubscribeToValidateConfigurationUpdates	1769
SendConfigurationValidityReport	1771
Récupérez les valeurs secrètes	1772
Versions minimales du SDK	1772
Autorisation	1772
GetSecretValue	1774

Exemples	1780
Interagissez avec les ombres locales	1786
Versions minimales du SDK	1787
Autorisation	1787
GetThingShadow	1800
UpdateThingShadow	1807
DeleteThingShadow	1815
ListNamedShadowsForThing	1821
Gérez les déploiements et les composants locaux	1828
Versions minimales du SDK	1829
Autorisation	1830
CreateLocalDeployment	1832
ListLocalDeployments	1835
GetLocalDeploymentStatus	1836
ListComponents	1837
GetComponentDetails	1838
RestartComponent	1839
StopComponent	1840
CreateDebugPassword	1841
Authentifier et autoriser les appareils clients	1842
Versions minimales du SDK	1843
Autorisation	1843
VerifyClientDeviceIdentity	1845
GetClientDeviceAuthToken	1846
AuthorizeClientDeviceAction	1847
SubscribeToCertificateUpdates	1848
Interagissez avec les appareils IoT locaux	1850
Composants de l'appareil client	1851
Connect les appareils clients aux appareils principaux	1854
Prérequis	1855
Composants Greengrass pour le support des appareils clients	1868
Configuration de la découverte du cloud (console)	1870
Configurer la découverte du cloud (AWS CLI)	1871
Associer des appareils clients	1871
Authentification des clients hors ligne	1873
Gérez les principaux points de terminaison des appareils	1875

Choisissez un courtier MQTT	1883
Connexion à un courtier MQTT	1884
Test des communications	1887
API de découverte Greengrass RESTful	1899
Relayer les messages MQTT entre les appareils clients et AWS IoT Core	1906
Configuration et déploiement du composant de pont MQTT	1906
Messages MQTT relayés	1908
Interagir avec les appareils clients dans les composants	1909
Configuration et déploiement du composant de pont MQTT	1910
Recevoir des messages MQTT depuis les appareils clients	1911
Envoyer des messages MQTT aux appareils clients	1911
Interagissez avec les ombres des appareils clients et synchronisez-les	1912
Prérequis	1912
Permettre au Shadow Manager de communiquer avec les appareils clients	1913
Interagissez avec les ombres de l'appareil client dans les composants	1916
Synchroniser les ombres de l'appareil client avec AWS IoT Core	1916
Utilisation IPv6 pour la messagerie locale	1916
Configurer le détecteur IP à utiliser IPv6	1917
Résolution des problèmes	1921
Problèmes liés à Greengrass Discovery	1922
Problèmes de connexion MQTT	1930
Interagissez avec les ombres de l'appareil	1936
Interagir avec les ombres dans les composants	1937
Récupérez et modifiez les états d'ombre	1937
Réagir aux changements d'état de l'ombre	1938
Synchronisez les ombres de l'appareil local avec AWS IoT Core	1939
Prérequis	1940
Configuration du composant Shadow Manager	1940
Synchroniser les ombres locales	1942
Comportement des conflits liés à la fusion	1942
Gestion des flux de données	1944
Flux de travail de gestion des flux	1945
Prérequis	1946
Sécurité des données	1947
Sécurité des données locales	1947
Authentification client	1947

Consultez aussi	1948
Configuration du gestionnaire de flux	1948
Paramètres du gestionnaire de flux	1948
Consultez aussi	1951
Créez des composants personnalisés utilisant le gestionnaire de flux	1952
Définissez des recettes de composants utilisant le gestionnaire de flux	1952
Connectez-vous au gestionnaire de flux dans le code de l'application	1964
StreamManagerClient À utiliser pour travailler avec des flux	1967
Créer un flux de messages	1968
Ajouter un message	1972
Lire des messages	1979
Afficher la liste des flux	1981
Décrire le flux de messages	1983
Mettre à jour le flux de messages	1985
Supprimer le flux de messages	1989
Consultez aussi	1991
Exporter les configurations pour les destinations cloud prises en charge	1991
Exécuter l'inférence de Machine Learning	2008
Comment fonctionne l'inférence AWS IoT Greengrass ML	2008
Qu'est-ce qui est différent dans AWS IoT Greengrass la version 2 ?	2010
Prérequis	2010
Sources de modèles prises en charge	2011
Environnements d'exécution pris en charge	2011
Composants d'apprentissage automatique	2012
Utiliser SageMaker AI Edge Manager	2020
Comment ça marche	2021
Prérequis	2022
Commencez avec SageMaker AI Edge Manager	2024
Personnalisez vos composants d'apprentissage automatique	2024
Modifier la configuration d'un composant d'inférence public	2025
Utiliser un modèle personnalisé avec le composant d'inférence d'échantillon	2027
Création de composants d'apprentissage automatique personnalisés	2031
Création d'un composant d'inférence personnalisé	2034
Résolution des problèmes	2041
Impossible de récupérer la bibliothèque	2042
Cannot open shared object file	2043

Error: ModuleNotFoundError: No module named '<library>'	2043
Aucun appareil compatible CUDA n'est détecté	2045
Aucun fichier ou répertoire de ce type	2045
RuntimeError: module compiled against API version 0xf but this version of NumPy is <version>	2046
picamera.exc.PiCameraError: Camera is not enabled	2046
Erreurs mémoires	2047
Erreurs liées à l'espace disque	2047
Erreurs de délai d'attente	2047
Gérez les principaux appareils avec AWS Systems Manager	2048
Installer l'agent du gestionnaire de systèmes	2049
Étape 1 : Exécution des étapes générales de configuration de Systems Manager	2050
Étape 2 : Création d'un rôle de service IAM pour Systems Manager	2050
Étape 3 : ajouter des autorisations au rôle d'échange de jetons	2050
Étape 4 : Déploiement du composant Systems Manager Agent	2055
Étape 5 : vérifier l'enregistrement de l'appareil principal avec Systems Manager	2058
Désinstallez l'agent Systems Manager	2059
Étape 1 : Désenregistrer le périphérique principal de Systems Manager	2060
Étape 2 : désinstallation du composant de l'agent Systems Manager	2060
Étape 3 : désinstallez le logiciel Systems Manager Agent	2061
Sécurité	2062
Protection des données	2063
Chiffrement des données	2064
Intégration de sécurité matérielle	2066
Authentification et autorisation d'appareil	2079
Certificats X.509	2080
AWS IoT politiques	2081
Mettre à jour la AWS IoT politique d'un appareil principal	2087
AWS IoT Politique minimale	2092
AWS IoT Politique minimale de prise en charge des appareils clients	2095
AWS IoT Politique minimale pour les appareils clients	2097
Gestion des identités et des accès	2099
Public ciblé	2099
Authentification par des identités	2100
Gestion de l'accès à l'aide de politiques	2101
Consultez aussi	2103

Comment AWS IoT Greengrass fonctionne avec IAM	2103
Exemples de politiques basées sur l'identité	2107
Autoriser les appareils principaux à interagir avec les AWS services	2110
Politique IAM minimale permettant au programme d'installation de provisionner les ressources	2116
Rôle de service Greengrass	2119
AWS politiques gérées	2128
Prévention du problème de l'adjoint confus entre services	2135
Résolution des problèmes d'identité et d'accès	2136
Autoriser le trafic des appareils via un proxy ou un pare-feu	2138
Points de terminaison pour le fonctionnement de base	2138
Points de terminaison pour l'installation avec provisionnement automatique	2144
Points de terminaison pour les composants AWS fournis	2145
Validation de conformité	2146
Points de terminaison FIPS	2146
Activez les points de terminaison FIPS grâce au déploiement	2147
Installation de Nucleus avec des points de terminaison FIPS avec provisionnement manuel des ressources	2149
Installation de points de terminaison FIPS avec provisionnement de flotte	2184
Installation de points de terminaison FIPS avec provisionnement automatique des ressources	2203
Composants de première partie conformes à la norme FIPS	2220
Résilience	2221
Sécurité de l'infrastructure	2222
Analyse de la configuration et des vulnérabilités	2223
Intégrité du code	2224
Points de terminaison d'un VPC (AWS PrivateLink)	2225
Considérations relatives aux points de AWS IoT Greengrass terminaison VPC	2226
Création d'un point de terminaison VPC d'interface pour les opérations du plan AWS IoT Greengrass de contrôle	2226
Création d'une politique de point de terminaison VPC pour AWS IoT Greengrass	2227
Faire fonctionner un appareil AWS IoT Greengrass principal dans un VPC	2228
Bonnes pratiques de sécurité	2234
Accorder le moins d'autorisations possibles	2234
Ne codez pas en dur les informations d'identification dans les composants Greengrass	2234
Ne journalisez pas les informations sensibles	2235

Veiller à la synchronisation de l'horloge de votre appareil	2235
Recommandations de Cipher Suite	2236
Consultez aussi	2236
Utilisation AWS IoT Device Tester pour la AWS IoT Greengrass V2	2237
AWS IoT Greengrass suite de qualifications	2237
Suites de tests personnalisées	2238
Versions prises en charge	2238
Dernière version IDT pour V2 AWS IoT Greengrass	2239
Versions IDT antérieures pour AWS IoT Greengrass	2240
Versions non prises en charge de AWS IoT Device Tester for V2 AWS IoT Greengrass	2240
Télécharger IDT pour V2 AWS IoT Greengrass	2246
Télécharger IDT manuellement	2247
Téléchargez IDT par programmation	2247
Utiliser IDT pour exécuter la suite de AWS IoT Greengrass qualifications	2253
Versions de la suite de tests	2254
Descriptions des groupes de tests	2254
Prérequis	2257
Configurez votre appareil pour exécuter des tests IDT	2279
Configurer les paramètres IDT	2290
Exécutez la suite AWS IoT Greengrass de qualifications	2308
Présentation des résultats et des journaux	2312
Utilisez IDT pour développer et exécuter vos propres suites de tests	2315
Télécharger la dernière version d'IDT pour AWS IoT Greengrass	2257
Flux de travail de création de suites de tests	2316
Tutoriel : création et exécution de l'exemple de suite de tests IDT	2317
Tutoriel : Développement d'une suite de tests IDT simple	2323
Création de fichiers de configuration de la suite de tests IDT	2332
Configuration de l'orchestrateur de test IDT	2340
Configuration de la machine d'état IDT	2348
Création d'exécutables de scénarios de test IDT	2372
Utiliser le contexte IDT	2380
Configuration des paramètres pour les testeurs	2385
Déboguer et exécuter des suites de tests personnalisées	2396
Consulter les résultats et les journaux des tests IDT	2399
Métriques d'utilisation de l'IDT	2406
Résolution des problèmes liés à IDT pour V2 AWS IoT Greengrass	2413

Où rechercher les erreurs	2413
Résolution des erreurs IDT pour la AWS IoT Greengrass V2	2415
Politique de support AWS IoT Device Tester pour AWS IoT Greengrass	2422
Solutions IoT basées sur Greengrass	2423
Eurotech	2423
Résolution des problèmes	2424
Afficher les journaux des logiciels AWS IoT Greengrass principaux et des composants	2424
AWS IoT Greengrass Principaux problèmes liés au logiciel	2424
ThrottlingException depuis l' ListDeployments API	2426
Impossible de configurer le périphérique principal	2426
Impossible de démarrer le logiciel AWS IoT Greengrass Core en tant que service système	2426
Impossible de configurer Nucleus en tant que service système	2426
Impossible de se connecter à AWS IoT Core	2427
Erreur de mémoire insuffisante	2427
Impossible d'installer Greengrass CLI	2428
User root is not allowed to execute	2428
com.aws.greengrass.lifecyclemanager.GenericExternalService: Could not determine user/ group to run with	2428
Failed to map segment from shared object: operation not permitted	2429
Impossible de configurer le service Windows	2429
com.aws.greengrass.util.exceptions.TLSAuthException: Failed to get trust manager	2430
com.aws.greengrass.deployment.lotJobsHelper: No connection available during subscribing to lot Jobs descriptions topic. Will retry in sometime	2430
software.amazon.awssdk.services.iam.model.IamException: The security token included in the request is invalid	2431
software.amazon.awssdk.services.iot.model.IotException: User: <user> is not authorized to perform: iot:GetPolicy	2432
Error: com.aws.greengrass.shadowmanager.sync.model.FullShadowSyncRequest: Could not execute cloud shadow get request	2432
Operation aws.greengrass#<operation> is not supported by Greengrass	2433
java.io.FileNotFoundException: <stream-manager-store-root-dir>/ stream_manager_metadata_store (Permission denied)	2433
com.aws.greengrass.security.provider.pkcs11.PKCS11CryptoKeyService: Private key or certificate with label <label> does not exist	2434
software.amazon.awssdk.services.secretsmanager.model.SecretsManagerException: User: <user> is not authorized to perform: secretsmanager:GetSecretValue on resource: <arn> .	2434

software.amazon.awssdk.services.secretsmanager.model.SecretsManagerException:	
Access to KMS is not allowed	2435
java.lang.NoClassDefFoundError: com/aws/greengrass/security/CryptoKeySpi	2436
com.aws.greengrass.security.provider.pkcs11.PKCS11CryptoKeyService:	
CKR_OPERATION_NOT_INITIALIZED	2436
Greengrass core device stuck on nucleus v2.12.3	2436
Greengrass nucleus v2.14.0 systemd template issue	2438
AWS IoT Greengrass problèmes liés au cloud	2439
An error occurred (AccessDeniedException) when calling the CreateComponentVersion operation: User: arn:aws:iam::123456789012:user/<username> is not authorized to perform:	
null	2439
Invalid Input: Encountered following errors in Artifacts: {<s3ArtifactUri> = Specified artifact resource cannot be accessed}	2440
INACTIVE deployment status	2440
Principaux problèmes liés au déploiement des appareils	2440
Error: com.aws.greengrass.componentmanager.exceptions.PackageDownloadException:	
Failed to download artifact	2441
Error:	
com.aws.greengrass.componentmanager.exceptions.ArtifactChecksumMismatchException:	
Integrity check for downloaded artifact failed. Probably due to file corruption.	2443
Error:	
com.aws.greengrass.componentmanager.exceptions.NoAvailableComponentVersionException:	
Failed to negotiate component <name> version with cloud and no local applicable version satisfying requirement <requirements>	2443
software.amazon.awssdk.services.greengrassv2data.model.ResourceNotFoundException:	
The latest version of Component <componentName> doesn't claim platform <coreDevicePlatform> compatibility	2444
com.aws.greengrass.componentmanager.exceptions.PackagingException: The deployment attempts to update the nucleus from aws.greengrass.Nucleus-<version> to aws.greengrass.Nucleus-<version> but no component of type nucleus was included as target component	2445
Error: com.aws.greengrass.deployment.exceptions.DeploymentException: Unable to process deployment. Greengrass launch directory is not set up or Greengrass is not set up as a system service	2445

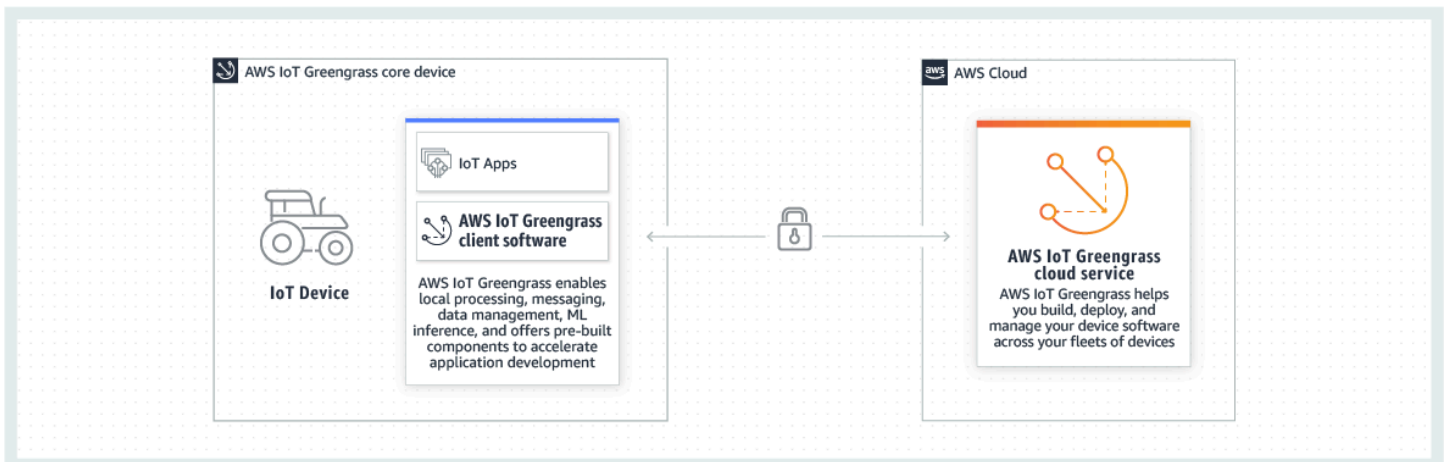
Info:	
com.aws.greengrass.deployment.exceptions.RetryableDeploymentDocumentDownloadException:	
Greengrass Cloud Service returned an error when getting full deployment configuration	2447
Warn: com.aws.greengrass.deployment.DeploymentService: Failed to get thing group	
hierarchy	2447
Info: com.aws.greengrass.deployment.DeploymentDocumentDownloader: Calling	
Greengrass cloud to get full deployment configuration	2448
Caused by:	
software.amazon.awssdk.services.greengrassv2data.model.GreengrassV2DataException:	
null (Service: GreengrassV2Data, Status Code: 403, Request ID: <some_request_id>,	
Extended Request ID: null)	2448
Principaux problèmes liés aux composants de l'appareil	2448
Warn: '<command>' is not recognized as an internal or external command	2449
Le script Python n'enregistre pas les messages	2450
La configuration des composants ne se met pas à jour lors de la modification de la	
configuration par défaut	2451
awsiot.greengrasscoreipc.model.UnauthorizedError	2452
com.aws.greengrass.authorization.exceptions.AuthorizationException: Duplicate policy ID	
"<id>" for principal "<componentList>"	2452
com.aws.greengrass.tes.CredentialRequestHandler: Error in retrieving AwsCredentials from	
TES (HTTP 400)	2453
com.aws.greengrass.tes.CredentialRequestHandler: Error in retrieving AwsCredentials from	
TES (HTTP 403)	2455
com.aws.greengrass.tes.CredentialsProviderError: Could not load credentials from any	
providers	2455
Received error when attempting to retrieve ECS metadata: Could not connect to the	
endpoint URL: "<tokenExchangeServiceEndpoint>"	2456
copyFrom: <configurationPath> is already a container, not a leaf	2456
com.aws.greengrass.componentmanager.plugins.docker.exceptions.DockerLoginException:	
Error logging into the registry using credentials - 'The stub received bad data.'	2457
java.io.IOException: Cannot run program "cmd" ...: [LogonUser] The password for this	
account has expired.	2457
aws.greengrass.StreamManager: Instant exceeds minimum or maximum instant	2459
Principaux problèmes liés aux composants de la fonction Lambda de l'appareil	2459
The following cgroup subsystems are not mounted: devices, memory	2459

ipc_client.py:64,HTTP Error 400:Bad Request, b'No subscription exists for the source <label-or-lambda-arn> and subject <label-or-lambda-arn>	2460
Version du composant abandonnée	2460
Problèmes liés à la Greengrass CLI	2461
java.lang.RuntimeException: Unable to create ipc client	2461
AWS CLI problèmes	2462
Error: Invalid choice: 'greengrassv2'	2462
Codes d'erreur de déploiement détaillés	2463
Erreur d'autorisation	2464
Erreur de demande	2466
Erreur de recette du composant	2468
AWS erreur de composant, erreur de composant utilisateur, erreur de composant	2470
Erreur de l'appareil	2471
Erreur de dépendance	2473
Erreur HTTP	2474
Erreur réseau	2475
Erreur Nucleus	2475
Erreur du serveur	2476
Erreur du service cloud	2477
Erreurs génériques	2478
Erreur inconnue	2479
Codes d'état détaillés des composants	2479
Balisage de vos ressources	2482
Utilisation de balises dans AWS IoT Greengrass V2	2482
Marquez avec le AWS Management Console	2482
Marquer avec l' AWS IoT Greengrass V2 API	2482
Utilisation des balises avec des politiques IAM	2484
AWS CloudFormation ressources	2486
AWS IoT Greengrass et CloudFormation modèles	2486
ComponentVersion exemple de modèle	2486
Exemple de modèle de déploiement	2487
En savoir plus sur CloudFormation	2488
Logiciel open source	2489
Historique de la documentation	2490
AWS Glossaire	2561
.....	mmdlxii

Qu'est-ce que c'est AWS IoT Greengrass ?

AWS IoT Greengrass est un environnement d'exécution périphérique et un service cloud open source pour l'Internet des objets (IoT) qui vous aide à créer, déployer et gérer des applications IoT sur vos appareils. Vous pouvez l'utiliser AWS IoT Greengrass pour créer un logiciel qui permet à vos appareils d'agir localement sur les données qu'ils génèrent, d'exécuter des prédictions basées sur des modèles d'apprentissage automatique, ainsi que de filtrer et d'agrégier les données des appareils. AWS IoT Greengrass permet à vos appareils de collecter et d'analyser des données plus près de l'endroit où elles sont générées, de réagir de manière autonome aux événements locaux et de communiquer en toute sécurité avec les autres appareils du réseau local. Les appareils Greengrass peuvent également communiquer en toute sécurité avec AWS IoT Core et exporter des données IoT vers le. AWS Cloud Vous pouvez l'utiliser AWS IoT Greengrass pour créer des applications de périphérie à l'aide de modules logiciels prédéfinis, appelés composants, qui peuvent connecter vos appareils de périphérie à des AWS services ou à des services tiers. Vous pouvez également l'utiliser AWS IoT Greengrass pour emballer et exécuter votre logiciel à l'aide de fonctions Lambda, de conteneurs Docker, de processus de système d'exploitation natifs ou d'environnements d'exécution personnalisés de votre choix.

L'exemple suivant montre comment un AWS IoT Greengrass appareil interagit avec. AWS Cloud



Nouvelles fonctionnalités

AWS IoT Greengrass V2 introduit de nouvelles fonctionnalités et améliorations. Vous trouverez ci-dessous de plus amples informations sur les nouvelles fonctionnalités proposées dans la version 2.

- [Quoi de neuf dans AWS IoT Greengrass Version 2](#)

Pour les nouveaux utilisateurs de AWS IoT Greengrass

Si vous êtes nouveau dans ce AWS IoT Greengrass domaine, nous vous recommandons de consulter la section suivante :

- [Comment AWS IoT Greengrass fonctionne](#)

Suivez ensuite le [didacticiel de démarrage](#) pour essayer les fonctionnalités de base de AWS IoT Greengrass. Dans ce didacticiel, vous allez installer le logiciel AWS IoT Greengrass Core sur un appareil, développer un composant Hello World et emballer ce composant pour le déploiement.

Pour les utilisateurs existants de AWS IoT Greengrass V1

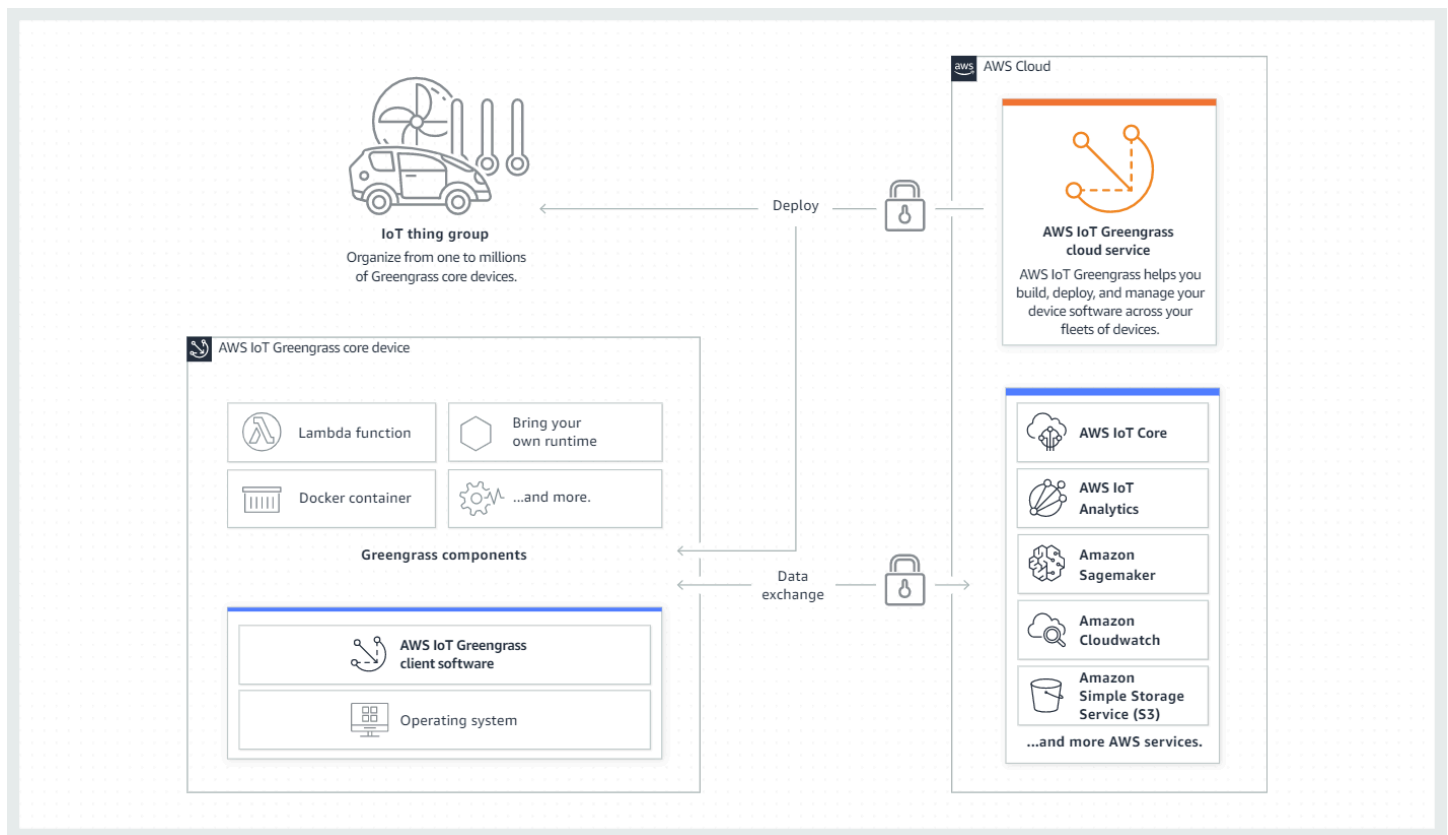
Avis de fin de support : le 7 octobre 2026, AWS le support de AWS IoT Greengrass Version 1. Après le 7 octobre 2026, vous ne pourrez plus accéder à la AWS IoT Greengrass V1 console ni aux AWS IoT Greengrass V1 ressources. Pour plus d'informations sur le passage de la version 1 à la version 2, consultez [Migrer depuis AWS IoT Greengrass la version 1](#).

Comment AWS IoT Greengrass fonctionne

Le logiciel AWS IoT Greengrass client, également appelé logiciel AWS IoT Greengrass Core, fonctionne sur des distributions Windows et Linux, telles que Ubuntu ou Raspberry Pi OS, pour les appareils dotés d'architectures ARM ou x86. Vous pouvez ainsi programmer les appareils pour qu'ils agissent localement sur les données qu'ils génèrent, exécuter des prédictions basées sur des modèles d'apprentissage automatique, et filtrer et agréger les données des appareils. AWS IoT Greengrass permet l'exécution locale de AWS Lambda fonctions, de conteneurs Docker, de processus de système d'exploitation natifs ou d'environnements d'exécution personnalisés de votre choix.

AWS IoT Greengrass fournit des modules logiciels prédéfinis appelés composants qui vous permettent d'étendre facilement les fonctionnalités des appareils de pointe. AWS IoT Greengrass les composants vous permettent de vous connecter à AWS des services et à des applications tierces en périphérie. Une fois que vous avez développé vos applications AWS IoT Greengrass IoT, vous pouvez déployer, configurer et gérer à distance ces applications sur votre parc d'appareils sur le terrain.

L'exemple suivant montre comment un AWS IoT Greengrass appareil interagit avec le service AWS IoT Greengrass cloud et les autres AWS services du AWS Cloud.



Concepts clés pour AWS IoT Greengrass

Les concepts suivants sont essentiels à la compréhension et à l'utilisation AWS IoT Greengrass :

AWS IoT chose

Un AWS IoT objet est une représentation d'un dispositif ou d'une entité logique spécifique. Les informations relatives à un objet sont stockées dans le AWS IoT registre.

Appareil Greengrass Core

Un appareil qui exécute le logiciel AWS IoT Greengrass Core. Un appareil Greengrass Core est un produit de l' AWS IoT. Vous pouvez ajouter plusieurs appareils principaux à des groupes AWS IoT d'objets pour créer et gérer des groupes d'appareils principaux Greengrass. Pour de plus amples informations, veuillez consulter [Configuration des appareils AWS IoT Greengrass principaux](#).

Appareil client Greengrass

Un appareil qui se connecte et communique avec un périphérique principal de Greengrass via MQTT. Un appareil client Greengrass existe. AWS IoT Le périphérique principal peut traiter,

filtrer et agréger les données provenant des appareils clients qui s'y connectent. Vous pouvez configurer le périphérique principal pour relayer les messages MQTT entre les appareils clients, le service AWS IoT Core cloud et les composants de Greengrass. Pour de plus amples informations, veuillez consulter [Interagissez avec les appareils IoT locaux](#).

Les appareils clients peuvent exécuter [FreeRTOS](#) ou utiliser l'API de [découverte Kit SDK des appareils AWS IoT Greengrass](#) pour obtenir des informations sur les principaux appareils auxquels ils peuvent se connecter.

Composant Greengrass

Module logiciel déployé et exécuté sur un appareil principal de Greengrass. Tous les logiciels développés et déployés avec AWS IoT Greengrass sont modélisés en tant que composant. AWS IoT Greengrass fournit des composants publics prédéfinis qui fournissent des fonctionnalités que vous pouvez utiliser dans vos applications. Vous pouvez également développer vos propres composants personnalisés, sur votre appareil local ou dans le cloud. Après avoir développé un composant personnalisé, vous pouvez utiliser le service AWS IoT Greengrass cloud pour le déployer sur un ou plusieurs appareils principaux. Vous pouvez créer un composant personnalisé et le déployer sur un appareil principal. Lorsque vous le faites, le périphérique principal télécharge les ressources suivantes pour exécuter le composant :

- Recette : fichier JSON ou YAML qui décrit le module logiciel en définissant les détails, la configuration et les paramètres des composants.
- Artifacts : code source, fichiers binaires ou scripts qui définissent le logiciel qui s'exécutera sur votre appareil. Vous pouvez créer des artefacts à partir de zéro, ou vous pouvez créer un composant à l'aide d'une fonction Lambda, d'un conteneur Docker ou d'un environnement d'exécution personnalisé.
- Dépendance : relation entre les composants qui vous permet d'imposer des mises à jour ou des redémarrages automatiques des composants dépendants. Par exemple, vous pouvez avoir un composant de traitement des messages sécurisé dépendant d'un composant de chiffrement. Cela garantit que toutes les mises à jour du composant de chiffrement mettent automatiquement à jour et redémarrent le composant de traitement des messages.

Pour plus d'informations, consultez [AWS-composants fournis](#) et [Développer des AWS IoT Greengrass composants](#).

Déploiement

Processus permettant d'envoyer des composants et d'appliquer la configuration de composants souhaitée à un équipement cible de destination, qui peut être un seul appareil principal

Greengrass ou un groupe de périphériques principaux Greengrass. Les déploiements appliquent automatiquement toutes les configurations de composants mises à jour à la cible et incluent tous les autres composants définis comme des dépendances. Vous pouvez également cloner un déploiement existant pour créer un nouveau déploiement qui utilise les mêmes composants mais qui est déployé sur une cible différente. Les déploiements sont continus, ce qui signifie que toutes les mises à jour que vous apportez aux composants ou à la configuration des composants d'un déploiement sont automatiquement envoyées à toutes les cibles de destination. Pour de plus amples informations, veuillez consulter [Déployer AWS IoT Greengrass des composants sur des appareils](#).

AWS IoT Greengrass Logiciel de base

À partir de la version 2.14, il AWS IoT Greengrass fournit deux implémentations alternatives de son environnement d'exécution de périphérique, un exécutable connu sous le nom de noyau. Le premier noyau, qui était auparavant le seul, est implémenté en Java. Ce choix offre la plus grande portabilité entre les architectures et les systèmes d'exploitation. Cependant, il s'accompagne également d'une dépendance à l'égard de la machine virtuelle Java, ce qui entraîne une importante empreinte mémoire.

Le second noyau, récemment ajouté, est implémenté en C. Ce choix réduit considérablement son encombrement. Cependant, il nécessite une distribution (ou une compilation à partir des sources) séparément pour les différentes architectures cibles et systèmes d'exploitation. Lorsqu'il sera nécessaire de distinguer les deux, nous désignerons la première implémentation sous le nom de noyau Greengrass et la seconde sous le nom de Greengrass nucleus lite.

- Composants facultatifs : ces composants configurables sont fournis par vos appareils Edge AWS IoT Greengrass et activent des fonctionnalités supplémentaires sur ceux-ci. En fonction de vos besoins, vous pouvez choisir les composants facultatifs que vous souhaitez déployer sur votre appareil, tels que le streaming de données, l'inférence d'apprentissage automatique local ou une interface de ligne de commande locale. Pour de plus amples informations, veuillez consulter [AWS-composants fournis](#).

Vous pouvez mettre à niveau votre logiciel AWS IoT Greengrass principal en déployant de nouvelles versions de vos composants sur votre appareil.

Caractéristiques de AWS IoT Greengrass

AWS IoT Greengrass Version 2 se compose des éléments suivants :

- Distributions de logiciels

- Le [composant Greengrass nucleus](#), qui est l'installation minimale du logiciel AWS IoT Greengrass Core. Ce composant gère les déploiements, l'orchestration et la gestion du cycle de vie des composants Greengrass.
- [Composants supplémentaires AWS fournis](#) en option qui s'intègrent aux services, aux protocoles et aux logiciels.
- [Outils de développement Greengrass](#), que vous pouvez utiliser pour créer, tester, créer, publier et déployer des composants Greengrass personnalisés.
- Le Kit SDK des appareils AWS IoT, qui contient la bibliothèque de [communication interprocessus \(IPC\) pour les composants Greengrass personnalisés et la bibliothèque](#) de découverte Greengrass pour les [appareils](#) clients.
- Le SDK Stream Manager, que vous pouvez utiliser pour [gérer les flux de données](#) sur les appareils principaux.
- Service dans le cloud
 - AWS IoT Greengrass V2 API
 - AWS IoT Greengrass V2 console

AWS IoT Greengrass Logiciel de base

Vous pouvez utiliser le logiciel AWS IoT Greengrass Core qui s'exécute sur vos appareils Edge pour effectuer les opérations suivantes :

- Traitez les flux de données sur l'appareil local avec des exportations automatiques vers le AWS cloud. Pour de plus amples informations, veuillez consulter [Gérez les flux de données sur les appareils principaux de Greengrass](#).
- Support de la messagerie MQTT entre AWS IoT et composants. Pour de plus amples informations, veuillez consulter [Publier/souscrire AWS IoT Core des messages MQTT](#).
- Interagissez avec les appareils locaux qui se connectent et communiquent via MQTT. Pour de plus amples informations, veuillez consulter [Interagissez avec les appareils IoT locaux](#).
- Support de publication locale et de messagerie d'abonnement entre les composants. Pour de plus amples informations, veuillez consulter [Publier/souscrire des messages locaux](#).
- Déployez et appelez des composants et des fonctions Lambda. Pour de plus amples informations, veuillez consulter [Déployer AWS IoT Greengrass des composants sur des appareils](#).

- Gérez le cycle de vie des composants, notamment en prenant en charge l'installation et l'exécution de scripts. Pour de plus amples informations, veuillez consulter [AWS IoT Greengrass référence de recette de composant](#).
- Effectuez des mises à jour logicielles sécurisées over-the-air (OTA) du logiciel AWS IoT Greengrass principal et des composants personnalisés. Pour plus d'informations, consultez [Mettre à jour le logiciel AWS IoT Greengrass principal \(OTA\)](#) et [Déployer AWS IoT Greengrass des composants sur des appareils](#).
- Fournissez un stockage sécurisé et crypté des secrets locaux et contrôlez l'accès par composants. Pour de plus amples informations, veuillez consulter [Directeur des services secrets](#).
- Connexions sécurisées entre les appareils et le AWS cloud grâce à l'authentification et à l'autorisation des appareils. Pour de plus amples informations, veuillez consulter [Authentification et autorisation de l'appareil pour AWS IoT Greengrass](#).

Vous configurez et gérez les appareils principaux de Greengrass grâce AWS IoT Greengrass APIs auxquels vous créez des déploiements logiciels continus. Pour de plus amples informations, veuillez consulter [Déployer AWS IoT Greengrass des composants sur des appareils](#).

Certaines fonctionnalités ne sont prises en charge que sur certaines plateformes. Pour de plus amples informations, veuillez consulter [Compatibilité des fonctionnalités de Greengrass](#).

Pour plus d'informations sur les plateformes prises en charge, les exigences et les téléchargements, consultez [Configuration des appareils AWS IoT Greengrass principaux](#).

Si vous téléchargez ce logiciel, vous acceptez le [contrat de licence du logiciel Greengrass Core](#).

Compatibilité des fonctionnalités de Greengrass

AWS IoT Greengrass prend en charge les appareils qui exécutent différents systèmes d'exploitation. Certaines fonctionnalités ne sont prises en charge que sur certains systèmes d'exploitation. Consultez les tableaux suivants pour savoir quelles fonctionnalités sont disponibles pour chaque système d'exploitation pris en charge. Pour plus d'informations sur les systèmes d'exploitation pris en charge, les exigences et la configuration des appareils principaux de Greengrass, consultez [Configuration des appareils AWS IoT Greengrass principaux](#)

Messagerie

Fonctionnalité	Linux	Windows	Greengrass Nucleus Lite (Linux)
Échange de messages MQTT entre AWS IoT et composants	 Oui	 Oui	 Oui (sauf pour les MQTT5 extensions)
Échange de publish/subscribe messages locaux entre les composants	 Oui	 Oui	 Oui
Interagissez avec des appareils IoT locaux via MQTT	 Oui	 Oui	 Non
Interagissez avec les appareils Modbus-RTU locaux à l'aide du composant Modbus-RTU	 Oui	 Non	 Non

Sécurité

Fonctionnalité	Linux	Windows	Greengrass Lite (Linux)
Connexions sécurisées avec authentification et autorisation de l'appareil	 Oui	 Oui	 Oui

Fonctionnalité	Linux	Windows	Greengrass Lite (Linux)
Déployez et accédez à des secrets sécurisés et chiffrés depuis AWS Secrets Manager	 Oui	 Oui	 Non
Utilisez un module de sécurité matérielle (HSM) pour stocker en toute sécurité la clé privée et le certificat de l'appareil	 Oui	 Non	Oui
Auditez les principaux appareils avec AWS IoT Device Defender	 Oui	 Oui	 Non
Utiliser les AWS informations d'identification pour interagir avec les AWS services	 Oui	 Oui	 Oui

Installation

Fonctionnalité	Linux	Windows	Greengrass Lite (Linux)
Installation AWS IoT Greengrass avec provisionnement automatique	 Oui	 Oui	 Non

Fonctionnalité	Linux	Windows	Greengrass Lite (Linux)
Installation AWS IoT Greengrass avec provisionnement manuel	 Oui	 Oui	 Oui
Installation AWS IoT Greengrass avec provisionnement du AWS IoT parc	 Oui	 Oui	 Oui
Installation AWS IoT Greengrass avec provisionnement personnalisé	<u>Oui</u>	<u>Oui</u>	 Oui
Exécuter AWS IoT Greengrass dans un conteneur Docker à l'aide d'une image Docker prédéfinie	 Oui	 Non	 Non

Note

AWS IoT Greengrass peut être installé et exécuté dans un conteneur docker compatible avec Systemd.

Maintenance et mises à jour à distance

Fonctionnalité	Linux	Windows	Greengrass Lite (Linux)
Effectuez des mises à jour logicielles sécurisées over-the-air (OTA)	 Oui	 Oui	 Oui
Gérez les principaux appareils avec AWS Systems Manager	 Oui	 Non	 Non
Connectez-vous aux principaux appareils grâce à un AWS IoT tunneling sécurisé	 Oui	 Non	 Oui

Machine learning

Fonctionnalité	Linux	Windows	Greengrass Lite (Linux)
Effectuez des inférences basées sur le machine learning à l'aide d'Amazon SageMaker AI Edge Manager	 Oui	 Oui	 Non
Effectuez des inférences basées sur le machine learning à l'aide d'Amazon Lookout for Vision	 Oui	 Non	 Non

Fonctionnalité	Linux	Windows	Greengrass Lite (Linux)
Effectuer une inférence d'apprentissage automatique à l'aide du DLR	 Oui	 Oui	 Non
Effectuez une inférence d'apprentissage automatique à l'aide de TensorFlow	 Oui	 Oui	 Non

Caractéristiques des composants

Fonctionnalité	Linux	Windows	Greengrass Lite (Linux)
Déployer et invoquer des fonctions Lambda	 Oui	 Non	 Non
Exécuter des conteneurs Docker dans des composants	 Oui	 Oui	 Oui
Traitez et exportez de gros volumes de données à l'aide du gestionnaire de flux	 Oui	 Oui	 Oui

Fonctionnalité	Linux	Windows	Greengrass Lite (Linux)
Gérez le cycle de vie des composants à l'aide de scripts de cycle de vie	 Oui	 Oui	 Oui
Interagissez avec les ombres de l'appareil	 Oui	 Oui	 Non
Charger des journaux sur Amazon CloudWatch Logs	 Oui	 Oui	 Oui
Chargez des données vers Amazon CloudWatch Metrics à l'aide du composant CloudWatch Metrics	 Oui	 Oui	 Oui
Publiez des messages sur Amazon Simple Notification Service à l'aide du composant Amazon SNS	 Oui	 Non	 Non
Publiez des données dans les flux de diffusion Amazon Kinesis Data Streams à l'aide du gestionnaire de flux	 Oui	 Oui	 Oui

Fonctionnalité	Linux	Windows	Greengrass Lite (Linux)
Publiez des données sur les flux de diffusion Amazon Data Firehose à l'aide du composant Firehose	 Oui	 Non	 Non
Collectez les mesures de télémétrie du système en temps réel et agissez en conséquence	 Oui	 Oui	 Non
Configuration des limites de ressources système pour les processus relatifs aux composants	 Oui	 Non	 Non
Suspendre et reprendre les processus relatifs aux composants	 Oui	 Non	 Non
Intégrer à AWS IoT SiteWise l'utilisation des AWS IoT SiteWise composants	 Oui	 Oui	 Non

Fonctionnalité	Linux	Windows	Greengrass Lite (Linux)
Publiez des flux vidéo sur Amazon Kinesis Video Streams à l'aide du connecteur Edge pour le composant Kinesis Video Streams	 Oui	 Non	 Non

Développement de composants

Fonctionnalité	Linux	Windows	Greengrass Lite (Linux)
Développez des composants localement sur les appareils principaux	 Oui	 Oui	 Oui
Interagissez avec un périphérique principal à l'aide de la AWS IoT Greengrass CLI	 Oui	 Oui	 Non
Interagissez avec un périphérique principal à l'aide de la console de débogage locale	 Oui	 Oui	 Non
Utiliser le Kit SDK des appareils AWS IoT pour Python dans les composants personnalisés	 Oui	 Oui	 Oui

Fonctionnalité	Linux	Windows	Greengrass Lite (Linux)
Utiliser le Kit SDK des appareils AWS IoT pour C ++ dans les composants personnalisés	 Oui	 Oui	 Oui
Utiliser le Kit SDK des appareils AWS IoT pour Java dans les composants personnalisés	 Oui	 Oui	 Oui

Certification de l'appareil

Fonctionnalité	Linux	Windows	Greengrass Lite (Linux)
À utiliser AWS IoT Device Tester AWS IoT Greengrass V2 pour valider les appareils IoT	 Oui	 Oui	 Non

Choix de votre environnement d'exécution AWS IoT Greengrass Nucleus

À partir de la version 2.14.0, AWS IoT Greengrass fournit deux implémentations alternatives de son environnement d'exécution de périphérique, un exécutable connu sous le nom de noyau. Malgré leurs différences d'implémentation, les deux environnements d'exécution sont compatibles avec le AWS IoT Greengrass service APIs et vous permettent de déployer des composants fournis par le SDK Greengrass AWS ou de développer des composants personnalisés à l'aide du SDK Greengrass. Il est également possible de mélanger des appareils, en utilisant l'un ou l'autre type de noyau au sein du même parc selon les besoins.

Cependant, afin d'obtenir la portabilité souhaitée ou les avantages spécifiques en matière d'économie de mémoire, il est essentiel de s'assurer que le noyau que vous déployez sur vos appareils Greengrass est compatible avec les composants que vous avez l'intention d'utiliser pour accélérer le développement de vos AWS IoT solutions. Pour en savoir plus sur la compatibilité des composants, consultez la section [Composants](#).

En fin de compte, le choix entre les deux options d'exécution de Greengrass dépendra de votre cas d'utilisation spécifique, des contraintes de l'appareil, des fonctionnalités requises et du système d'exploitation.

Noyau de Greengrass

AWS IoT Greengrass nucleus est un environnement d'exécution complet qui vous permet de fonctionner AWS IoT Greengrass sur un large éventail d'appareils, notamment des passerelles, des serveurs et des appareils de périphérie dotés de davantage de ressources de calcul. Envisagez de choisir le noyau Greengrass si :

- Ressources informatiques : votre appareil dispose de ressources de calcul suffisantes, telles que plus de 128 Mo de RAM et un processeur relativement puissant (supérieur à 1 GHz horloge, par exemple).
- Un support complet du système d'exploitation est nécessaire : Greengrass nucleus prend en charge le plus large éventail de systèmes d'exploitation (y compris la plupart des distributions Linux et Windows).
- Compatibilité des composants : Greengrass nucleus offre une compatibilité totale avec les composants existants publiés par l'équipe de AWS IoT service et les partenaires.

Greengrass Nucleus Lite

AWS IoT Greengrass nucleus lite est un environnement d'exécution léger et open source qui vous permet de fonctionner AWS IoT Greengrass sur des appareils aux ressources limitées. Cela peut être utile pour les ordinateurs monocarte à faible coût dotés d'applications à volume élevé, tels que les hubs domestiques intelligents, les compteurs d'énergie intelligents, les véhicules intelligents, l'IA de pointe et la robotique. Envisagez de choisir Greengrass Nucleus Lite si vos appareils sont :

- Ressources limitées : les ressources de votre appareil sont limitées, telles que la mémoire RAM (512 Mo ou moins), l'espace de stockage (FLASH) ou un processeur peu performant (moins de 1) GHz

- Dépendance limitée : la plate-forme logicielle du fournisseur de votre appareil ne prend pas en charge Java ni la JVM spécifique requise par Greengrass nucleus.
- Système d'exploitation : vos appareils exécutent une distribution Linux compatible avec systemd (par exemple : Ubuntu, Yocto).

Limitations actuelles de Greengrass Nucleus Lite

Comme inclus dans la AWS IoT Greengrass version 2.14.0, le runtime Greengrass nucleus lite (v.2.0.0) offre un sous-ensemble des fonctionnalités disponibles par Greengrass nucleus (v2.14.0).

Le mécanisme AWS IoT Greengrass IPC (communication interprocessus) permet aux composants de communiquer avec le noyau de Greengrass. La version allégée du noyau prend en charge le sous-ensemble suivant :

Fonctionnalité	Disponibilité
SubscribeToTopic	Available
PublishToTopic	Available
PublishToIoTCore	Available
SubscribeToIoTCore	Available
UpdateState	Non disponible actuellement
SubscribeToComponentUpdates	Non disponible actuellement
DeferComponentUpdate	Non disponible actuellement
GetConfiguration	Available
UpdateConfiguration	Available
SubscribeToConfigurationUpdate	Available
SubscribeToValidateConfigurationUpdates	Non disponible actuellement.
SendConfigurationValidityReport	Non disponible actuellement.

Fonctionnalité	Disponibilité
GetSecretValue	Non disponible actuellement.
PutComponentMetric	Non disponible actuellement
GetComponentDetails	Non disponible actuellement
RestartComponent	Non disponible actuellement
StopComponent	Non disponible actuellement
CreateLocalDeployment	Available
CancelLocalDeployment	Non disponible actuellement
GetLocalDeploymentStatus	Non disponible actuellement
ListLocalDeployments	Non disponible actuellement
ListComponents	Non disponible actuellement
ValidateAuthorizationToken	Available
CreateDebugPassword	Non disponible actuellement
PauseComponent	Non disponible actuellement
ResumeComponent	Non disponible actuellement
GetThingShadow	Non disponible actuellement
UpdateThingShadow	Non disponible actuellement
DeleteThingShadow	Non disponible actuellement
ListNamedShadowsForThing	Non disponible actuellement
SubscribeToCertificateUpdates	Non disponible actuellement
VerifyClientDeviceIdentity	Non disponible actuellement

Fonctionnalité	Disponibilité
GetClientDeviceAuthToken	Non disponible actuellement
AuthorizeClientDeviceAction	Non disponible actuellement

Quoi de neuf dans AWS IoT Greengrass Version 2

AWS IoT Greengrass Version 2 est une version majeure de AWS IoT Greengrass qui introduit les fonctionnalités suivantes :

- **AWS IoT Greengrass nucleus lite** — Les versions 2.14.0 et ultérieures de AWS IoT Greengrass AWS IoT Greengrass Nucleus proposent Nucleus Lite. Greengrass nucleus lite est un environnement d'exécution léger et open source qui vous permet d'exécuter Greengrass sur des appareils aux ressources limitées. Cela peut être utile pour les ordinateurs monocarte à faible coût dotés d'applications à volume élevé, tels que les hubs domestiques intelligents, les compteurs d'énergie intelligents, les véhicules intelligents, l'IA de pointe et la robotique. Pour de plus amples informations, veuillez consulter [Greengrass Nucleus Lite](#).
- **Composants pris en charge par l'éditeur** : propose AWS IoT Greengrass désormais des composants pris en charge par l'éditeur. Ces composants sont développés, proposés et entretenus par des fournisseurs tiers. Pour de plus amples informations, veuillez consulter [Composants pris en charge par l'éditeur](#).
- **Utiliser un appareil Greengrass en VPC** — L'utilisation d'un appareil principal Greengrass en VPC est désormais disponible. Cela vous permet d'effectuer des déploiements en VPC sans accès public à Internet. Pour de plus amples informations, veuillez consulter [Faire fonctionner un appareil AWS IoT Greengrass principal dans un VPC](#).
- **Greengrass Testing Framework (GTF)** — GTF pour AWS IoT Greengrass Version 2 est désormais disponible. GTF est un ensemble d'éléments de base destinés à soutenir l'end-to-end automatisé. Il permet aux clients AWS IoT Greengrass Version 2 internes d'utiliser le même cadre de test que celui utilisé par l'équipe de service à des fins de qualification des modifications logicielles, d'acceptation automatique et d'assurance qualité. Pour plus d'informations, consultez [Greengrass Testing Framework sur Github](#).
- **Certifié PSA** : les versions 2.7.0 et ultérieures de AWS IoT Greengrass Nucleus sont désormais certifiées Platform Security Architecture (PSA). Pour plus d'informations, voir [AWS IoT Greengrass est certifié PSA](#).

AWS IoT Greengrass les notes de version fournissent des détails sur AWS IoT Greengrass les versions : nouvelles fonctionnalités, mises à jour et améliorations, et correctifs généraux. AWS IoT Greengrass possède les types de versions suivants :

- Nouvelles versions de fonctionnalités pour AWS IoT Greengrass

- AWS IoT Greengrass Mises à jour logicielles de base

Cette section contient toutes les notes de AWS IoT Greengrass V2 publication, les plus récentes en premier, et inclut les principales modifications apportées aux fonctionnalités et les corrections de bogues importantes. Pour plus d'informations sur les corrections mineures supplémentaires, consultez l'organisation [aws-greengrass](https://github.com/aws-greengrass) sur GitHub

Notes de mise à jour

- [Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.16.1 le 23 décembre 2025](#)
- [Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.16.0 le 6 novembre 2025](#)
- [Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.15.1 le 6 novembre 2025](#)
- [Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.15.0 le 3 juillet 2025](#)
- [Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.14.3 le 11 avril 2025](#)
- [Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.14.2 le 27 mars 2025](#)
- [Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.14.1 le 7 février 2025](#)
- [Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.14.0 le 16 décembre 2024](#)
- [Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.13.0 le 26 août 2024](#)
- [Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.12.6 le 24 mai 2024](#)
- [Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.12.5 le 25 avril 2024](#)
- [Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.12.4 le 2 avril 2024](#)
- [Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.12.3 le 27 mars 2024](#)
- [Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.12.2 le 15 février 2024](#)
- [Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.12.1 le 8 décembre 2023](#)
- [Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.12.0 le 7 novembre 2023](#)
- [Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.11.3 le 18 octobre 2023](#)
- [Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.11.2 le 9 août 2023](#)
- [Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.11.1 le 21 juillet 2023](#)
- [Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.11.0 le 28 juin 2023](#)
- [Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.10.3 le 21 juin 2023](#)
- [Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.10.2 le 5 juin 2023](#)
- [Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.10.1 le 11 mai 2023](#)
- [Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.10.0 le 9 mai 2023](#)

- [Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.9.6 le 20 avril 2023](#)
- [Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.9.5 le 30 mars 2023](#)
- [Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.9.4 le 24 février 2023](#)
- [Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.9.3 le 1er février 2023](#)
- [Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.9.2 le 22 décembre 2022](#)
- [Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.9.1 le 18 novembre 2022](#)
- [Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.9.0 le 15 novembre 2022](#)
- [Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.8.1 le 13 octobre 2022](#)
- [Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.8.0 le 7 octobre 2022](#)
- [Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.7.0 le 28 juillet 2022](#)
- [Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.6.0 le 27 juin 2022](#)
- [Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.5.6 le 31 mai 2022](#)
- [Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.5.5 le 6 avril 2022](#)
- [Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.5.4 le 23 mars 2022](#)
- [Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.5.3 le 6 janvier 2022](#)
- [Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.5.2 le 3 décembre 2021](#)
- [Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.5.1 le 23 novembre 2021](#)
- [Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.5.0 le 12 novembre 2021](#)
- [Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.4.0 le 3 août 2021](#)
- [Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.3.0 le 29 juin 2021](#)
- [Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.2.0 le 18 juin 2021](#)
- [Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.1.0 le 26 avril 2021](#)
- [Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.0.5 le 09 mars 2021](#)
- [Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.0.4 le 04 février 2021](#)

Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.16.1 le 23 décembre 2025

Cette version fournit la version 2.16.1 du composant Greengrass nucleus et la version 2.16.1 du composant Greengrass CLI.

Date de sortie : 23 décembre 2025

Mises à jour des composants publics

Le tableau suivant répertorie les composants fournis par AWS qui incluent des fonctionnalités nouvelles et mises à jour.

Important

Lorsque vous déployez un composant, AWS IoT Greengrass installe les dernières versions prises en charge de toutes les dépendances de ce composant. De ce fait, les nouvelles versions de correctif des composants publics AWS fournis peuvent être automatiquement déployées sur vos appareils principaux si vous ajoutez de nouveaux appareils à un groupe d'objets ou si vous mettez à jour le déploiement qui cible ces appareils. Certaines mises à jour automatiques, telles que la mise à jour du noyau, peuvent provoquer le redémarrage inattendu de vos appareils.

Pour éviter les mises à jour involontaires d'un composant en cours d'exécution sur votre appareil, nous vous recommandons d'inclure directement votre version préférée de ce composant lorsque vous [créez un déploiement](#). Pour plus d'informations sur le comportement de mise à jour du logiciel AWS IoT Greengrass Core, consultez [Mettre à jour le logiciel AWS IoT Greengrass principal \(OTA\)](#).

Composant	Détails
Noyau de Greengrass	La version 2.16.1 du noyau Greengrass est disponible. Corrections de bugs et améliorations • Ajoute une configuration pour les intervalles entre les tentatives d'identification après les échecs du service d'échange de jetons.
Greengrass CLI	La version 2.16.1 de la CLI Greengrass est disponible. Version mise à jour pour la version 2.16.1 de Greengrass Nucleus.

Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.16.0 le 6 novembre 2025

Cette version fournit la version 2.16.0 du composant Greengrass Nucleus et des mises à jour des composants fournis. AWS

Date de sortie : 6 novembre 2025

Détails de la version

- [Mises à jour des composants publics](#)

Mises à jour des composants publics

Le tableau suivant répertorie les composants fournis par AWS qui incluent des fonctionnalités nouvelles et mises à jour.

Important

Lorsque vous déployez un composant, AWS IoT Greengrass installe les dernières versions prises en charge de toutes les dépendances de ce composant. De ce fait, les nouvelles versions de correctif des composants publics AWS fournis peuvent être automatiquement déployées sur vos appareils principaux si vous ajoutez de nouveaux appareils à un groupe d'objets ou si vous mettez à jour le déploiement qui cible ces appareils. Certaines mises à jour automatiques, telles que la mise à jour du noyau, peuvent provoquer le redémarrage inattendu de vos appareils.

Pour éviter les mises à jour involontaires d'un composant en cours d'exécution sur votre appareil, nous vous recommandons d'inclure directement votre version préférée de ce composant lorsque vous [créez un déploiement](#). Pour plus d'informations sur le comportement de mise à jour du logiciel AWS IoT Greengrass Core, consultez [Mettre à jour le logiciel AWS IoT Greengrass principal \(OTA\)](#).

Composant	Détails
Noyau de Greengrass	La version 2.16.0 du noyau Greengrass est disponible.

Composant	Détails
	Nouvelles fonctionnalités - Nucleus prend désormais en charge les cgroups V2. Corrections de bogues et améliorations - Résout un problème selon lequel Nucleus ne nettoyait pas les déploiements périmés lorsqu'un déploiement était annulé.
Greengrass Nucleus Lite	La version 2.3.0 du Greengrass Nucleus Lite est disponible. Nouvelles fonctionnalités - Support pour l'utilisation de TPM 2.0 pour l'autorisation IoT Core MQTT. - Les exemples de paquets apt supportent désormais davantage de systèmes d'exploitation : Ubuntu 22.04, Ubuntu 24.04, Debian 12 et Debian 13. - RestartComponent L'IPC est désormais pris en charge. Corrections de bogues et améliorations - Les déploiements locaux ne nécessitent plus d'accès à Internet. - GetConfiguration a été mis à jour pour correspondre au comportement d'exécution de Greengrass Nucleus. (Changement radical) - Correction et amélioration de bogues généraux
Shadow Manager	La version 2.3.12 du gestionnaire d'ombres est disponible. Corrections de bogues et améliorations Résout un problème en raison duquel les déploiements étaient bloqués lorsque plus de 1 024 devices shadow étaient configurés.
Gestionnaire de journaux	La version 2.3.11 du gestionnaire de journaux est disponible. Corrections de bogues et améliorations Résout un problème en raison duquel la configuration d'exécution de Log Manager augmentait indéfiniment en raison des informations périmées des fichiers journaux téléchargés.

Composant	Détails
Redirecteur de journal du système	La version 2.1.0 du redirecteur de journal du système est disponible. Corrections de bogues et améliorations • Met à jour la recette du composant pour qu'il supporte correctement le noyau de Greengrass. • Résultat de journalisation amélioré lorsqu'il n'y a aucun journal à télécharger. • Correction et amélioration de bogues généraux

Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.15.1 le 6 novembre 2025

Cette version fournit la version 2.15.1 du composant Greengrass nucleus et la version 2.15.1 du composant Greengrass CLI.

Date de sortie : 6 novembre 2025

Détails de la version

- [Mises à jour des composants publics](#)

Mises à jour des composants publics

Le tableau suivant répertorie les composants fournis par AWS qui incluent des fonctionnalités nouvelles et mises à jour.

Important

Lorsque vous déployez un composant, AWS IoT Greengrass installe les dernières versions prises en charge de toutes les dépendances de ce composant. De ce fait, les nouvelles versions de correctif des composants publics AWS fournis peuvent être automatiquement déployées sur vos appareils principaux si vous ajoutez de nouveaux appareils à un groupe d'objets ou si vous mettez à jour le déploiement qui cible ces appareils. Certaines mises à jour automatiques, telles que la mise à jour du noyau, peuvent provoquer le redémarrage inattendu de vos appareils.

Pour éviter les mises à jour involontaires d'un composant en cours d'exécution sur votre appareil, nous vous recommandons d'inclure directement votre version préférée de ce composant lorsque vous [créez un déploiement](#). Pour plus d'informations sur le comportement de mise à jour du logiciel AWS IoT Greengrass Core, consultez [Mettre à jour le logiciel AWS IoT Greengrass principal \(OTA\)](#).

Composant	Détails
Noyau de Greengrass La version 2.15.1	Corrections de bugs et améliorations Résout un problème selon lequel Nucleus ne parvenait pas à nettoyer les processus des composants après le délai d'arrêt de 30 secondes pendant les déploiements bootstrap.
Greengrass CLI La version 2.15.1	Version mise à jour pour la version 2.15.1 de Greengrass Nucleus.

Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.15.0 le 3 juillet 2025

Cette version fournit la version 2.15.0 du composant Greengrass Nucleus et des mises à jour des composants fournis. AWS

Date de sortie : 3 juillet 2025

Détails de la version

- [Mises à jour des composants publics](#)

Mises à jour des composants publics

Le tableau suivant répertorie les composants fournis par AWS qui incluent des fonctionnalités nouvelles et mises à jour.

⚠ Important

Lorsque vous déployez un composant, AWS IoT Greengrass installe les dernières versions prises en charge de toutes les dépendances de ce composant. De ce fait, les nouvelles versions de correctif des composants publics AWS fournis peuvent être automatiquement déployées sur vos appareils principaux si vous ajoutez de nouveaux appareils à un groupe d'objets ou si vous mettez à jour le déploiement qui cible ces appareils. Certaines mises à jour automatiques, telles que la mise à jour du noyau, peuvent provoquer le redémarrage inattendu de vos appareils.

Pour éviter les mises à jour involontaires d'un composant en cours d'exécution sur votre appareil, nous vous recommandons d'inclure directement votre version préférée de ce composant lorsque vous [créez un déploiement](#). Pour plus d'informations sur le comportement de mise à jour du logiciel AWS IoT Greengrass Core, consultez [Mettre à jour le logiciel AWS IoT Greengrass principal \(OTA\)](#).

Composant	Détails
Noyau de Greengrass	La version 2.15.0 du noyau Greengrass est disponible. Nouvelles fonctionnalités • Ajoute une fonction de télémétrie pour inclure des informations sur le système hôte, telles que les détails du processeur et du système d'exploitation. Pour plus d'informations, consultez la section Mesures de télémétrie. • Ajoute <code>deploymentConfigurationTimeSource</code> une configuration. Pour plus d'informations, consultez la section Configuration du noyau de Greengrass. Corrections de bogues et améliorations • Nucleus donne désormais la priorité à l'utilisation de versions et d'artefacts locaux par défaut, ce qui améliore la cohérence du déploiement et réduit les dépendances externes. Les utilisateurs peuvent modifier ce comportement en spécifiant d'autres préférences dans le document de déploiement.

Composant	Détails
	Ajoute un nouveau journal d'avertissement lorsqu'un utilisateur Windows fourni ne correspond pas au jeu de caractères normalement accepté par Windows.
Greengrass CLI	La version 2.15.0 de la CLI Greengrass est disponible. Corrections de bogues et améliorations - Ajoute un correctif afin que lorsque la configuration d'authentification change, le composant ne se réinstalle pas mais redémarre tout de même. - Correction et amélioration de bogues généraux
Greengrass Nucleus Lite	La version 2.2.0 du Greengrass Nucleus Lite est disponible. Nouvelles fonctionnalités - Ajoute la prise en charge de l'artefact URIs d'image du conteneur. Corrections de bogues et améliorations - Correction et amélioration de bogues généraux
Gestionnaire de journaux	La version 2.3.10 du gestionnaire de journaux est disponible. Nouvelles fonctionnalités - Ajoute une nouvelle clé de configuration (<code>updateToTlogIntervalSec</code>) pour contrôler la fréquence à laquelle les détails des événements de téléchargement du journal sont conservés dans le journal des transactions local (<code>config.tlog</code>). Corrections de bogues et améliorations - Améliore le gestionnaire de journaux pour actualiser le client Cloudwatch en cas d'erreur de connexion au socket.

Composant	Détails
Directeur des services secrets	La version 2.2.6 du gestionnaire de secrets est disponible. Corrections de bogues et améliorations • Résout un problème selon lequel le gestionnaire de secrets ne parvient pas à obtenir un secret en raison de la lenteur ou de l'absence de réponse du module Trusted Platform.

Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.14.3 le 11 avril 2025

Cette version fournit la version 2.14.3 du composant Greengrass Nucleus et des mises à jour des composants fournis. AWS

Date de sortie : 11 avril 2025

Détails de la version

- [Mises à jour des composants publics](#)

Mises à jour des composants publics

Le tableau suivant répertorie les composants fournis par AWS qui incluent des fonctionnalités nouvelles et mises à jour.

Important

Lorsque vous déployez un composant, AWS IoT Greengrass installe les dernières versions prises en charge de toutes les dépendances de ce composant. De ce fait, les nouvelles versions de correctif des composants publics AWS fournis peuvent être automatiquement déployées sur vos appareils principaux si vous ajoutez de nouveaux appareils à un groupe d'objets ou si vous mettez à jour le déploiement qui cible ces appareils. Certaines mises à jour automatiques, telles que la mise à jour du noyau, peuvent provoquer le redémarrage inattendu de vos appareils.

Pour éviter les mises à jour involontaires d'un composant en cours d'exécution sur votre appareil, nous vous recommandons d'inclure directement votre version préférée de ce

composant lorsque vous [créez un déploiement](#). Pour plus d'informations sur le comportement de mise à jour du logiciel AWS IoT Greengrass Core, consultez [Mettre à jour le logiciel AWS IoT Greengrass principal \(OTA\)](#).

Composant	Détails
Noyau de Greengrass	La version 2.14.3 du noyau Greengrass est disponible. Corrections de bogues et améliorations • Permet au service d'échange de jetons de redémarrer en cas de modification de la configuration du port. • Résout un problème selon lequel le service d'état du parc n'envoie pas de messages de modification de l'état des composants pour les lambdas non épinglés s'ils ne sont pas déclenchés. • Résout un problème en raison duquel les composants ne sont pas correctement arrêtés lors du déploiement d'une nouvelle version de composant. • Résout un problème selon lequel les hooks d'injection des plugins intégrés étaient exécutés deux fois, ce qui entraînait des événements de cycle de vie supplémentaires et une journalisation dupliquée. • Améliore la journalisation du cycle de vie des composants sur les appareils Windows.
Greengrass CLI	La version 2.14.3 de la CLI Greengrass est disponible. Corrections de bogues et améliorations • Version mise à jour pour la version 2.14.3 de Greengrass Nucleus.
Directeur secret	La version 2.2.4 du gestionnaire de secrets est disponible. Corrections de bogues et améliorations • Réduit la fréquence des écritures dans le magasin secret local. Le gestionnaire de secrets écrit désormais dans le magasin local uniquement lorsque les secrets sont mis à jour.

Composant	Détails
Courtier MQTT EMAX	La version 2.0.3 du broker EMQX MQTT est disponible. Corrections de bogues et améliorations • Résout le problème selon lequel EMQX ne démarre pas sous Windows si le chemin contient des espaces.

Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.14.2 le 27 mars 2025

Cette version fournit la version 2.14.2 du composant Greengrass Nucleus et des mises à jour des composants fournis. AWS

Date de sortie : 27 mars 2025

Détails de la version

- [Mises à jour des composants publics](#)

Mises à jour des composants publics

Le tableau suivant répertorie les composants fournis par AWS qui incluent des fonctionnalités nouvelles et mises à jour.

Important

Lorsque vous déployez un composant, AWS IoT Greengrass installe les dernières versions prises en charge de toutes les dépendances de ce composant. De ce fait, les nouvelles versions de correctif des composants publics AWS fournis peuvent être automatiquement déployées sur vos appareils principaux si vous ajoutez de nouveaux appareils à un groupe d'objets ou si vous mettez à jour le déploiement qui cible ces appareils. Certaines mises à jour automatiques, telles que la mise à jour du noyau, peuvent provoquer le redémarrage inattendu de vos appareils.

Pour éviter les mises à jour involontaires d'un composant en cours d'exécution sur votre appareil, nous vous recommandons d'inclure directement votre version préférée de ce composant lorsque vous [créez un déploiement](#). Pour plus d'informations sur le comportement

de mise à jour du logiciel AWS IoT Greengrass Core, consultez [Mettre à jour le logiciel AWS IoT Greengrass principal \(OTA\)](#).

Composant	Détails
Noyau de Greengrass	La version 2.14.2 du noyau Greengrass est disponible. Corrections de bogues et améliorations • Résout un problème selon lequel un client HTTP n'est pas configuré avec une authentification mutuelle.
Greengrass CLI	La version 2.14.2 de la CLI Greengrass est disponible. Corrections de bogues et améliorations • Version mise à jour pour la version 2.14.2 de Greengrass Nucleus.

Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.14.1 le 7 février 2025

Cette version fournit la version 2.14.1 du composant Greengrass Nucleus et des mises à jour des composants fournis. AWS

Date de sortie : 7 février 2025

Détails de la version

- [Mises à jour des composants publics](#)

Mises à jour des composants publics

Le tableau suivant répertorie les composants fournis par AWS qui incluent des fonctionnalités nouvelles et mises à jour.

Important

Lorsque vous déployez un composant, AWS IoT Greengrass installe les dernières versions prises en charge de toutes les dépendances de ce composant. De ce fait, les nouvelles

versions de correctif des composants publics AWS fournis peuvent être automatiquement déployées sur vos appareils principaux si vous ajoutez de nouveaux appareils à un groupe d'objets ou si vous mettez à jour le déploiement qui cible ces appareils. Certaines mises à jour automatiques, telles que la mise à jour du noyau, peuvent provoquer le redémarrage inattendu de vos appareils.

Pour éviter les mises à jour involontaires d'un composant en cours d'exécution sur votre appareil, nous vous recommandons d'inclure directement votre version préférée de ce composant lorsque vous [créez un déploiement](#). Pour plus d'informations sur le comportement de mise à jour du logiciel AWS IoT Greengrass Core, consultez [Mettre à jour le logiciel AWS IoT Greengrass principal \(OTA\)](#).

Composant	Détails
Noyau de Greengrass	La version 2.14.1 du noyau Greengrass est disponible. Corrections de bogues et améliorations • Résout un problème en raison duquel les composants n'étaient pas correctement arrêtés sur les nouvelles installations de Greengrass.
Greengrass CLI	La version 2.14.1 de la CLI Greengrass est disponible. Corrections de bogues et améliorations • Version mise à jour pour la version 2.14.1 de Greengrass Nucleus.
Tunneling sécurisé	La version 1.1.1 du composant de tunneling sécurisé est disponible. Corrections de bogues et améliorations • Ajoute une configuration pour prendre en charge Greengrass Nucleus Lite.

Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.14.0 le 16 décembre 2024

Cette version fournit la version 2.14.0 du composant Greengrass Nucleus, ainsi que de nouvelles AWS IoT Greengrass mises à jour de Nucleus Lite. Le AWS IoT Greengrass nucleus lite est un

nouveau runtime, disponible pour AWS IoT Greengrass la version 2. Il offre une alternative à l'encombrement mémoire réduit. Il s'agit d'une bonne option pour les appareils dont les ressources sont limitées. Il implémente un sous-ensemble des fonctionnalités du noyau avec une compatibilité accrue prévue pour les futures versions. Le code source est désormais disponible sur [Github](#). Avec le runtime Nucleus Lite, vous pouvez :

- Déployez des composants sur les appareils principaux de Greengrass. Utilisez le même format de recette, même si certaines fonctionnalités avancées ne sont peut-être pas encore disponibles.
- Les applications déployées en tant que composants Greengrass peuvent utiliser l'appareil SDKs pour accéder aux IPC Greengrass pris en charge APIs, tels que l'accès AWS IoT Core MQTT, le pub/sub local et l'accès à la configuration Greengrass. Consultez le tableau de compatibilité pour obtenir la liste des [IPC APIs pris en charge](#).
- Certains composants AWS gérés ont été mis à jour pour le support de Nucleus Lite. Consultez les [composants AWS fournis](#) pour obtenir la liste des composants compatibles existants.

Nouvelles fonctions :

- Utilise moins de mémoire et d'espace disque (moins de 5 Mo de RAM et moins de 5 Mo de stockage).
- Les composants s'intègrent au gestionnaire de services du système hôte (systemd pour les plateformes Linux actuellement prises en charge).

Points à surveiller :

- AWS IoT Greengrass les recettes Nucleus Lite distinguent les majuscules et minuscules. Assurez-vous que le bon boîtier (clés) est utilisé, comme indiqué dans la référence de la <https://docs.aws.amazon.com/greengrass/v2/developerguide/component-recipe-reference.html> recette.
- Le runtime Nucleus Lite prend en charge les déploiements par groupes d'objets, mais ne prend pas encore en charge le type de cible de déploiement (unique) d'appareils Core. Pour effectuer un déploiement sur un seul appareil Greengrass, utilisez un groupe d'objets contenant uniquement cet appareil.
- Le runtime Nucleus Lite utilise des ressources de mémoire limitées ; les fonctionnalités qui évoluent en fonction de l'utilisation sur le runtime classique peuvent échouer en raison du dépassement des ressources disponibles sur Lite. Cela inclut une limite actuelle sur un maximum de 50 abonnements MQTT à la fois, ainsi que des limites maximales sur la taille des fichiers

de recettes et les déploiements. Certaines de ces limites sont configurables au moment de la compilation si vous compilez vous-même le runtime Lite.

- Le runtime Nucleus Lite n'est pas fourni avec Java. Pour utiliser des composants nécessitant Java, le système doit avoir déjà installé Java, ou un composant peut être utilisé pour installer Java.
- Nous vous recommandons de compiler le runtime Nucleus Lite à partir des sources et d'utiliser votre propre version adaptée à votre système. Pour les systèmes Yocto, une couche est disponible pour intégrer le runtime Nucleus Lite dans l'image de votre système.
- Actuellement, le nucleus lite suppose un système Linux utilisant systemd, ou une image de conteneur utilisant systemd.
- Bien que vous puissiez gérer les conteneurs Docker à l'aide de scripts de recettes, les artefacts des conteneurs gérés par Greengrass ne sont pas encore disponibles.
- Le runtime Nucleus Lite ne prend pas encore en charge les clés stockées dans un PKCS11 module. Si votre cas d'utilisation nécessite des clés stockées sur un élément sécurisé, le runtime classique peut actuellement prendre en charge ce cas d'utilisation. Pour éviter toute fuite des informations d'identification de votre appareil, assurez-vous que les appareils de production utilisent le chiffrement complet du disque.

Parallèlement à l'introduction de Nucleus Lite, nous publions également la version 2.14.0 de Nucleus. Cette mise à jour apporte des améliorations significatives au noyau Greengrass existant.

Principales fonctionnalités et améliorations :

- La nouvelle prise en charge des terminaux à double pile permet la communication IPv6 réseau.
- Résilience améliorée contre les échecs de redémarrage du noyau et la corruption de répertoires.
- Correction de fuites de mémoire lors des fermetures PubSub d'abonnements IPC.

Date de sortie : 16 décembre 2024

Mises à jour des composants publics

Le tableau suivant répertorie les composants AWS fournis qui incluent des fonctionnalités nouvelles et mises à jour.

⚠ Important

Lorsque vous déployez un composant, AWS IoT Greengrass installe les dernières versions prises en charge de toutes les dépendances de ce composant. De ce fait, les nouvelles versions de correctif des composants publics AWS fournis peuvent être automatiquement déployées sur vos appareils principaux si vous ajoutez de nouveaux appareils à un groupe d'objets ou si vous mettez à jour le déploiement qui cible ces appareils. Certaines mises à jour automatiques, telles que la mise à jour du noyau, peuvent provoquer le redémarrage inattendu de vos appareils.

Pour éviter les mises à jour involontaires d'un composant en cours d'exécution sur votre appareil, nous vous recommandons d'inclure directement votre version préférée de ce composant lorsque vous [créez un déploiement](#). Pour plus d'informations sur le comportement de mise à jour du logiciel AWS IoT Greengrass Core, consultez [Mettre à jour le logiciel AWS IoT Greengrass principal \(OTA\)](#).

Composant	Détails
Greengrass Nucleus Lite	La version 2.0.0 du Greengrass noyau lite est disponible. Nouvelles fonctionnalités • Utilise moins de mémoire et d'espace disque (moins de 5 Mo de RAM et moins de 5 Mo de stockage). • Les composants s'intègrent au gestionnaire de services du système hôte (systemd pour les plateformes Linux actuellement prises en charge).
Noyau de Greengrass	La version 2.14.0 du noyau Greengrass est disponible. Nouvelles fonctionnalités • La nouvelle prise en charge des terminaux à double pile permet la communication IPv6 réseau. • Résilience améliorée contre les échecs de redémarrage du noyau et la corruption de répertoires. Corrections de bogues et améliorations • Correction de fuites de mémoire lors des fermetures PubSub d'abonnements IPC.

Composant	Détails
	• Corrige le cycle de vie d'exécution du composant lorsqu'il entre dans l'état ERRORED en raison du délai de démarrage lorsque la condition skipif est vraie. • Résout un problème en raison duquel le périphérique principal ne parvient pas à se connecter AWS IoT Core lorsque la politique TLS est définie sur TLS13 _1_3_2022_10.
Greengrass CLI	La version 2.14.0 de la CLI Greengrass est disponible. Corrections de bogues et améliorations • Validez le paramètre de cible de déploiement dans la commande cli.
Gestionnaire de flux	La version 2.14.0 du gestionnaire de flux est disponible. Nouvelles fonctionnalités • Ajoute une nouvelle clé de configuration pour l'expiration du délai de démarrage. La valeur par défaut est de 120 secondes. • Ajoutez des supports de recettes pour Greengrass Nucleus Lite.
Courtier MQTT 5 (EMQX)	La version 2.0.2 du broker MQTT 5 (EMQX) est disponible. Corrections de bogues et améliorations • Résout un problème selon lequel EMQX démarre avant que le composant d'authentification du périphérique client ne soit prêt.
Composant d'exécution Lambda	La version 2.0.9 du composant d'exécution Lambda est disponible. Corrections de bogues et améliorations • Corrige un avertissement de syntaxe avec Python 3.12
Composant du gestionnaire Lambda	La version 2.3.5 du composant Lambda Manager est disponible. Corrections de bogues et améliorations • Améliore les performances en utilisant epoll au lieu de nio lorsqu'il est disponible

Composant	Détails
Composant du gestionnaire de secrets	La version 2.2.2 du composant Secret manager est disponible. Corrections de bogues et améliorations • Résout un problème en raison duquel le gestionnaire de secrets ne télécharge pas les secrets configurés avec des arns partiels.
Composant de tunneling sécurisé	La version 1.1.0 du composant Secure Tunneling est disponible. Nouvelles fonctionnalités • Ajout de supports de recettes pour Greengrass Nucleus Lite
CloudWatch composant de métriques	La version 1.1.0 du composant CloudWatch métriques est disponible. Nouvelles fonctionnalités • Ajout de supports de recettes pour Greengrass Nucleus Lite

Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.13.0 le 26 août 2024

Cette version fournit la version 2.13.0 du composant Greengrass nucleus.

Date de sortie : 26 août 2024

Mises à jour des composants publics

Le tableau suivant répertorie les composants AWS fournis qui incluent des fonctionnalités nouvelles et mises à jour.

Important

Lorsque vous déployez un composant, AWS IoT Greengrass installe les dernières versions prises en charge de toutes les dépendances de ce composant. De ce fait, les nouvelles versions de correctif des composants publics AWS fournis peuvent être automatiquement déployées sur vos appareils principaux si vous ajoutez de nouveaux appareils à un groupe d'objets ou si vous mettez à jour le déploiement qui cible ces appareils. Certaines mises à

jour automatiques, telles que la mise à jour du noyau, peuvent provoquer le redémarrage inattendu de vos appareils.

Pour éviter les mises à jour involontaires d'un composant en cours d'exécution sur votre appareil, nous vous recommandons d'inclure directement votre version préférée de ce composant lorsque vous [créez un déploiement](#). Pour plus d'informations sur le comportement de mise à jour du logiciel AWS IoT Greengrass Core, consultez [Mettre à jour le logiciel AWS IoT Greengrass principal \(OTA\)](#).

Composant	Détails
Noyau de Greengrass	La version 2.13.0 du noyau Greengrass est disponible. Nouvelles fonctionnalités • Support du point de terminaison FIPS dans Nucleus. Pour de plus amples informations, veuillez consulter Points de terminaison FIPS. Corrections de bogues et améliorations • Annuler les améliorations apportées au déploiement : les déploiements peuvent désormais être annulés pendant la fusion d'une nouvelle configuration et en attendant le démarrage des services.
Gestionnaire de flux	La version 2.1.13 du composant Stream manager est disponible. Corrections de bogues et améliorations • Support du point de terminaison FIPS dans AWS IoT SiteWise
Directeur des services secrets	La version 2.2.0 du composant Secret manager est disponible. Nouvelles fonctionnalités • Prend en charge l'actualisation périodique des secrets configurés par le biais d'une nouvelle clé de configuration de composant. • Ajoute la prise en charge d'un nouveau paramètre de demande dans la requête GetSecretValueIPC pour actualiser les secrets par demande
Détecteur IP	La version 2.2.0 du composant du détecteur IP est disponible.

Composant	Détails
	Nouvelles fonctionnalités • Ajoute la prise en charge de IPv6. Vous pouvez désormais l'utiliser IPv6 pour la messagerie locale.
Authentification de l'appareil client	La version 2.5.1 de l'authentification du périphérique client est disponible. Corrections de bogues et améliorations • Bugs généraux et corrections. • Supporte le point de terminaison FIPS.
Console de débogage locale	La version 2.4.3 de la console de débogage locale est disponible. Corrections de bogues et améliorations • Résout un problème qui affichait de manière incorrecte STREAM_MANAGER_EXPORTER_MAX_BANDWIDTH en Mbps au lieu de bytes/sec

Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.12.6 le 24 mai 2024

Cette version fournit la version 2.12.6 du composant Greengrass Nucleus et des mises à jour des composants fournis. AWS

Date de sortie : 24 mai 2024

Détails de la version

- [Mises à jour des composants publics](#)

Mises à jour des composants publics

Le tableau suivant répertorie les composants fournis par AWS qui incluent des fonctionnalités nouvelles et mises à jour.

 Important

Lorsque vous déployez un composant, AWS IoT Greengrass installe les dernières versions prises en charge de toutes les dépendances de ce composant. De ce fait, les nouvelles versions de correctif des composants publics AWS fournis peuvent être automatiquement déployées sur vos appareils principaux si vous ajoutez de nouveaux appareils à un groupe d'objets ou si vous mettez à jour le déploiement qui cible ces appareils. Certaines mises à jour automatiques, telles que la mise à jour du noyau, peuvent provoquer le redémarrage inattendu de vos appareils.

Pour éviter les mises à jour involontaires d'un composant en cours d'exécution sur votre appareil, nous vous recommandons d'inclure directement votre version préférée de ce composant lorsque vous [créez un déploiement](#). Pour plus d'informations sur le comportement de mise à jour du logiciel AWS IoT Greengrass Core, consultez [Mettre à jour le logiciel AWS IoT Greengrass principal \(OTA\)](#).

Composant	Détails
Noyau de Greengrass	La version 2.12.6 du noyau Greengrass est disponible. Corrections de bugs et améliorations • Résout un problème qui provoquait un crash au démarrage sur certains ARMv8 processeurs, notamment le Jetson Nano.
Greengrass CLI	La version 2.12.6 de la CLI Greengrass est disponible. Corrections de bugs et améliorations • Version mise à jour pour la version 2.12.6 de Greengrass Nucleus.
Directeur secret	La version 2.1.8 du gestionnaire de secrets est disponible. Corrections de bugs et améliorations • Résout un problème selon lequel le gestionnaire de secrets n'accepte pas un ARN partiel.

Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.12.5 le 25 avril 2024

Cette version fournit la version 2.12.5 du composant Greengrass Nucleus et des mises à jour des composants fournis.

Date de sortie : 25 avril 2024

Détails de la version

- [Mises à jour des composants publics](#)

Mises à jour des composants publics

Le tableau suivant répertorie les composants fournis par AWS qui incluent des fonctionnalités nouvelles et mises à jour.

Important

Lorsque vous déployez un composant, AWS IoT Greengrass installe les dernières versions prises en charge de toutes les dépendances de ce composant. De ce fait, les nouvelles versions de correctif des composants publics AWS fournis peuvent être automatiquement déployées sur vos appareils principaux si vous ajoutez de nouveaux appareils à un groupe d'objets ou si vous mettez à jour le déploiement qui cible ces appareils. Certaines mises à jour automatiques, telles que la mise à jour du noyau, peuvent provoquer le redémarrage inattendu de vos appareils.

Pour éviter les mises à jour involontaires d'un composant en cours d'exécution sur votre appareil, nous vous recommandons d'inclure directement votre version préférée de ce composant lorsque vous [créez un déploiement](#). Pour plus d'informations sur le comportement de mise à jour du logiciel AWS IoT Greengrass Core, consultez [Mettre à jour le logiciel AWS IoT Greengrass principal \(OTA\)](#).

Composant	Détails
Noyau de Greengrass	La version 2.12.5 du noyau Greengrass est disponible .

Composant	Détails
	Corrections de bugs et améliorations • Résout un problème qui empêchait parfois l'annulation d'un déploiement lors de la restauration d'un composant précédemment défectueux présentant des dépendances strictes. • Résout un problème selon lequel le noyau ne publie pas de mises à jour de statut après le provisionnement de la flotte. • Ajoute de nouvelles tentatives pour l'GetDeploymentConfiguration API après l'obtention d'erreurs 404.

Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.12.4 le 2 avril 2024

Cette version fournit la version 2.12.4 du composant Greengrass Nucleus et des mises à jour des composants fournis. AWS

Date de sortie : 02 avril 2024

Détails de la version

- [Mises à jour des composants publics](#)

Mises à jour des composants publics

Le tableau suivant répertorie les composants fournis par AWS qui incluent des fonctionnalités nouvelles et mises à jour.

Important

Lorsque vous déployez un composant, AWS IoT Greengrass installe les dernières versions prises en charge de toutes les dépendances de ce composant. De ce fait, les nouvelles versions de correctif des composants publics AWS fournis peuvent être automatiquement déployées sur vos appareils principaux si vous ajoutez de nouveaux appareils à un groupe d'objets ou si vous mettez à jour le déploiement qui cible ces appareils. Certaines mises à jour automatiques, telles que la mise à jour du noyau, peuvent provoquer le redémarrage inattendu de vos appareils.

Pour éviter les mises à jour involontaires d'un composant en cours d'exécution sur votre appareil, nous vous recommandons d'inclure directement votre version préférée de ce composant lorsque vous [créez un déploiement](#). Pour plus d'informations sur le comportement de mise à jour du logiciel AWS IoT Greengrass Core, consultez [Mettre à jour le logiciel AWS IoT Greengrass principal \(OTA\)](#).

Composant	Détails
Noyau de Greengrass	La version 2.12.4 du noyau Greengrass est disponible. Corrections de bugs et améliorations • Résout un problème de blocage du noyau lors du démarrage sur certains appareils Linux.

Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.12.3 le 27 mars 2024

Cette version fournit la version 2.12.3 du composant Greengrass Nucleus et des mises à jour des composants fournis. AWS

Date de sortie : 27 mars 2024

Détails de la version

- [Mises à jour des composants publics](#)

Mises à jour des composants publics

Le tableau suivant répertorie les composants fournis par AWS qui incluent des fonctionnalités nouvelles et mises à jour.

Important

Lorsque vous déployez un composant, AWS IoT Greengrass installe les dernières versions prises en charge de toutes les dépendances de ce composant. De ce fait, les nouvelles versions de correctif des composants publics AWS fournis peuvent être automatiquement

déployées sur vos appareils principaux si vous ajoutez de nouveaux appareils à un groupe d'objets ou si vous mettez à jour le déploiement qui cible ces appareils. Certaines mises à jour automatiques, telles que la mise à jour du noyau, peuvent provoquer le redémarrage inattendu de vos appareils.

Pour éviter les mises à jour involontaires d'un composant en cours d'exécution sur votre appareil, nous vous recommandons d'inclure directement votre version préférée de ce composant lorsque vous [créez un déploiement](#). Pour plus d'informations sur le comportement de mise à jour du logiciel AWS IoT Greengrass Core, consultez [Mettre à jour le logiciel AWS IoT Greengrass principal \(OTA\)](#).

Composant	Détails
Noyau de Greengrass	La version 2.12.3 du noyau Greengrass est disponible. Corrections de bugs et améliorations • Résout un problème selon lequel le noyau ne signalait pas l'état correct des composants après le redémarrage du noyau et pendant la restauration des composants. • Correction et amélioration de bogues généraux
Shadow Manager	La version 2.3.7 du composant Shadow Manager est disponible. Corrections de bugs et améliorations Résout un problème selon lequel le Shadow Manager enregistre régulièrement une <code>NullPointerException</code> erreur lors d'une synchronisation du Shadow Manager.
Approvisionnement de flotte	La version 1.2.1 du plugin de provisionnement de AWS IoT flotte est disponible. Corrections de bugs et améliorations Résout un problème en raison duquel le plug-in de provisionnement de flotte est hors ligne lors du démarrage du noyau Greengrass. Le plugin de provisionnement de flotte réessaie désormais indéfiniment les appels MQTT connect.

Composant	Détails
Détecteur IP	La version 2.1.9 du composant spouleur de disque est disponible. Corrections de bugs et améliorations Ajuste l'étape d'acquisition de l'adresse IP pour envoyer uniquement les journaux au niveau du journal de débogage.
Composant de broker Moquette MQTT 3.1.1	La version 2.3.6 du composant broker Moquette MQTT 3.1.1 est disponible. Corrections de bugs et améliorations Correction et amélioration de bogues généraux
Gestionnaire Lambda	La version 2.3.3 du composant Lambda Manager est disponible. Corrections de bugs et améliorations Correction et amélioration de bogues généraux
Console de débogage locale	La version 2.4.2 du composant de console de débogage local est disponible. Corrections de bugs et améliorations Correction et amélioration de bogues généraux

Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.12.2 le 15 février 2024

Cette version fournit la version 2.12.2 du composant Greengrass Nucleus et des mises à jour des composants fournis. AWS

Date de sortie : 15 février 2024

Détails de la version

- [Mises à jour des composants publics](#)

Mises à jour des composants publics

Le tableau suivant répertorie les composants fournis par AWS qui incluent des fonctionnalités nouvelles et mises à jour.

Important

Lorsque vous déployez un composant, AWS IoT Greengrass installe les dernières versions prises en charge de toutes les dépendances de ce composant. De ce fait, les nouvelles versions de correctif des composants publics AWS fournis peuvent être automatiquement déployées sur vos appareils principaux si vous ajoutez de nouveaux appareils à un groupe d'objets ou si vous mettez à jour le déploiement qui cible ces appareils. Certaines mises à jour automatiques, telles que la mise à jour du noyau, peuvent provoquer le redémarrage inattendu de vos appareils.

Pour éviter les mises à jour involontaires d'un composant en cours d'exécution sur votre appareil, nous vous recommandons d'inclure directement votre version préférée de ce composant lorsque vous [créez un déploiement](#). Pour plus d'informations sur le comportement de mise à jour du logiciel AWS IoT Greengrass Core, consultez [Mettre à jour le logiciel AWS IoT Greengrass principal \(OTA\)](#).

Composant	Détails
Noyau de Greengrass	La version 2.12.2 du noyau Greengrass est disponible. Corrections de bugs et améliorations • Résout un problème en raison duquel les anciens journaux n'étaient pas nettoyés correctement. • Correction et amélioration de bogues généraux
Shadow Manager	La version 2.3.6 du composant Shadow Manager est disponible. Corrections de bugs et améliorations Résout un problème selon lequel les propriétés d'ombre supprimées par le biais de AWS Cloud mises à jour alors que l'appareil est hors ligne continuent d'exister dans l'ombre locale une fois la connectivité rétablie.

Composant	Détails
Lanceur Lambda	La version 2.0.13 du composant Lambda Launcher est disponible. Corrections de bugs et améliorations Correction et amélioration de bogues généraux
spouleur à disque	La version 1.0.3 du composant spouleur de disque est disponible. Corrections de bugs et améliorations Améliore les performances en réutilisant les connexions aux bases de données.

Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.12.1 le 8 décembre 2023

Cette version fournit la version 2.12.1 du composant Greengrass Nucleus et des mises à jour des composants fournis. AWS

Date de sortie : 8 décembre 2023

Détails de la version

- [Mises à jour des composants publics](#)

Mises à jour des composants publics

Le tableau suivant répertorie les composants fournis par AWS qui incluent des fonctionnalités nouvelles et mises à jour.

Important

Lorsque vous déployez un composant, AWS IoT Greengrass installe les dernières versions prises en charge de toutes les dépendances de ce composant. De ce fait, les nouvelles versions de correctif des composants publics AWS fournis peuvent être automatiquement déployées sur vos appareils principaux si vous ajoutez de nouveaux appareils à un groupe d'objets ou si vous mettez à jour le déploiement qui cible ces appareils. Certaines mises à

jour automatiques, telles que la mise à jour du noyau, peuvent provoquer le redémarrage inattendu de vos appareils.

Pour éviter les mises à jour involontaires d'un composant en cours d'exécution sur votre appareil, nous vous recommandons d'inclure directement votre version préférée de ce composant lorsque vous [créez un déploiement](#). Pour plus d'informations sur le comportement de mise à jour du logiciel AWS IoT Greengrass Core, consultez [Mettre à jour le logiciel AWS IoT Greengrass principal \(OTA\)](#).

Composant	Détails
Noyau de Greengrass	La version 2.12.1 du noyau Greengrass est disponible. Corrections de bogues et améliorations • Résout un problème selon lequel le noyau pouvait dupliquer les abonnements MQTT aux sujets de déploiement, ce qui entraînait une journalisation supplémentaire et des publications MQTT.
Authentification de l'appareil client	La version 2.4.5 du composant d'authentification de l'appareil client est disponible. Nouvelles fonctionnalités Ajoute la prise en charge des préfixes génériques pour sélectionner des noms d'objets avec le <code>selectionRule</code> paramètre. Corrections de bogues et améliorations Résout un problème selon lequel les certificats ne sont pas mis à jour avec les nouvelles informations de connectivité dans certains cas.
spouleur à disque	La version 1.0.2 du composant Disk Spooler est disponible. Corrections de bogues et améliorations Résout un problème en raison duquel le champ de format de message MQTT n'est pas conservé dans certains cas.
Pont MQTT	La version 2.3.1 du composant spouleur de disque est disponible.

Composant	Détails
	Corrections de bogues et améliorations Résout un problème où le client MQTT local entre dans une boucle de déconnexion.
Gestionnaire de flux	La version 2.1.12 du composant du gestionnaire de flux est disponible. Corrections de bogues et améliorations Met à jour l'ordre dans lequel les informations d'identification sont utilisées afin que les informations d'identification Greengrass soient préférées pour les demandes de AWS service.

Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.12.0 le 7 novembre 2023

Cette version fournit la version 2.12.0 du composant Greengrass Nucleus et des mises à jour des composants fournis. AWS

Date de sortie : 7 novembre 2023

Points forts de la publication

- Bootstrap on rollback : fournit AWS IoT Greengrass désormais un paramètre de configuration du noyau Greengrass appelé. `BootstrapOnRollback` Cette fonctionnalité vous permet d'exécuter les étapes du cycle de vie du bootstrap dans le cadre d'un déploiement rétroactif.

Détails de la version

- [Mises à jour des composants publics](#)

Mises à jour des composants publics

Le tableau suivant répertorie les composants fournis par AWS qui incluent des fonctionnalités nouvelles et mises à jour.

⚠ Important

Lorsque vous déployez un composant, AWS IoT Greengrass installe les dernières versions prises en charge de toutes les dépendances de ce composant. De ce fait, les nouvelles versions de correctif des composants publics AWS fournis peuvent être automatiquement déployées sur vos appareils principaux si vous ajoutez de nouveaux appareils à un groupe d'objets ou si vous mettez à jour le déploiement qui cible ces appareils. Certaines mises à jour automatiques, telles que la mise à jour du noyau, peuvent provoquer le redémarrage inattendu de vos appareils.

Pour éviter les mises à jour involontaires d'un composant en cours d'exécution sur votre appareil, nous vous recommandons d'inclure directement votre version préférée de ce composant lorsque vous [créez un déploiement](#). Pour plus d'informations sur le comportement de mise à jour du logiciel AWS IoT Greengrass Core, consultez [Mettre à jour le logiciel AWS IoT Greengrass principal \(OTA\)](#).

Composant	Détails
Noyau de Greengrass	La version 2.12.0 du noyau Greengrass est disponible. Nouvelles fonctionnalités • Vous permet d'exécuter les étapes du cycle de vie du bootstrap dans le cadre d'un déploiement rétroactif.

Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.11.3 le 18 octobre 2023

Cette version fournit la version 2.11.3 du composant Greengrass nucleus.

Date de parution : 18 octobre 2023

Détails de la version

- [Mises à jour des composants publics](#)

Mises à jour des composants publics

Le tableau suivant répertorie les composants fournis par AWS qui incluent des fonctionnalités nouvelles et mises à jour.

Important

Lorsque vous déployez un composant, AWS IoT Greengrass installe les dernières versions prises en charge de toutes les dépendances de ce composant. De ce fait, les nouvelles versions de correctif des composants publics AWS fournis peuvent être automatiquement déployées sur vos appareils principaux si vous ajoutez de nouveaux appareils à un groupe d'objets ou si vous mettez à jour le déploiement qui cible ces appareils. Certaines mises à jour automatiques, telles que la mise à jour du noyau, peuvent provoquer le redémarrage inattendu de vos appareils.

Pour éviter les mises à jour involontaires d'un composant en cours d'exécution sur votre appareil, nous vous recommandons d'inclure directement votre version préférée de ce composant lorsque vous [créez un déploiement](#). Pour plus d'informations sur le comportement de mise à jour du logiciel AWS IoT Greengrass Core, consultez [Mettre à jour le logiciel AWS IoT Greengrass principal \(OTA\)](#).

Composant	Détails
Noyau de Greengrass	La version 2.11.3 du noyau Greengrass est disponible. Corrections de bugs et améliorations • Résout un problème dans le noyau qui pouvait entraîner le démarrage incorrect d'un composant en cas d'échec de ses dépendances. Nouvelles fonctionnalités • Ajoute un type de point de terminaison s3 configurable.
Gestionnaire Lambda	La version 2.3.1 du composant Lambda Manager est disponible. Corrections de bugs et améliorations • Ajuste les niveaux de journalisation pour certaines erreurs.

Composant	Détails
Console de débogage locale	La version 2.4.0 du composant Lambda Manager est disponible. Nouvelles fonctionnalités • Ajoute une console de débogage au gestionnaire de flux.
Gestionnaire de journaux	La version 2.3.6 du composant du gestionnaire de journaux est disponible. Corrections de bugs et améliorations • Ajuste les niveaux de journalisation pour certaines erreurs.
Shadow Manager	La version 2.3.4 du composant Shadow Manager est disponible. Corrections de bugs et améliorations • Ajoute la prise en charge des documents à état fictif nuls et vides.

Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.11.2 le 9 août 2023

Cette version fournit la version 2.11.2 du composant Greengrass nucleus.

Date de sortie : 9 août 2023

Détails de la version

- [Mises à jour des composants publics](#)

Mises à jour des composants publics

Le tableau suivant répertorie les composants fournis par AWS qui incluent des fonctionnalités nouvelles et mises à jour.

Important

Lorsque vous déployez un composant, AWS IoT Greengrass installe les dernières versions prises en charge de toutes les dépendances de ce composant. De ce fait, les nouvelles versions de correctif des composants publics AWS fournis peuvent être automatiquement

déployées sur vos appareils principaux si vous ajoutez de nouveaux appareils à un groupe d'objets ou si vous mettez à jour le déploiement qui cible ces appareils. Certaines mises à jour automatiques, telles que la mise à jour du noyau, peuvent provoquer le redémarrage inattendu de vos appareils.

Pour éviter les mises à jour involontaires d'un composant en cours d'exécution sur votre appareil, nous vous recommandons d'inclure directement votre version préférée de ce composant lorsque vous [créez un déploiement](#). Pour plus d'informations sur le comportement de mise à jour du logiciel AWS IoT Greengrass Core, consultez [Mettre à jour le logiciel AWS IoT Greengrass principal \(OTA\)](#).

Composant	Détails
Noyau de Greengrass	La version 2.11.2 du noyau Greengrass est disponible. Corrections de bogues et améliorations • Résout un problème dans le client Nucleus MQTT 5 qui pouvait apparaître hors ligne lorsqu'un grand nombre (> 50) d'abonnements sont utilisés. • Ajoute une nouvelle tentative en cas d'échec du protocole TCP du docker Dial.

Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.11.1 le 21 juillet 2023

Cette version fournit la version 2.11.1 du composant Greengrass nucleus.

Date de sortie : 21 juillet 2023

Détails de la version

- [Mises à jour des composants publics](#)

Mises à jour des composants publics

Le tableau suivant répertorie les composants fournis par AWS qui incluent des fonctionnalités nouvelles et mises à jour.

Important

Lorsque vous déployez un composant, AWS IoT Greengrass installe les dernières versions prises en charge de toutes les dépendances de ce composant. De ce fait, les nouvelles versions de correctif des composants publics AWS fournis peuvent être automatiquement déployées sur vos appareils principaux si vous ajoutez de nouveaux appareils à un groupe d'objets ou si vous mettez à jour le déploiement qui cible ces appareils. Certaines mises à jour automatiques, telles que la mise à jour du noyau, peuvent provoquer le redémarrage inattendu de vos appareils.

Pour éviter les mises à jour involontaires d'un composant en cours d'exécution sur votre appareil, nous vous recommandons d'inclure directement votre version préférée de ce composant lorsque vous [créez un déploiement](#). Pour plus d'informations sur le comportement de mise à jour du logiciel AWS IoT Greengrass Core, consultez [Mettre à jour le logiciel AWS IoT Greengrass principal \(OTA\)](#).

Composant	Détails
Noyau de Greengrass	La version 2.11.1 du noyau Greengrass est disponible. Corrections de bogues et améliorations • Résout un problème selon lequel le noyau ne démarre pas en cas d'échec d'une tâche de démarrage et d'endommagement du fichier de métadonnées de déploiement. • Résout un problème selon lequel les composants Lambda à la demande ne sont pas signalés dans les mises à jour de l'état du déploiement. • Ajoute la prise en charge de la politique d'autorisation dupliquée IDs.
Gestionnaire Lambda	La version 2.2.11 du gestionnaire Lambda est disponible.

Composant	Détails
	Corrections de bogues et améliorations • Résout un problème selon lequel la LegacySubscriptionRouter configuration ne se met pas à jour lorsque la configuration Lambda change.

Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.11.0 le 28 juin 2023

Cette version fournit la version 2.11.0 du composant Greengrass nucleus.

Date de sortie : 28 juin 2023

Points forts de la publication

- Spouleur de disque persistant : fournit AWS IoT Greengrass désormais une implémentation de spouleur persistant pour les messages envoyés depuis les appareils principaux de Greengrass vers AWS IoT Core. Ce composant stockera ces messages sortants sur le disque. Pour de plus amples informations, veuillez consulter [spouleur à disque](#).
- Améliorations du déploiement local — Vous pouvez désormais annuler les déploiements locaux, définir des politiques de gestion des échecs de déploiement et obtenir l'état détaillé du déploiement.
- Améliorations de la vitesse de journalisation — Les vitesses de téléchargement des journaux pour le composant du gestionnaire de journaux ont été améliorées.

Détails de la version

- [Mises à jour des composants publics](#)

Mises à jour des composants publics

Le tableau suivant répertorie les composants fournis par AWS qui incluent des fonctionnalités nouvelles et mises à jour.

⚠ Important

Lorsque vous déployez un composant, AWS IoT Greengrass installe les dernières versions prises en charge de toutes les dépendances de ce composant. De ce fait, les nouvelles versions de correctif des composants publics AWS fournis peuvent être automatiquement déployées sur vos appareils principaux si vous ajoutez de nouveaux appareils à un groupe d'objets ou si vous mettez à jour le déploiement qui cible ces appareils. Certaines mises à jour automatiques, telles que la mise à jour du noyau, peuvent provoquer le redémarrage inattendu de vos appareils.

Pour éviter les mises à jour involontaires d'un composant en cours d'exécution sur votre appareil, nous vous recommandons d'inclure directement votre version préférée de ce composant lorsque vous [créez un déploiement](#). Pour plus d'informations sur le comportement de mise à jour du logiciel AWS IoT Greengrass Core, consultez [Mettre à jour le logiciel AWS IoT Greengrass principal \(OTA\)](#).

Composant	Détails
Noyau de Greengrass	La version 2.11.0 du noyau Greengrass est disponible. Nouvelles fonctionnalités • Vous permet d'annuler un déploiement local. • Vous permet de configurer une politique de gestion des défaillances pour un déploiement local. • Ajoute la prise en charge d'un plugin de spouleur de disque.
Greengrass CLI	La version 2.11.0 de la CLI Greengrass est disponible. Nouvelles fonctionnalités • Vous permet d'annuler un déploiement local. • Vous permet de configurer une politique de gestion des défaillances pour un déploiement local. • Améliore les rapports détaillés sur l'état du déploiement.
spouleur à disque	La version 1.0.0 du composant spouleur de disque est disponible.

Composant	Détails
	Le composant Disk Spooler assure le stockage permanent des messages envoyés par les principaux appareils de Greengrass à. AWS IoT Core
Gestionnaire de journaux	La version 2.3.5 du composant du gestionnaire de journaux est disponible. Améliorations Améliore la vitesse de téléchargement des journaux.

Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.10.3 le 21 juin 2023

Cette version fournit la version 2.10.3 du composant Greengrass nucleus.

Date de sortie : 21 juin 2023

Détails de la version

- [Mises à jour des composants publics](#)

Mises à jour des composants publics

Le tableau suivant répertorie les composants fournis par AWS qui incluent des fonctionnalités nouvelles et mises à jour.

Important

Lorsque vous déployez un composant, AWS IoT Greengrass installe les dernières versions prises en charge de toutes les dépendances de ce composant. De ce fait, les nouvelles versions de correctif des composants publics AWS fournis peuvent être automatiquement déployées sur vos appareils principaux si vous ajoutez de nouveaux appareils à un groupe d'objets ou si vous mettez à jour le déploiement qui cible ces appareils. Certaines mises à jour automatiques, telles que la mise à jour du noyau, peuvent provoquer le redémarrage inattendu de vos appareils.

Pour éviter les mises à jour involontaires d'un composant en cours d'exécution sur votre appareil, nous vous recommandons d'inclure directement votre version préférée de ce

composant lorsque vous [créez un déploiement](#). Pour plus d'informations sur le comportement de mise à jour du logiciel AWS IoT Greengrass Core, consultez [Mettre à jour le logiciel AWS IoT Greengrass principal \(OTA\)](#).

Composant	Détails
Noyau de Greengrass	La version 2.10.3 du noyau Greengrass est disponible. Corrections de bogues et améliorations • Résout un problème en raison duquel Greengrass ne s'abonne pas aux notifications de déploiement lorsqu'il utilise le fournisseur PKCS #11.

Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.10.2 le 5 juin 2023

Cette version fournit la version 2.10.2 du composant Greengrass nucleus.

Date de sortie : 5 juin 2023

Détails de la version

- [Mises à jour des composants publics](#)

Mises à jour des composants publics

Le tableau suivant répertorie les composants fournis par AWS qui incluent des fonctionnalités nouvelles et mises à jour.

Important

Lorsque vous déployez un composant, AWS IoT Greengrass installe les dernières versions prises en charge de toutes les dépendances de ce composant. De ce fait, les nouvelles versions de correctif des composants publics AWS fournis peuvent être automatiquement déployées sur vos appareils principaux si vous ajoutez de nouveaux appareils à un groupe d'objets ou si vous mettez à jour le déploiement qui cible ces appareils. Certaines mises à

jour automatiques, telles que la mise à jour du noyau, peuvent provoquer le redémarrage inattendu de vos appareils.

Pour éviter les mises à jour involontaires d'un composant en cours d'exécution sur votre appareil, nous vous recommandons d'inclure directement votre version préférée de ce composant lorsque vous [créez un déploiement](#). Pour plus d'informations sur le comportement de mise à jour du logiciel AWS IoT Greengrass Core, consultez [Mettre à jour le logiciel AWS IoT Greengrass principal \(OTA\)](#).

Composant	Détails
Noyau de Greengrass	La version 2.10.2 du noyau Greengrass est disponible. Corrections de bogues et améliorations • Permet d'analyser le cycle de vie des composants sans distinction majuscules/minuscules. • Résout un problème en raison duquel la variable d'environnement PATH n'était pas recrée correctement. • Corrige le codage de l'URI du proxy pour les composants, y compris le gestionnaire de flux pour les noms d'utilisateur contenant des caractères spéciaux.
Authentification de l'appareil client	La version 2.4.2 du composant d'authentification de l'appareil client est disponible. Nouvelles fonctionnalités Ajoute une nouvelle option <code>startupTimeoutSeconds</code> de configuration.
Gestionnaire Lambda	La version 2.2.9 du composant Lambda Manager est disponible. Corrections de bogues et améliorations Résout un problème d'altération du numéro de port en raison d'une horloge asymétrique.

Composant	Détails
Gestionnaire de journaux	La version 2.3.4 du composant du gestionnaire de journaux est disponible. Corrections de bogues et améliorations • Permet de définir le <code>periodicUploadIntervalSec</code> paramètre sur des valeurs fractionnaires. Le minimum est de 1 microseconde. • Résout un problème en raison duquel le gestionnaire de journaux ne respectait pas les <code>CloudWatch putLogEvents</code> limites.
Courtier MQTT 3.1 (Moquette)	La version 2.3.3 du composant broker MQTT 3.1 (Moquette) est disponible. Nouvelles fonctionnalités Ajoute une nouvelle option <code>startupTimeoutSeconds</code> de configuration.
Pont MQTT	La version 2.2.6 du composant de pont MQTT est disponible. Nouvelles fonctionnalités Ajoute une nouvelle option <code>startupTimeoutSeconds</code> de configuration.
Gestionnaire de flux	La version 2.1.7 du composant du gestionnaire de flux est disponible. Corrections de bogues et améliorations Résout un problème en raison duquel le gestionnaire de flux ne lisait pas correctement la configuration du proxy.

Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.10.1 le 11 mai 2023

Cette version fournit la version 2.10.1 du composant Greengrass nucleus.

Date de sortie : 11 mai 2023

Détails de la version

- [Mises à jour des composants publics](#)

Mises à jour des composants publics

Le tableau suivant répertorie les composants fournis par AWS qui incluent des fonctionnalités nouvelles et mises à jour.

Important

Lorsque vous déployez un composant, AWS IoT Greengrass installe les dernières versions prises en charge de toutes les dépendances de ce composant. De ce fait, les nouvelles versions de correctif des composants publics AWS fournis peuvent être automatiquement déployées sur vos appareils principaux si vous ajoutez de nouveaux appareils à un groupe d'objets ou si vous mettez à jour le déploiement qui cible ces appareils. Certaines mises à jour automatiques, telles que la mise à jour du noyau, peuvent provoquer le redémarrage inattendu de vos appareils.

Pour éviter les mises à jour involontaires d'un composant en cours d'exécution sur votre appareil, nous vous recommandons d'inclure directement votre version préférée de ce composant lorsque vous [créez un déploiement](#). Pour plus d'informations sur le comportement de mise à jour du logiciel AWS IoT Greengrass Core, consultez [Mettre à jour le logiciel AWS IoT Greengrass principal \(OTA\)](#).

Composant	Détails
Noyau de Greengrass	La version 2.10.1 du noyau Greengrass est disponible. Corrections de bugs et améliorations • Résout un problème susceptible de provoquer un crash au démarrage sur certains ARMv8 processeurs, notamment le Jetson Nano. • Greengrass ne ferme plus le standard d'un composant, ce qui rétablit le comportement d'avant la version 2.10.0
Gestionnaire de flux	La version 2.1.6 du nouveau gestionnaire de flux est disponible.

Composant	Détails
	Corrections de bugs et améliorations Résout un problème susceptible de provoquer un crash au démarrage sur certains ARMv8 processeurs, notamment le Jetson Nano.

Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.10.0 le 9 mai 2023

Cette version fournit la version 2.10.0 du composant Greengrass Nucleus et des mises à jour des composants fournis. AWS

Date de sortie : 9 mai 2023

Points forts de la publication

- MQTT5 support — prend AWS IoT Greengrass désormais en charge l'envoi et la réception de messages AWS IoT Core en utilisant MQTT5. Pour plus d'informations, consultez [Publier des messages AWS IoT Core MQTT](#).

Détails de la version

- [Mises à jour des composants publics](#)

Mises à jour des composants publics

Le tableau suivant répertorie les composants fournis par AWS qui incluent des fonctionnalités nouvelles et mises à jour.

Important

Lorsque vous déployez un composant, AWS IoT Greengrass installe les dernières versions prises en charge de toutes les dépendances de ce composant. De ce fait, les nouvelles versions de correctif des composants publics AWS fournis peuvent être automatiquement déployées sur vos appareils principaux si vous ajoutez de nouveaux appareils à un groupe d'objets ou si vous mettez à jour le déploiement qui cible ces appareils. Certaines mises à

jour automatiques, telles que la mise à jour du noyau, peuvent provoquer le redémarrage inattendu de vos appareils.

Pour éviter les mises à jour involontaires d'un composant en cours d'exécution sur votre appareil, nous vous recommandons d'inclure directement votre version préférée de ce composant lorsque vous [créez un déploiement](#). Pour plus d'informations sur le comportement de mise à jour du logiciel AWS IoT Greengrass Core, consultez [Mettre à jour le logiciel AWS IoT Greengrass principal \(OTA\)](#).

Composant	Détails
Noyau de Greengrass	La version 2.10.0 du noyau Greengrass est disponible. Nouvelles fonctionnalités • Ajoute la <code>interpolateComponentConfiguration</code> prise en charge de l'expression régulière vide. Greengrass interpole désormais à partir de l'objet de configuration racine. • Ajoute le support pour MQTT5. • Ajoute un mécanisme permettant de charger rapidement les composants du plugin sans les scanner. • Permet à Greengrass d'économiser de l'espace disque en supprimant les images Docker inutilisées. Corrections de bugs et améliorations • Résout un problème selon lequel la restauration laisse certaines valeurs de configuration inchangées lors d'un déploiement. • Résout un problème à cause duquel le noyau Greengrass valide une séquence de AWS domaine dans des points de terminaison de AWS données et des informations d'identification personnalisés. • Met à jour la résolution des dépendances multigroupes afin de résoudre à nouveau toutes les dépendances de groupe par le biais de AWS Cloud la négociation, au lieu de se limiter à la version active. Cette mise à jour supprime également le code d'erreur de déploiement <code>INSTALLED_COMPONENT_NOT_FOUND</code>. • Met à jour le noyau Greengrass pour éviter de télécharger des images Docker lorsqu'elles existent déjà localement.

Composant	Détails
	• Met à jour le noyau Greengrass pour redémarrer une étape d'installation d'un composant avant l'expiration du délai imparti. • Corrections et améliorations mineures supplémentaires.
Shadow Manager	La version 2.3.2 du nouveau gestionnaire d'ombres est disponible. Corrections de bugs et améliorations Résout un problème selon lequel Shadow Manager passe à l'<code>BROKEN</code> état lorsque la base de données parallèle locale est corrompue.

Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.9.6 le 20 avril 2023

Cette version fournit la version 2.9.6 du composant Greengrass nucleus.

Date de version : 20 avril 2023

Détails de la version

- [Mises à jour des composants publics](#)

Mises à jour des composants publics

Le tableau suivant répertorie les composants fournis par AWS qui incluent des fonctionnalités nouvelles et mises à jour.

Important

Lorsque vous déployez un composant, AWS IoT Greengrass installe les dernières versions prises en charge de toutes les dépendances de ce composant. De ce fait, les nouvelles versions de correctif des composants publics AWS fournis peuvent être automatiquement déployées sur vos appareils principaux si vous ajoutez de nouveaux appareils à un groupe d'objets ou si vous mettez à jour le déploiement qui cible ces appareils. Certaines mises à jour automatiques, telles que la mise à jour du noyau, peuvent provoquer le redémarrage inattendu de vos appareils.

Pour éviter les mises à jour involontaires d'un composant en cours d'exécution sur votre appareil, nous vous recommandons d'inclure directement votre version préférée de ce composant lorsque vous [créez un déploiement](#). Pour plus d'informations sur le comportement de mise à jour du logiciel AWS IoT Greengrass Core, consultez [Mettre à jour le logiciel AWS IoT Greengrass principal \(OTA\)](#).

Composant	Détails
Noyau de Greengrass	La version 2.9.6 du noyau Greengrass est disponible. Corrections de bugs et améliorations • Résout un problème selon lequel un déploiement de Greengrass échoue avec l'erreur LAUNCH_DIRECTORY_CORRUPTED et un redémarrage ultérieur de l'appareil ne permet pas de démarrer Greengrass. Cette erreur peut se produire lorsque vous déplacez l'appareil Greengrass entre plusieurs groupes d'objets lors de déploiements nécessitant le redémarrage de Greengrass.

Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.9.5 le 30 mars 2023

Cette version fournit la version 2.9.5 du composant Greengrass nucleus.

Date de sortie : 30 mars 2023

Détails de la version

- [Mises à jour des composants publics](#)

Mises à jour des composants publics

Le tableau suivant répertorie les composants fournis par AWS qui incluent des fonctionnalités nouvelles et mises à jour.

⚠ Important

Lorsque vous déployez un composant, AWS IoT Greengrass installe les dernières versions prises en charge de toutes les dépendances de ce composant. De ce fait, les nouvelles versions de correctif des composants publics AWS fournis peuvent être automatiquement déployées sur vos appareils principaux si vous ajoutez de nouveaux appareils à un groupe d'objets ou si vous mettez à jour le déploiement qui cible ces appareils. Certaines mises à jour automatiques, telles que la mise à jour du noyau, peuvent provoquer le redémarrage inattendu de vos appareils.

Pour éviter les mises à jour involontaires d'un composant en cours d'exécution sur votre appareil, nous vous recommandons d'inclure directement votre version préférée de ce composant lorsque vous [créez un déploiement](#). Pour plus d'informations sur le comportement de mise à jour du logiciel AWS IoT Greengrass Core, consultez [Mettre à jour le logiciel AWS IoT Greengrass principal \(OTA\)](#).

Composant	Détails
Noyau de Greengrass	La version 2.9.5 du noyau Greengrass est disponible. Nouvelles fonctionnalités • Ajoute la prise en charge de la vérification des signatures du logiciel Greengrass Nucleus. Corrections de bugs et améliorations • Résout un problème d'échec d'un déploiement lorsque la région de métadonnées de recette locale ne correspond pas à la région de lancement du noyau Greengrass. Le noyau Greengrass renégocie désormais avec le cloud lorsque cela se produit. • Résout un problème à cause duquel le spouleur de messages MQTT se remplit et ne supprime jamais les messages. • Corrections et améliorations mineures supplémentaires.

Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.9.4 le 24 février 2023

Cette version fournit la version 2.9.4 du composant Greengrass nucleus.

Date de sortie : 24 février 2023

Détails de la version

- [Mises à jour des composants publics](#)

Mises à jour des composants publics

Le tableau suivant répertorie les composants fournis par AWS qui incluent des fonctionnalités nouvelles et mises à jour.

Important

Lorsque vous déployez un composant, AWS IoT Greengrass installe les dernières versions prises en charge de toutes les dépendances de ce composant. De ce fait, les nouvelles versions de correctif des composants publics AWS fournis peuvent être automatiquement déployées sur vos appareils principaux si vous ajoutez de nouveaux appareils à un groupe d'objets ou si vous mettez à jour le déploiement qui cible ces appareils. Certaines mises à jour automatiques, telles que la mise à jour du noyau, peuvent provoquer le redémarrage inattendu de vos appareils.

Pour éviter les mises à jour involontaires d'un composant en cours d'exécution sur votre appareil, nous vous recommandons d'inclure directement votre version préférée de ce composant lorsque vous [créez un déploiement](#). Pour plus d'informations sur le comportement de mise à jour du logiciel AWS IoT Greengrass Core, consultez [Mettre à jour le logiciel AWS IoT Greengrass principal \(OTA\)](#).

Composant	Détails
Noyau de Greengrass	La version 2.9.4 du noyau Greengrass est disponible .

Composant	Détails
	Corrections de bogues et améliorations • Vérifie la présence d'un message nul avant de supprimer les messages QOS 0. • Tronque les valeurs détaillées du statut de la tâche si elles dépassent la limite de 1024 caractères. • Met à jour le script bootstrap pour Windows afin de lire correctement le chemin racine de Greengrass si ce chemin inclut des espaces. • Met à jour l'abonnement AWS IoT Core afin de supprimer les messages des clients si la réponse à l'abonnement n'a pas été envoyée. • Garantit que le noyau charge sa configuration à partir de fichiers de sauvegarde lorsque le fichier de configuration principal est endommagé ou manquant.

Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.9.3 le 1er février 2023

Cette version fournit la version 2.9.3 du composant Greengrass nucleus.

Date de sortie : 01 février 2023

Détails de la version

- [Mises à jour des composants publics](#)

Mises à jour des composants publics

Le tableau suivant répertorie les composants fournis par AWS qui incluent des fonctionnalités nouvelles et mises à jour.

Important

Lorsque vous déployez un composant, AWS IoT Greengrass installe les dernières versions prises en charge de toutes les dépendances de ce composant. De ce fait, les nouvelles versions de correctif des composants publics AWS fournis peuvent être automatiquement

déployées sur vos appareils principaux si vous ajoutez de nouveaux appareils à un groupe d'objets ou si vous mettez à jour le déploiement qui cible ces appareils. Certaines mises à jour automatiques, telles que la mise à jour du noyau, peuvent provoquer le redémarrage inattendu de vos appareils.

Pour éviter les mises à jour involontaires d'un composant en cours d'exécution sur votre appareil, nous vous recommandons d'inclure directement votre version préférée de ce composant lorsque vous [créez un déploiement](#). Pour plus d'informations sur le comportement de mise à jour du logiciel AWS IoT Greengrass Core, consultez [Mettre à jour le logiciel AWS IoT Greengrass principal \(OTA\)](#).

Composant	Détails
Noyau de Greengrass	La version 2.9.3 du noyau Greengrass est disponible. Corrections de bugs et améliorations • Garantit que les clients MQTT IDs ne sont pas dupliqués. • Améliore la lecture et l'écriture de fichiers afin d'éviter la corruption et de récupérer les données en cas de corruption. • Réessaie docker image pull en cas d'erreurs spécifiques liées au réseau. • Ajoute l'<code>noProxyAddresses</code> option de connexion MQTT.

Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.9.2 le 22 décembre 2022

Cette version fournit la version 2.9.2 du composant Greengrass nucleus.

Date de sortie : 22 décembre 2022

Détails de la version

- [Mises à jour des composants publics](#)

Mises à jour des composants publics

Le tableau suivant répertorie les composants AWS fournis qui incluent des fonctionnalités nouvelles et mises à jour.

Important

Lorsque vous déployez un composant, AWS IoT Greengrass installe les dernières versions prises en charge de toutes les dépendances de ce composant. De ce fait, les nouvelles versions de correctif des composants publics AWS fournis peuvent être automatiquement déployées sur vos appareils principaux si vous ajoutez de nouveaux appareils à un groupe d'objets ou si vous mettez à jour le déploiement qui cible ces appareils. Certaines mises à jour automatiques, telles que la mise à jour du noyau, peuvent provoquer le redémarrage inattendu de vos appareils.

Pour éviter les mises à jour involontaires d'un composant en cours d'exécution sur votre appareil, nous vous recommandons d'inclure directement votre version préférée de ce composant lorsque vous [créez un déploiement](#). Pour plus d'informations sur le comportement de mise à jour du logiciel AWS IoT Greengrass Core, consultez [Mettre à jour le logiciel AWS IoT Greengrass principal \(OTA\)](#).

Composant	Détails
Noyau de Greengrass	La version 2.9.2 du noyau Greengrass est disponible. Corrections de bugs et améliorations • Résout un problème selon lequel la configuration <code>interpolateComponentConfiguration</code> ne s'appliquait pas à un déploiement en cours. • Utilise OSHI pour répertorier tous les processus enfants.

Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.9.1 le 18 novembre 2022

Cette version fournit la version 2.9.1 du composant Greengrass Nucleus et des mises à jour des composants fournis.

Date de parution : 18 novembre 2022

Points forts de la publication

- Gestionnaire de journaux — Le gestionnaire de journaux traite et télécharge désormais directement les fichiers journaux actifs au lieu d'attendre la rotation des nouveaux fichiers. Cette amélioration réduit considérablement les délais de journalisation. Pour plus d'informations, voir [Gestionnaire de journaux](#)

Détails de la version

- [Mises à jour des composants publics](#)

Mises à jour des composants publics

Le tableau suivant répertorie les composants AWS fournis qui incluent des fonctionnalités nouvelles et mises à jour.

Important

Lorsque vous déployez un composant, AWS IoT Greengrass installe les dernières versions prises en charge de toutes les dépendances de ce composant. De ce fait, les nouvelles versions de correctif des composants publics AWS fournis peuvent être automatiquement déployées sur vos appareils principaux si vous ajoutez de nouveaux appareils à un groupe d'objets ou si vous mettez à jour le déploiement qui cible ces appareils. Certaines mises à jour automatiques, telles que la mise à jour du noyau, peuvent provoquer le redémarrage inattendu de vos appareils.

Pour éviter les mises à jour involontaires d'un composant en cours d'exécution sur votre appareil, nous vous recommandons d'inclure directement votre version préférée de ce composant lorsque vous [créez un déploiement](#). Pour plus d'informations sur le comportement de mise à jour du logiciel AWS IoT Greengrass Core, consultez [Mettre à jour le logiciel AWS IoT Greengrass principal \(OTA\)](#).

Composant	Détails
Noyau de Greengrass	La version 2.9.1 du noyau Greengrass est disponible .

Composant	Détails
	Corrections de bogues et améliorations • Ajoute un correctif selon lequel Greengrass redémarre si un déploiement supprime un composant du plugin.
Gestionnaire de journaux	La version 2.3.0 du nouveau gestionnaire de journaux est disponible.  Note Nous vous recommandons de passer à Greengrass nucleus 2.9.1 lors de la mise à niveau vers le gestionnaire de journaux 2.3.0. Nouvelles fonctionnalités • Réduit les délais de journalisation en traitant et en téléchargeant directement les fichiers journaux actifs au lieu d'attendre la rotation des nouveaux fichiers. Corrections de bogues et améliorations • Améliore la prise en charge de la rotation des journaux lors de la rotation de fichiers portant un nom unique. • Corrections et améliorations mineures supplémentaires.

Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.9.0 le 15 novembre 2022

Cette version fournit la version 2.9.0 du composant Greengrass Nucleus et des mises AWS à jour des composants fournis.

Date de sortie : 15 novembre 2022

Points forts de la publication

- **Authentification hors ligne** : prend AWS IoT Greengrass désormais en charge l'authentification hors ligne. Vous pouvez configurer votre appareil AWS IoT Greengrass principal de manière à ce

que les appareils clients puissent se connecter à un appareil principal, même si celui-ci n'est pas connecté au cloud. Pour plus d'informations, consultez la section [Authentification hors ligne](#).

- Sous-déploiements : vous pouvez désormais créer des sous-déploiements. Vous pouvez utiliser un sous-déploiement pour résoudre les déploiements infructueux. Chaque sous-déploiement peut tester une configuration différente d'un déploiement infructueux sur un plus petit sous-ensemble d'appareils. Pour plus d'informations, consultez la section [Création de sous-déploiements](#).

Détails de la version

- [Mises à jour des composants publics](#)

Mises à jour des composants publics

Le tableau suivant répertorie les composants AWS fournis qui incluent des fonctionnalités nouvelles et mises à jour.

Important

Lorsque vous déployez un composant, AWS IoT Greengrass installe les dernières versions prises en charge de toutes les dépendances de ce composant. De ce fait, les nouvelles versions de correctif des composants publics AWS fournis peuvent être automatiquement déployées sur vos appareils principaux si vous ajoutez de nouveaux appareils à un groupe d'objets ou si vous mettez à jour le déploiement qui cible ces appareils. Certaines mises à jour automatiques, telles que la mise à jour du noyau, peuvent provoquer le redémarrage inattendu de vos appareils.

Pour éviter les mises à jour involontaires d'un composant en cours d'exécution sur votre appareil, nous vous recommandons d'inclure directement votre version préférée de ce composant lorsque vous [créez un déploiement](#). Pour plus d'informations sur le comportement de mise à jour du logiciel AWS IoT Greengrass Core, consultez [Mettre à jour le logiciel AWS IoT Greengrass principal \(OTA\)](#).

Composant	Détails
Noyau de Greengrass	La version 2.9.0 du noyau Greengrass est disponible .

Composant	Détails
	Nouvelles fonctionnalités • Permet de créer des sous-déploiements qui relancent les déploiements avec un plus petit sous-ensemble d'appareils. Cette fonctionnalité permet de tester et de résoudre les déploiements infructueux de manière plus efficace. Corrections de bogues et améliorations • Améliore la prise en charge des systèmes qui ne disposent pas de <code>useraddgroupadd</code>, et <code>usermod</code>. • Corrections et améliorations mineures supplémentaires.
Authentification de l'appareil client	La version 2.3.0 du composant d'authentification de l'appareil client est disponible. Nouvelles fonctionnalités • Ajoute la prise en charge de l'authentification hors ligne des appareils clients. Grâce à cette fonctionnalité, les appareils clients peuvent continuer à se connecter au périphérique principal lorsque celui-ci n'est pas connecté à Internet. • Ajoute la prise en charge des autorités de certification (CA) fournies par le client. Votre appareil principal utilise une autorité de certification fournie par le client comme certificat racine pour générer des certificats de courtier MQTT.
Courtier MQTT 5 (EMQX)	La version 1.2.0 du Courtier MQTT 5 (EMQX) composant est disponible. Nouvelles fonctionnalités Ajoute la prise en charge des chaînes de certificats.
Courtier MQTT Moquette	La version 2.3.0 du nouveau composant du broker Moquette MQTT est disponible. Nouvelles fonctionnalités Ajoute la prise en charge des chaînes de certificats.

Composant	Détails
Directeur des services secrets	La version 2.1.4 du nouveau gestionnaire de secrets est disponible. Corrections de bogues et améliorations Résout un problème en raison duquel les secrets mis en cache étaient supprimés lors du déploiement du gestionnaire de secrets et du redémarrage du noyau de Greengrass.
Gestionnaire de flux	La version 2.1.2 du nouveau gestionnaire de flux est disponible. Corrections de bogues et améliorations Résout un problème sur les systèmes d'exploitation Windows qui utilisent une langue autre que l'anglais.

Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.8.1 le 13 octobre 2022

Cette version fournit la version 2.8.1 du composant Greengrass nucleus.

Date de sortie : 13 octobre 2022

Note

Si vous utilisez Greengrass nucleus version 2.8.0, nous vous recommandons vivement de passer à Greengrass nucleus version 2.8.1.

Détails de la version

- [Mises à jour des composants publics](#)

Mises à jour des composants publics

Le tableau suivant répertorie les composants AWS fournis qui incluent des fonctionnalités nouvelles et mises à jour.

⚠ Important

Lorsque vous déployez un composant, AWS IoT Greengrass installe les dernières versions prises en charge de toutes les dépendances de ce composant. De ce fait, les nouvelles versions de correctif des composants publics AWS fournis peuvent être automatiquement déployées sur vos appareils principaux si vous ajoutez de nouveaux appareils à un groupe d'objets ou si vous mettez à jour le déploiement qui cible ces appareils. Certaines mises à jour automatiques, telles que la mise à jour du noyau, peuvent provoquer le redémarrage inattendu de vos appareils.

Pour éviter les mises à jour involontaires d'un composant en cours d'exécution sur votre appareil, nous vous recommandons d'inclure directement votre version préférée de ce composant lorsque vous [créez un déploiement](#). Pour plus d'informations sur le comportement de mise à jour du logiciel AWS IoT Greengrass Core, consultez [Mettre à jour le logiciel AWS IoT Greengrass principal \(OTA\)](#).

Composant	Détails
Noyau de Greengrass	La version 2.8.1 du noyau Greengrass est disponible. Corrections de bugs et améliorations • Résout un problème en raison duquel les codes d'erreur de déploiement n'étaient pas générés correctement à cause des erreurs de l'API Greengrass. • Résout un problème selon lequel les mises à jour de l'état du parc envoient des informations inexactes lorsqu'un composant atteint un ERRORRED état au cours d'un déploiement. • Résout un problème en raison duquel les déploiements ne pouvaient pas être achevés lorsque Greengrass avait plus de 50 abonnements existants.

Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.8.0 le 7 octobre 2022

Cette version fournit la version 2.8.0 du composant Greengrass nucleus et la version 1.1.0 du composant MQTT 5 broker (EMQX).

Date de sortie : 7 octobre 2022

Points forts de la publication

- Codes d'erreur de déploiement — Le noyau Greengrass signale désormais une réponse sur [l'état de santé du déploiement](#) qui inclut des codes d'erreur détaillés lorsque le déploiement d'un composant ne peut pas être terminé. Pour de plus amples informations, veuillez consulter [Codes d'erreur de déploiement détaillés](#).
- Statuts d'erreur des composants — Le noyau Greengrass indique désormais [une réponse sur l'état de santé des composants](#) qui inclut des états d'erreur détaillés lorsqu'un composant passe BROKEN à l'état ou. ERRORED Pour de plus amples informations, veuillez consulter [Codes d'état détaillés des composants](#).

Détails de la version

- [Mises à jour des composants publics](#)

Mises à jour des composants publics

Le tableau suivant répertorie les composants AWS fournis qui incluent des fonctionnalités nouvelles et mises à jour.

Important

Lorsque vous déployez un composant, AWS IoT Greengrass installe les dernières versions prises en charge de toutes les dépendances de ce composant. De ce fait, les nouvelles versions de correctif des composants publics AWS fournis peuvent être automatiquement déployées sur vos appareils principaux si vous ajoutez de nouveaux appareils à un groupe d'objets ou si vous mettez à jour le déploiement qui cible ces appareils. Certaines mises à jour automatiques, telles que la mise à jour du noyau, peuvent provoquer le redémarrage inattendu de vos appareils.

Pour éviter les mises à jour involontaires d'un composant en cours d'exécution sur votre appareil, nous vous recommandons d'inclure directement votre version préférée de ce composant lorsque vous [créez un déploiement](#). Pour plus d'informations sur le comportement de mise à jour du logiciel AWS IoT Greengrass Core, consultez [Mettre à jour le logiciel AWS IoT Greengrass principal \(OTA\)](#).

Composant	Détails
Noyau de Greengrass	La version 2.8.0 du noyau Greengrass est disponible. Nouvelles fonctionnalités • Met à jour le noyau Greengrass pour signaler une réponse sur l'état de santé du déploiement qui inclut des codes d'erreur détaillés en cas de problème lors du déploiement de composants sur un périphérique principal. Pour de plus amples informations, veuillez consulter Codes d'erreur de déploiement détaillés. • Met à jour le noyau Greengrass pour signaler une réponse sur l'état de santé d'un composant qui inclut des codes d'erreur détaillés lorsqu'un composant passe à l'état BROKEN ou ERRORERD. Pour de plus amples informations, veuillez consulter Codes d'état détaillés des composants. • Étend les champs des messages d'état afin d'améliorer les informations de disponibilité du cloud pour les appareils. • Améliore la robustesse du service d'état de la flotte. Corrections de bogues et améliorations • Permet à un composant défectueux de se réinstaller lorsque sa configuration change. • Résout un problème selon lequel un redémarrage du noyau pendant le déploiement du bootstrap entraîne l'échec d'un déploiement. • Résout un problème dans Windows où l'installation échoue lorsqu'un chemin racine contient des espaces. • Résout un problème selon lequel un composant arrêté lors d'un déploiement utilise le script d'arrêt de la nouvelle version. • Diverses améliorations en matière d'arrêt.

Composant	Détails
	• Corrections et améliorations mineures supplémentaires.
Courtier MQTT 5 (EMQX)	La version 1.1.0 du Courtier MQTT 5 (EMQX) composant est disponible. Nouvelles fonctionnalités • Prend en charge les configurations EMQX, y compris les options de broker et les plug-ins. Corrections de bogues et améliorations • Met à jour EMQX vers la version 4.4.9.

Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.7.0 le 28 juillet 2022

Cette version fournit la version 2.7.0 du composant Greengrass nucleus, la version 2.1.0 du composant du gestionnaire de flux et la version 2.2.5 du composant du gestionnaire Lambda.

Date de publication : 28 juillet 2022

Points forts de la publication

- Métriques de télémétrie du gestionnaire de flux : le gestionnaire de flux envoie désormais automatiquement des mesures de télémétrie à Amazon EventBridge, afin que vous puissiez créer des applications cloud qui surveillent et analysent le volume de données que vos principaux appareils téléchargent. Pour de plus amples informations, veuillez consulter [Collectez les données de télémétrie relatives à l'état du système à partir des principaux appareils AWS IoT Greengrass](#).
- Autorité de certification personnalisée (CA) — Les certificats clients signés par une autorité de certification personnalisée, auprès de laquelle l'autorité de certification n'est pas enregistrée AWS IoT, sont désormais pris en charge. Pour de plus amples informations, veuillez consulter [Utiliser un certificat d'appareil signé par une autorité de certification privée](#).

Détails de la version

- [Mises à jour des composants publics](#)

Mises à jour des composants publics

Le tableau suivant répertorie les composants AWS fournis qui incluent des fonctionnalités nouvelles et mises à jour.

Important

Lorsque vous déployez un composant, AWS IoT Greengrass installe les dernières versions prises en charge de toutes les dépendances de ce composant. De ce fait, les nouvelles versions de correctif des composants publics AWS fournis peuvent être automatiquement déployées sur vos appareils principaux si vous ajoutez de nouveaux appareils à un groupe d'objets ou si vous mettez à jour le déploiement qui cible ces appareils. Certaines mises à jour automatiques, telles que la mise à jour du noyau, peuvent provoquer le redémarrage inattendu de vos appareils.

Pour éviter les mises à jour involontaires d'un composant en cours d'exécution sur votre appareil, nous vous recommandons d'inclure directement votre version préférée de ce composant lorsque vous [créez un déploiement](#). Pour plus d'informations sur le comportement de mise à jour du logiciel AWS IoT Greengrass Core, consultez [Mettre à jour le logiciel AWS IoT Greengrass principal \(OTA\)](#).

Composant	Détails
Noyau de Greengrass	La version 2.7.0 du noyau Greengrass est disponible. Nouvelles fonctionnalités • Met à jour le noyau Greengrass pour envoyer des mises à jour de statut au AWS IoT Greengrass cloud lorsque l'appareil principal effectue un déploiement local. • Ajoute la prise en charge des certificats clients signés par une autorité de certification (CA) personnalisée, auprès de laquelle l'autorité de certification n'est pas enregistrée AWS IoT. Pour utiliser cette fonctionnalité, vous pouvez définir la nouvelle option <code>greengrassDataPlaneEndpoint</code> de configuration <code>suriotdata</code>. Pour de plus amples informations, veuillez consulter Utiliser un certificat d'appareil signé par une autorité de certification privée.

Composant	Détails
	Corrections de bogues et améliorations • Résout un problème selon lequel le noyau Greengrass annule un déploiement dans certains scénarios lorsque le noyau est arrêté ou redémarré. Le noyau reprend désormais le déploiement après le redémarrage du noyau. • Met à jour le programme d'installation de Greengrass afin de respecter l'--startargument lorsque vous spécifiez de configurer le logiciel en tant que service système. • Met à jour le comportement de SubscribeToComponentUpdates pour définir l'ID de déploiement dans les événements où le noyau a mis à jour un composant. • Corrections et améliorations mineures supplémentaires.
Gestionnaire de flux	La version 2.1.0 du composant du gestionnaire de flux est disponible. Nouvelles fonctionnalités • Met à jour ce composant pour envoyer automatiquement des métriques de télémétrie à Amazon. EventBridge Pour de plus amples informations, veuillez consulter Collectez les données de télémétrie relatives à l'état du système à partir des principaux appareils AWS IoT Greengrass. Cette fonctionnalité nécessite la version 2.7.0 ou ultérieure du composant Greengrass nucleus. • Version mise à jour pour la version 2.7.0 de Greengrass Nucleus.
Gestionnaire Lambda	La version 2.2.5 du composant Lambda Manager est disponible. Nouvelles fonctionnalités • Ajoute la prise en charge des caractères génériques de sujets MQTT dans les sources d'événements où vous vous abonnez à des messages locaux de publication/d'abonnement. Cette fonctionnalité nécessite la version 2.6.0 ou ultérieure du composant Greengrass nucleus. • Version mise à jour pour la version 2.7.0 de Greengrass Nucleus.

Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.6.0 le 27 juin 2022

Cette version fournit la version 2.6.0 du composant Greengrass Nucleus, les AWS nouveaux composants fournis et les mises à jour AWS des composants fournis.

Date de sortie : 27 juin 2022

Points forts de la publication

- Caractères génériques dans les sujets de publication/d'abonnement locaux — Vous pouvez désormais utiliser des caractères génériques MQTT lorsque vous vous abonnez à des sujets de publication/d'abonnement locaux. Pour plus d'informations, consultez [Publier/souscrire des messages locaux](#) et [SubscribeToTopic](#).
- Prise en charge des ombres des appareils clients : vous pouvez désormais interagir avec les ombres des appareils clients dans des composants personnalisés et synchroniser les ombres des appareils clients avec AWS IoT Core. Pour de plus amples informations, veuillez consulter [Interagissez avec les ombres des appareils clients et synchronisez-les](#).
- Support local de MQTT 5 pour les appareils clients — Vous pouvez désormais déployer le broker EMQX MQTT 5 pour utiliser les fonctionnalités MQTT 5 dans la communication entre les appareils clients et un périphérique principal. Pour plus d'informations, consultez [Courtier MQTT 5 \(EMQX\)](#) et [Connect les appareils clients aux appareils principaux](#).
- Variables de recette dans les configurations de composants — Vous pouvez désormais utiliser des variables de recette spécifiques dans les configurations de composants. Vous pouvez utiliser ces variables de recette lorsque vous définissez la configuration par défaut d'un composant dans une recette ou lorsque vous configurez un composant dans un déploiement. Pour plus d'informations, consultez [Variables de recette](#) et [Utiliser des variables de recette dans les mises à jour de fusion](#).
- Caractères génériques dans les politiques d'autorisation IPC — Vous pouvez désormais utiliser le * caractère générique pour correspondre à n'importe quelle combinaison de caractères dans les politiques d'autorisation de communication interprocessus (IPC). Ce caractère générique vous permet d'autoriser l'accès à plusieurs ressources dans le cadre d'une seule politique d'autorisation. Pour de plus amples informations, veuillez consulter [Des caractères génériques dans les politiques d'autorisation](#).
- Opérations IPC qui gèrent les déploiements et les composants locaux — Vous pouvez désormais développer des composants personnalisés qui gèrent les déploiements locaux et afficher les

détails des composants. Pour plus d'informations, voir [IPC : Gérer les déploiements et les composants locaux](#).

- Opérations IPC qui authentifient et autorisent les appareils clients : vous pouvez désormais utiliser ces opérations pour créer un composant de courtier local personnalisé. Pour plus d'informations, voir [IPC : Authentifier et autoriser les appareils clients](#).

Détails de la version

- [Mises à jour des composants publics](#)

Mises à jour des composants publics

Le tableau suivant répertorie les composants AWS fournis qui incluent des fonctionnalités nouvelles et mises à jour.

Important

Lorsque vous déployez un composant, AWS IoT Greengrass installe les dernières versions prises en charge de toutes les dépendances de ce composant. De ce fait, les nouvelles versions de correctif des composants publics AWS fournis peuvent être automatiquement déployées sur vos appareils principaux si vous ajoutez de nouveaux appareils à un groupe d'objets ou si vous mettez à jour le déploiement qui cible ces appareils. Certaines mises à jour automatiques, telles que la mise à jour du noyau, peuvent provoquer le redémarrage inattendu de vos appareils.

Pour éviter les mises à jour involontaires d'un composant en cours d'exécution sur votre appareil, nous vous recommandons d'inclure directement votre version préférée de ce composant lorsque vous [créez un déploiement](#). Pour plus d'informations sur le comportement de mise à jour du logiciel AWS IoT Greengrass Core, consultez [Mettre à jour le logiciel AWS IoT Greengrass principal \(OTA\)](#).

Composant	Détails
Noyau de Greengrass	La version 2.6.0 du noyau Greengrass est disponible .

Composant	Détails
	Nouvelles fonctionnalités • Prend en charge les caractères génériques MQTT lorsque vous vous abonnez à des rubriques de publication/d'abonnement locales. Pour plus d'informations, consultez Publier/souscrire des messages locaux et SubscribeToTopic. • Ajoute la prise en charge des variables de recette dans les configurations de composants, autres que la variable de <i>component_dependency_name</i> :configuration: <i>json_pointer</i> recette. Vous pouvez utiliser ces variables de recettes lorsque vous définissez un composant DefaultConfiguration dans une recette ou lorsque vous configurez un composant dans un déploiement. Pour activer cette fonctionnalité, définissez l'option interpolateComponentConfiguration de configuration sur true. Pour plus d'informations, consultez Variables de recette et Utiliser des variables de recette dans les mises à jour de fusion. • Ajoute une prise en charge complète du * caractère générique dans les politiques d'autorisation de communication interprocessus (IPC). Vous pouvez désormais spécifier le * caractère dans une chaîne de ressources pour qu'il corresponde à n'importe quelle combinaison de caractères. Pour de plus amples informations, veuillez consulter Des caractères génériques dans les politiques d'autorisation. • Ajoute la prise en charge des composants personnalisés pour appeler les opérations IPC utilisées par la CLI Greengrass. Vous pouvez utiliser ces opérations IPC pour gérer les déploiements locaux, afficher les détails des composants et générer un mot de passe que vous pouvez utiliser pour vous connecter à la console de débogage locale. Pour plus d'informations, voir IPC : Gérer les déploiements et les composants locaux. Corrections de bogues et améliorations • Résout un problème selon lequel les composants dépendants ne réagissaient pas lorsque leurs dépendances matérielles redémarraient ou changeaient d'état dans certains scénarios.

Composant	Détails
	• Améliore les messages d'erreur que le périphérique principal envoie au service AWS IoT Greengrass cloud en cas d'échec d'un déploiement. • Résout un problème selon lequel le noyau Greengrass appliquait deux fois le déploiement d'un objet dans certains scénarios lors du redémarrage du noyau. • Corrections et améliorations mineures supplémentaires. Pour plus d'informations, consultez les versions sur GitHub.
Courtier MQTT 5 (EMQX)	La version 1.0.0 du nouveau composant broker EMQX MQTT 5 est disponible. Nouvelles fonctionnalités • Ajoute le support pour le broker EMQX MQTT 5 local. Les appareils clients peuvent se connecter à ce courtier MQTT pour communiquer avec un périphérique principal à l'aide des fonctionnalités de MQTT 5.

Composant	Détails
Shadow Manager	La version 2.2.0 du composant Shadow Manager est disponible. Nouvelles fonctionnalités • Ajoute la prise en charge du service parallèle local par le biais du courtier de publish/subscribe interface. You can now communicate with the local publish/subscribe messages local sur les sujets du shadow MQTT afin d'obtenir, de mettre à jour et de supprimer les ombres sur le périphérique principal. Cette fonctionnalité vous permet de connecter des appareils clients au service parallèle local en utilisant le pont MQTT pour relayer des messages sur des sujets cachés entre les appareils clients et l'interface de publication/d'abonnement locale. Cette fonctionnalité nécessite la version 2.6.0 ou ultérieure du composant Greengrass nucleus. Pour connecter les appareils clients au service parallèle local, vous devez également utiliser la version 2.2.0 ou ultérieure du composant pont MQTT. • Ajoute l'<code>direction</code> option que vous pouvez configurer pour personnaliser la direction afin de synchroniser les ombres entre le service d'ombre local et le AWS Cloud. Vous pouvez configurer cette option pour réduire la bande passante et les connexions au AWS Cloud.
Authentification de l'appareil client	La version 2.2.0 du composant d'authentification de l'appareil client est disponible. Nouvelles fonctionnalités • Ajoute la prise en charge de composants personnalisés pour appeler des opérations de communication interprocessus (IPC) afin d'authentifier et d'autoriser les appareils clients. Vous pouvez utiliser ces opérations dans un composant de courtier MQTT personnalisé, par exemple. Pour plus d'informations, voir IPC : Authentifier et autoriser les appareils clients. • Ajoute les <code>threadPoolSize</code> options <code>maxActiveAuthTokens</code> <code>cloudQueueSize</code> , et que vous pouvez configurer pour ajuster les performances de ce composant.

Composant	Détails
Pont MQTT	La version 2.2.0 du composant pont MQTT est disponible. Nouvelles fonctionnalités • Ajoute la prise en charge des caractères génériques des rubriques MQTT (#et+) lorsque vous spécifiez la publication ou l'abonnement local comme courtier de messages source. Cette fonctionnalité nécessite la version 2.6.0 ou ultérieure du composant Greengrass nucleus. • Ajoute l'<code>targetTopicPrefix</code> option, que vous pouvez spécifier pour configurer le pont MQTT afin d'ajouter un préfixe au sujet cible lorsqu'il relaie un message.
Greengrass CLI	La version 2.6.0 de Greengrass CLI est disponible. Nouvelles fonctionnalités • Ajoute la prise en charge de composants personnalisés pour appeler les opérations de communication interprocessus (IPC) utilisées par la Greengrass CLI. Vous pouvez utiliser ces opérations IPC pour gérer les déploiements locaux, afficher les détails des composants et générer un mot de passe que vous pouvez utiliser pour vous connecter à la console de débogage locale. Pour plus d'informations, voir IPC : Gérer les déploiements et les composants locaux. Corrections de bogues et améliorations • Corrections et améliorations mineures supplémentaires.

Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.5.6 le 31 mai 2022

Cette version fournit la version 2.5.6 du composant Greengrass nucleus et la version 2.2.4 du composant du gestionnaire de journaux.

Date de sortie : 31 mai 2022

Détails de la version

- [Mises à jour des composants publics](#)

Mises à jour des composants publics

Le tableau suivant répertorie les composants AWS fournis qui incluent des fonctionnalités nouvelles et mises à jour.

Important

Lorsque vous déployez un composant, AWS IoT Greengrass installe les dernières versions prises en charge de toutes les dépendances de ce composant. De ce fait, les nouvelles versions de correctif des composants publics AWS fournis peuvent être automatiquement déployées sur vos appareils principaux si vous ajoutez de nouveaux appareils à un groupe d'objets ou si vous mettez à jour le déploiement qui cible ces appareils. Certaines mises à jour automatiques, telles que la mise à jour du noyau, peuvent provoquer le redémarrage inattendu de vos appareils.

Pour éviter les mises à jour involontaires d'un composant en cours d'exécution sur votre appareil, nous vous recommandons d'inclure directement votre version préférée de ce composant lorsque vous [créez un déploiement](#). Pour plus d'informations sur le comportement de mise à jour du logiciel AWS IoT Greengrass Core, consultez [Mettre à jour le logiciel AWS IoT Greengrass principal \(OTA\)](#).

Composant	Détails
Noyau de Greengrass	La version 2.5.6 du noyau Greengrass est disponible. Nouvelles fonctionnalités • Ajoute la prise en charge des modules de sécurité matériels qui utilisent des clés ECC. Vous pouvez utiliser un module de sécurité matérielle (HSM) pour stocker en toute sécurité la clé privée et le certificat de l'appareil. Pour de plus amples informations, veuillez consulter Intégration de sécurité matérielle.

Composant	Détails
	Corrections de bogues et améliorations • Résout un problème selon lequel le déploiement ne se termine jamais lorsque vous déployez un composant avec un script d'installation défectueux dans certains scénarios. • Améliore les performances au démarrage. • Corrections et améliorations mineures supplémentaires.
Gestionnaire de journaux	La version 2.2.4 du composant du gestionnaire de journaux est disponible. Corrections de bogues et améliorations • Améliore la stabilité lors de la gestion de configurations non valides. • Corrections et améliorations mineures supplémentaires.

Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.5.5 le 6 avril 2022

Cette version fournit la version 2.5.5 du composant Greengrass nucleus.

Date de sortie : 6 avril 2022

Détails de la version

- [Mises à jour des composants publics](#)

Mises à jour des composants publics

Le tableau suivant répertorie les composants AWS fournis qui incluent des fonctionnalités nouvelles et mises à jour.

Important

Lorsque vous déployez un composant, AWS IoT Greengrass installe les dernières versions prises en charge de toutes les dépendances de ce composant. De ce fait, les nouvelles versions de correctif des composants publics AWS fournis peuvent être automatiquement déployées sur vos appareils principaux si vous ajoutez de nouveaux appareils à un groupe

d'objets ou si vous mettez à jour le déploiement qui cible ces appareils. Certaines mises à jour automatiques, telles que la mise à jour du noyau, peuvent provoquer le redémarrage inattendu de vos appareils.

Pour éviter les mises à jour involontaires d'un composant en cours d'exécution sur votre appareil, nous vous recommandons d'inclure directement votre version préférée de ce composant lorsque vous [créez un déploiement](#). Pour plus d'informations sur le comportement de mise à jour du logiciel AWS IoT Greengrass Core, consultez [Mettre à jour le logiciel AWS IoT Greengrass principal \(OTA\)](#).

Composant	Détails
Noyau de Greengrass	La version 2.5.5 du noyau Greengrass est disponible. Nouvelles fonctionnalités • Ajoute la variable d'GG_ROOT_CA_PATH environnement pour les composants, afin que vous puissiez accéder au certificat de l'autorité de certification (CA) racine dans les composants personnalisés. Corrections de bogues et améliorations • Ajoute la prise en charge des appareils Windows qui utilisent une langue d'affichage autre que l'anglais. • Met à jour la façon dont le noyau Greengrass analyse les arguments booléens du programme d'installation, afin que vous puissiez spécifier un argument booléen sans valeur booléenne pour spécifier une valeur. true Par exemple, vous pouvez désormais spécifier le provisionnement automatique des --provision true ressources --provision au lieu d'installer. • Résout un problème selon lequel le périphérique principal ne signalait pas son état au service AWS IoT Greengrass cloud après le provisionnement dans certains scénarios. • Corrections et améliorations mineures supplémentaires.

Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.5.4 le 23 mars 2022

Cette version fournit la version 2.5.4 du composant Greengrass nucleus et la version 2.0.10 du composant du lanceur Lambda.

Date de sortie : 23 mars 2022

Détails de la version

- [Mises à jour des composants publics](#)

Mises à jour des composants publics

Le tableau suivant répertorie les composants AWS fournis qui incluent des fonctionnalités nouvelles et mises à jour.

Important

Lorsque vous déployez un composant, AWS IoT Greengrass installe les dernières versions prises en charge de toutes les dépendances de ce composant. De ce fait, les nouvelles versions de correctif des composants publics AWS fournis peuvent être automatiquement déployées sur vos appareils principaux si vous ajoutez de nouveaux appareils à un groupe d'objets ou si vous mettez à jour le déploiement qui cible ces appareils. Certaines mises à jour automatiques, telles que la mise à jour du noyau, peuvent provoquer le redémarrage inattendu de vos appareils.

Pour éviter les mises à jour involontaires d'un composant en cours d'exécution sur votre appareil, nous vous recommandons d'inclure directement votre version préférée de ce composant lorsque vous [créez un déploiement](#). Pour plus d'informations sur le comportement de mise à jour du logiciel AWS IoT Greengrass Core, consultez [Mettre à jour le logiciel AWS IoT Greengrass principal \(OTA\)](#).

Composant	Détails
Noyau de Greengrass	La version 2.5.4 du noyau Greengrass est disponible .

Composant	Détails
	Corrections de bogues et améliorations • Correction et amélioration de bogues généraux
Lanceur Lambda	La version 2.0.10 du composant du lanceur Lambda est disponible. Corrections de bogues et améliorations • Correction et amélioration de bogues généraux

Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.5.3 le 6 janvier 2022

Cette version fournit la version 2.5.3 du composant Greengrass nucleus et le nouveau composant fournisseur PKCS #11.

Date de sortie : 6 janvier 2022

Points forts de la publication

- Intégration de la sécurité matérielle : vous pouvez désormais configurer le logiciel AWS IoT Greengrass principal pour utiliser une clé privée et un certificat que vous stockez en toute sécurité dans un module de sécurité matériel (HSM). Pour de plus amples informations, veuillez consulter [Intégration de sécurité matérielle](#).

Détails de la version

- [Mises à jour des composants publics](#)

Mises à jour des composants publics

Le tableau suivant répertorie les composants AWS fournis qui incluent des fonctionnalités nouvelles et mises à jour.

Important

Lorsque vous déployez un composant, AWS IoT Greengrass installe les dernières versions prises en charge de toutes les dépendances de ce composant. De ce fait, les nouvelles

versions de correctif des composants publics AWS fournis peuvent être automatiquement déployées sur vos appareils principaux si vous ajoutez de nouveaux appareils à un groupe d'objets ou si vous mettez à jour le déploiement qui cible ces appareils. Certaines mises à jour automatiques, telles que la mise à jour du noyau, peuvent provoquer le redémarrage inattendu de vos appareils.

Pour éviter les mises à jour involontaires d'un composant en cours d'exécution sur votre appareil, nous vous recommandons d'inclure directement votre version préférée de ce composant lorsque vous [créez un déploiement](#). Pour plus d'informations sur le comportement de mise à jour du logiciel AWS IoT Greengrass Core, consultez [Mettre à jour le logiciel AWS IoT Greengrass principal \(OTA\)](#).

Composant	Détails
Noyau de Greengrass	La version 2.5.3 du noyau Greengrass est disponible. Nouvelles fonctionnalités • Ajoute la prise en charge de l'intégration de la sécurité matérielle. Vous pouvez utiliser un module de sécurité matérielle (HSM) pour stocker en toute sécurité la clé privée et le certificat de l'appareil. Pour de plus amples informations, veuillez consulter Intégration de sécurité matérielle. Corrections de bogues et améliorations • Résout un problème lié aux exceptions d'exécution lorsque le noyau établit des connexions MQTT avec AWS IoT Core.
Fournisseur PKCS #11	La version 2.0.0 du composant fournisseur PKCS #11 est disponible. Nouvelles fonctionnalités • Ajoute la prise en charge de l'intégration de la sécurité matérielle. Vous pouvez utiliser un module de sécurité matérielle (HSM) pour stocker en toute sécurité la clé privée et le certificat de l'appareil. Pour de plus amples informations, veuillez consulter Intégration de sécurité matérielle.

Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.5.2 le 3 décembre 2021

Cette version fournit la version 2.5.2 du composant Greengrass nucleus.

Date de sortie : 3 décembre 2021

Détails de la version

- [Mises à jour des composants publics](#)

Mises à jour des composants publics

Le tableau suivant répertorie les composants AWS fournis qui incluent des fonctionnalités nouvelles et mises à jour.

Important

Lorsque vous déployez un composant, AWS IoT Greengrass installe les dernières versions prises en charge de toutes les dépendances de ce composant. De ce fait, les nouvelles versions de correctif des composants publics AWS fournis peuvent être automatiquement déployées sur vos appareils principaux si vous ajoutez de nouveaux appareils à un groupe d'objets ou si vous mettez à jour le déploiement qui cible ces appareils. Certaines mises à jour automatiques, telles que la mise à jour du noyau, peuvent provoquer le redémarrage inattendu de vos appareils.

Pour éviter les mises à jour involontaires d'un composant en cours d'exécution sur votre appareil, nous vous recommandons d'inclure directement votre version préférée de ce composant lorsque vous [créez un déploiement](#). Pour plus d'informations sur le comportement de mise à jour du logiciel AWS IoT Greengrass Core, consultez [Mettre à jour le logiciel AWS IoT Greengrass principal \(OTA\)](#).

Composant	Détails
Noyau de Greengrass	La version 2.5.2 du noyau Greengrass est disponible .

Composant	Détails
	Corrections de bogues et améliorations • Résout un problème selon lequel, après les mises à jour du noyau Greengrass, le service Windows ne redémarre pas une fois que vous l'avez arrêté ou redémarré l'appareil.
AWS IoT Device Defender	La version 3.0.1 du AWS IoT Device Defender composant est disponible. Cette version du AWS IoT Device Defender composant attend des paramètres de configuration différents de ceux de la version 2.x. Si vous utilisez une configuration autre que celle par défaut pour la version 2.x et que vous souhaitez passer de la version 2.x à la version 3.x, vous devez mettre à jour la configuration du composant. Pour plus d'informations, consultez la section Configuration des AWS IoT Device Defender composants. Nouvelles fonctionnalités • Ajoute la prise en charge des appareils principaux qui exécutent Windows. • Change le type de composant Lambda en composant générique. Ce composant ne dépend désormais plus de l'ancien composant routeur d'abonnement pour créer des abonnements. • Ajoute le nouveau paramètre <code>UseInstaller</code> de configuration qui permet de désactiver éventuellement le script d'installation qui installe les dépendances des composants.

Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.5.1 le 23 novembre 2021

Cette version fournit la version 2.5.1 du composant Greengrass nucleus.

Date de parution : 23 novembre 2021

Détails de la version

- [Mises à jour des composants publics](#)

Mises à jour des composants publics

Le tableau suivant répertorie les composants AWS fournis qui incluent des fonctionnalités nouvelles et mises à jour.

Important

Lorsque vous déployez un composant, AWS IoT Greengrass installe les dernières versions prises en charge de toutes les dépendances de ce composant. De ce fait, les nouvelles versions de correctif des composants publics AWS fournis peuvent être automatiquement déployées sur vos appareils principaux si vous ajoutez de nouveaux appareils à un groupe d'objets ou si vous mettez à jour le déploiement qui cible ces appareils. Certaines mises à jour automatiques, telles que la mise à jour du noyau, peuvent provoquer le redémarrage inattendu de vos appareils.

Pour éviter les mises à jour involontaires d'un composant en cours d'exécution sur votre appareil, nous vous recommandons d'inclure directement votre version préférée de ce composant lorsque vous [créez un déploiement](#). Pour plus d'informations sur le comportement de mise à jour du logiciel AWS IoT Greengrass Core, consultez [Mettre à jour le logiciel AWS IoT Greengrass principal \(OTA\)](#).

Composant	Détails
Noyau de Greengrass	La version 2.5.1 du noyau Greengrass est disponible. Corrections de bugs et améliorations • Ajoute la prise en charge des versions 32 bits de l'environnement d'exécution Java (JRE) sous Windows. • Modifie le comportement de suppression des groupes d'objets pour les principaux appareils dont la AWS IoT politique n'accorde pas l'<code>greengrass:ListThingGroupsForCoreDevice</code> autorisation. Dans cette version, le déploiement se poursuit, enregistre un avertissement et ne supprime aucun composant lorsque vous supprimez le périphérique principal d'un groupe d'objets. Pour de plus amples informations, veuillez consulter Déployer AWS IoT Greengrass des composants sur des appareils.

Composant	Détails
	• Résout un problème lié aux variables d'environnement système que le noyau de Greengrass met à la disposition des processus des composants de Greengrass. Vous pouvez désormais redémarrer un composant pour qu'il utilise les dernières variables d'environnement système.

Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.5.0 le 12 novembre 2021

Cette version fournit la version 2.5.0 du composant Greengrass Nucleus, les AWS nouveaux composants fournis et les mises à jour AWS des composants fournis.

Date de sortie : 12 novembre 2021

Points forts de la publication

- Prise en charge des appareils Windows : vous pouvez désormais exécuter le logiciel AWS IoT Greengrass Core sur des appareils exécutant des systèmes d'exploitation Windows. Pour de plus amples informations, veuillez consulter [Compatibilité des fonctionnalités de Greengrass](#).
- Nouveau comportement de suppression de groupes d'objets : vous pouvez désormais supprimer un appareil principal d'un groupe d'objets pour supprimer les composants de ce groupe d'objets lors du prochain déploiement sur cet appareil.

Important

À la suite de cette modification, la AWS IoT politique d'un appareil principal doit être `greengrass:ListThingGroupsForCoreDevice` autorisée. Si vous avez utilisé le [programme d'installation du logiciel AWS IoT Greengrass Core pour provisionner des ressources](#), la AWS IoT politique par défaut `l'autorisegreengrass:*`, y compris cette autorisation. Pour de plus amples informations, veuillez consulter [Authentification et autorisation de l'appareil pour AWS IoT Greengrass](#).

- Support de sécurité matérielle : vous pouvez désormais configurer le logiciel AWS IoT Greengrass principal pour utiliser un module de sécurité matérielle (HSM), afin de pouvoir stocker en toute

sécurité la clé privée et le certificat de l'appareil. Pour de plus amples informations, veuillez consulter [Intégration de sécurité matérielle](#).

- Support du proxy HTTPS : vous pouvez désormais configurer le logiciel AWS IoT Greengrass Core pour qu'il se connecte via des proxys HTTPS. Pour de plus amples informations, veuillez consulter [Connexion au port 443 ou via un proxy réseau](#).

Détails de la version

- [Mises à jour du support technique](#)
- [Mises à jour des composants publics](#)

Mises à jour du support technique

Plateforme	Détails
Windows	AWS IoT Greengrass prend désormais en charge l'exécution du logiciel AWS IoT Greengrass Core sur les versions suivantes de Windows : • Windows 10 • Windows Server 2019 Pour de plus amples informations, veuillez consulter Compatibilité des fonctionnalités de Greengrass.

Mises à jour des composants publics

Le tableau suivant répertorie les composants AWS fournis qui incluent des fonctionnalités nouvelles et mises à jour.

Important

Lorsque vous déployez un composant, AWS IoT Greengrass installe les dernières versions prises en charge de toutes les dépendances de ce composant. De ce fait, les nouvelles versions de correctif des composants publics AWS fournis peuvent être automatiquement déployées sur vos appareils principaux si vous ajoutez de nouveaux appareils à un groupe d'objets ou si vous mettez à jour le déploiement qui cible ces appareils. Certaines mises à

jour automatiques, telles que la mise à jour du noyau, peuvent provoquer le redémarrage inattendu de vos appareils.

Pour éviter les mises à jour involontaires d'un composant en cours d'exécution sur votre appareil, nous vous recommandons d'inclure directement votre version préférée de ce composant lorsque vous [créez un déploiement](#). Pour plus d'informations sur le comportement de mise à jour du logiciel AWS IoT Greengrass Core, consultez [Mettre à jour le logiciel AWS IoT Greengrass principal \(OTA\)](#).

Composant	Détails
Noyau de Greengrass	La version 2.5.0 du noyau Greengrass est disponible. Nouvelles fonctionnalités • Ajoute la prise en charge des appareils principaux qui exécutent Windows. • Modifiez le comportement de la suppression de groupes d'objets. Avec cette version, vous pouvez supprimer un périphérique principal d'un groupe d'objets pour désinstaller les composants de ce groupe d'objets lors du prochain déploiement. À la suite de cette modification, la AWS IoT politique d'un appareil principal doit être <code>greengrass:ListThingGroupsForCoreDevice</code> autorisée. Si vous avez utilisé le programme d'installation du logiciel AWS IoT Greengrass Core pour provisionner des ressources, la AWS IoT politique par défaut <code>l'authorisegreengrass:*</code>, y compris cette autorisation. Pour de plus amples informations, veuillez consulter Authentification et autorisation de l'appareil pour AWS IoT Greengrass. • Ajoute la prise en charge des configurations de proxy HTTPS. Pour de plus amples informations, veuillez consulter Connexion au port 443 ou via un proxy réseau. • Ajoute le nouveau paramètre <code>windowsUser</code> de configuration. Vous pouvez utiliser ce paramètre pour spécifier l'utilisateur par défaut à utiliser pour exécuter les composants sur un périphérique principal Windows. Pour de plus amples informations, veuillez consulter Configurer l'utilisateur qui exécute les composants.

Composant	Détails
	• Ajoute les nouvelles options <code>httpClient</code> de configuration que vous pouvez utiliser pour personnaliser les délais d'expiration des requêtes HTTP afin d'améliorer les performances sur les réseaux lents. Pour plus d'informations, consultez le paramètre de configuration HttpClient. Corrections de bogues et améliorations • Corrige l'option de cycle de vie du bootstrap permettant de redémarrer le périphérique principal à partir d'un composant. • Ajoute la prise en charge des traits d'union dans les variables de recette. • Corrige l'autorisation IPC pour les composants de la fonction Lambda à la demande. • Améliore les messages de journal et fait passer les journaux non critiques du DEBUG niveau INFO au niveau supérieur, afin que les journaux soient plus utiles. • Supprime l'<code>iot:DescribeCertificate</code> autorisation du rôle d'échange de jetons par défaut créé par le noyau Greengrass lorsque vous installez le logiciel AWS IoT Greengrass Core avec provisionnement automatique. Cette autorisation n'est pas utilisée par le noyau Greengrass. • Résout un problème selon lequel le script de provisionnement automatique ne nécessite pas d'<code>iam:GetPolicy</code> autorisation s'il <code>iam:CreatePolicy</code> est disponible pour la même politique. • Corrections et améliorations mineures supplémentaires.

Composant	Détails
Greengrass CLI	La version 2.5.0 de la CLI Greengrass est disponible. Nouvelles fonctionnalités • Ajoute la prise en charge des appareils principaux qui exécutent Windows. • Ajoute le nouveau paramètre <code>AuthorizedWindowsGroups</code> de configuration que vous pouvez spécifier pour autoriser les groupes de systèmes à utiliser la CLI Greengrass sur les appareils Windows. • Ajoute le <code>windowsUser</code> paramètre pour les déploiements locaux. Vous pouvez utiliser ce paramètre pour spécifier l'utilisateur à utiliser pour exécuter les composants sur un périphérique principal Windows.

Composant	Détails
CloudWatch métriques	La version 3.0.0 du composant CloudWatchmetrics est disponible. Cette version du composant CloudWatch metrics attend des paramètres de configuration différents de ceux de la version 2.x. Si vous utilisez une configuration autre que celle par défaut pour la version 2.x et que vous souhaitez passer de la version 2.x à la version 3.x, vous devez mettre à jour la configuration du composant. Pour plus d'informations, consultez la section Configuration CloudWatch des composants métriques. Nouvelles fonctionnalités • Ajoute la prise en charge des appareils principaux qui exécutent Windows. • Change le type de composant Lambda en composant générique. Ce composant ne dépend désormais plus de l'ancien composant routeur d'abonnement pour créer des abonnements. • Ajoute un nouveau paramètre de <code>InputTopic</code> de configuration pour spécifier le sujet auquel le composant s'abonne pour recevoir des messages. • Ajoute un nouveau paramètre de <code>OutputTopic</code> de configuration pour spécifier le sujet dans lequel le composant publie les réponses d'état. • Ajoute un nouveau paramètre <code>PubSubToIoTCore</code> de configuration pour spécifier s'il faut publier des sujets AWS IoT Core MQTT et s'y abonner. • Ajoute le nouveau paramètre <code>UseInstaller</code> de configuration qui permet de désactiver éventuellement le script d'installation qui installe les dépendances des composants. Corrections de bogues et améliorations Ajoute la prise en charge des horodatages dupliqués dans les données d'entrée.

Composant	Détails
Gestionnaire Lambda	La version 2.2.0 du composant Lambda Manager est disponible. Corrections de bogues et améliorations • Résout un problème en raison duquel les fonctions Lambda ne pouvaient pas écrire de journaux après un redémarrage. • Résout un problème selon lequel l'ancien routeur d'abonnement envoie des messages dupliqués lorsque le sujet contient des caractères génériques. • Résout un problème en raison duquel les fonctions Lambda non épinglées ne pouvaient pas utiliser la bibliothèque de communication interprocessus (IPC) Greengrass dans le Kit SDK des appareils AWS IoT

Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.4.0 le 3 août 2021

Cette version fournit la version 2.4.0 du composant Greengrass Nucleus, les AWS nouveaux composants fournis et les mises à jour AWS des composants fournis.

Date de sortie : 3 août 2021

Points forts de la publication

- Limites de ressources du système — Le composant Greengrass nucleus prend désormais en charge les limites de ressources du système. Vous pouvez configurer l'utilisation maximale du processeur et de la RAM que les processus de chaque composant peuvent utiliser sur le périphérique principal. Pour de plus amples informations, veuillez consulter [Configuration des limites de ressources système pour les composants](#).
- Suspendre/reprendre les composants : le noyau Greengrass prend désormais en charge la suspension et la reprise des composants. Vous pouvez utiliser la bibliothèque de communication interprocessus (IPC) pour développer des composants personnalisés qui interrompent et reprennent les processus des autres composants. Pour plus d'informations, consultez [PauseComponent](#) et [ResumeComponent](#).

- Installation avec provisionnement de AWS IoT flotte : utilisez le nouveau plug-in de provisionnement de AWS IoT flotte pour installer le logiciel AWS IoT Greengrass Core sur les appareils qui se connectent pour provisionner les ressources AWS IoT requises. Les appareils utilisent un certificat de réclamation pour le provisionnement. Vous pouvez intégrer le certificat de réclamation sur les appareils pendant la fabrication, afin que chaque appareil puisse être approvisionné dès sa mise en ligne. Pour de plus amples informations, veuillez consulter [Installation AWS IoT Greengrass du logiciel de base avec provisionnement du AWS IoT parc](#).
- Installation avec provisionnement personnalisé : développez un plug-in de provisionnement personnalisé pour provisionner les AWS ressources requises lorsque vous installez le logiciel AWS IoT Greengrass Core sur des appareils. Vous pouvez créer une application Java qui s'exécute pendant l'installation pour configurer les appareils principaux de Greengrass en fonction de votre cas d'utilisation personnalisé. Pour de plus amples informations, veuillez consulter [Installez le logiciel AWS IoT Greengrass Core avec un provisionnement personnalisé des ressources](#).

Détails de la version

- [Mises à jour des composants publics](#)

Mises à jour des composants publics

Le tableau suivant répertorie les composants AWS fournis qui incluent des fonctionnalités nouvelles et mises à jour.

Important

Lorsque vous déployez un composant, AWS IoT Greengrass installe les dernières versions prises en charge de toutes les dépendances de ce composant. De ce fait, les nouvelles versions de correctif des composants publics AWS fournis peuvent être automatiquement déployées sur vos appareils principaux si vous ajoutez de nouveaux appareils à un groupe d'objets ou si vous mettez à jour le déploiement qui cible ces appareils. Certaines mises à jour automatiques, telles que la mise à jour du noyau, peuvent provoquer le redémarrage inattendu de vos appareils.

Pour éviter les mises à jour involontaires d'un composant en cours d'exécution sur votre appareil, nous vous recommandons d'inclure directement votre version préférée de ce composant lorsque vous [créez un déploiement](#). Pour plus d'informations sur le comportement

de mise à jour du logiciel AWS IoT Greengrass Core, consultez [Mettre à jour le logiciel AWS IoT Greengrass principal \(OTA\)](#).

Composant	Détails
Noyau de Greengrass	La version 2.4.0 du noyau Greengrass est disponible. Nouvelles fonctionnalités • Ajoute la prise en charge des limites de ressources du système. Vous pouvez configurer l'utilisation maximale du processeur et de la RAM que les processus de chaque composant peuvent utiliser sur le périphérique principal. Pour de plus amples informations, veuillez consulter Configuration des limites de ressources système pour les composants. • Ajoute des opérations IPC pour suspendre et reprendre les composants. Pour plus d'informations, consultez PauseComposant et ResumeComposant. • Ajoute la prise en charge des plugins de provisionnement. Vous pouvez spécifier un fichier JAR à exécuter pendant l'installation afin de fournir les AWS ressources nécessaires à un périphérique principal Greengrass. Le noyau Greengrass inclut une interface que vous pouvez implémenter pour développer des plugins de provisionnement personnalisés. Pour de plus amples informations, veuillez consulter Installez le logiciel AWS IoT Greengrass Core avec un provisionnement personnalisé des ressources. • Ajoute l'<code>thing-name-policy</code> argument facultatif au programme d'installation du logiciel AWS IoT Greengrass Core. Vous pouvez utiliser cette option pour spécifier une AWS IoT politique existante ou personnalisée lorsque vous installez le logiciel AWS IoT Greengrass Core avec provisionnement automatique des ressources. Corrections de bogues et améliorations • Met à jour la configuration de journalisation au démarrage. Cela résout un problème en raison duquel la configuration de journalisation n'était pas appliquée au démarrage.

Composant	Détails
	• Met à jour le lien symbolique du chargeur Nucleus pour qu'il pointe vers le magasin de composants dans le dossier racine de Greengrass lors de l'installation. Cette mise à jour vous permet de supprimer le fichier JAR et les autres artefacts Nucleus que vous téléchargez lorsque vous installez le logiciel AWS IoT Greengrass Core. • Corrections et améliorations mineures supplémentaires. Pour plus d'informations, consultez les publications sur GitHub.
Greengrass CLI	La version 2.4.0 de la Greengrass CLI est disponible. Nouvelles fonctionnalités • Ajoute la prise en charge des limites de ressources du système. Lorsque vous créez un déploiement local, vous pouvez configurer la quantité maximale d'utilisation du processeur et de la RAM que les processus de chaque composant peuvent utiliser sur le périphérique principal. Pour plus d'informations, consultez la section Configuration des limites de ressources système pour les composants et la commande Deployment Create.
AWS IoT approvisi onnement de la flotte par sinistre	Le plugin AWS IoT de provisionnement de flotte par réclamation est désormais disponible. Pour de plus amples informations, veuillez consulter Installation AWS IoT Greengrass du logiciel de base avec provisionnement du AWS IoT parc. Nouvelles fonctionnalités • Ajoute la prise en charge de l'installation du logiciel de AWS IoT Greengrass base avec le provisionnement du AWS IoT parc. Pendant l'installation, les appareils se connectent AWS IoT pour fournir les AWS ressources requises et télécharger les certificats des appareils à utiliser pour les opérations régulières.

Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.3.0 le 29 juin 2021

Cette version fournit la version 2.3.0 du composant Greengrass nucleus.

Date de sortie : 29 juin 2021

Points forts de la publication

- Prise en charge de configurations étendues — Le composant Greengrass nucleus prend désormais en charge les documents de déploiement d'une taille maximale de 10 Mo. Vous pouvez désormais déployer des mises à jour de configuration plus importantes sur les composants de Greengrass.

Note

Pour utiliser cette fonctionnalité, la AWS IoT politique d'un appareil principal doit autoriser `greengrass:GetDeploymentConfiguration`. Si vous avez utilisé le [programme d'installation du logiciel AWS IoT Greengrass Core pour provisionner des ressources](#), la AWS IoT politique de votre appareil principal le permet `greengrass:*`, y compris cette autorisation. Pour de plus amples informations, veuillez consulter [Authentification et autorisation de l'appareil pour AWS IoT Greengrass](#).

Détails de la version

- [Mises à jour des composants publics](#)

Mises à jour des composants publics

Le tableau suivant répertorie les composants AWS fournis qui incluent des fonctionnalités nouvelles et mises à jour.

Important

Lorsque vous déployez un composant, AWS IoT Greengrass installe les dernières versions prises en charge de toutes les dépendances de ce composant. De ce fait, les nouvelles versions de correctif des composants publics AWS fournis peuvent être automatiquement

déployées sur vos appareils principaux si vous ajoutez de nouveaux appareils à un groupe d'objets ou si vous mettez à jour le déploiement qui cible ces appareils. Certaines mises à jour automatiques, telles que la mise à jour du noyau, peuvent provoquer le redémarrage inattendu de vos appareils.

Pour éviter les mises à jour involontaires d'un composant en cours d'exécution sur votre appareil, nous vous recommandons d'inclure directement votre version préférée de ce composant lorsque vous [créez un déploiement](#). Pour plus d'informations sur le comportement de mise à jour du logiciel AWS IoT Greengrass Core, consultez [Mettre à jour le logiciel AWS IoT Greengrass principal \(OTA\)](#).

Composant	Détails
Noyau de Greengrass	La version 2.3.0 du noyau Greengrass est disponible. Nouvelles fonctionnalités • Prend en charge les documents de configuration de déploiement jusqu'à 10 Mo, au lieu de 7 Ko (pour les déploiements ciblant des objets) ou 31 Ko (pour les déploiements ciblant des groupes d'objets). Pour utiliser cette fonctionnalité, la AWS IoT politique d'un appareil principal doit autoriser l'<code>greengrass:GetDeploymentConfiguration</code> autorisation. Si vous avez utilisé le programme d'installation du logiciel AWS IoT Greengrass Core pour provisionner des ressources, la AWS IoT politique de votre appareil principal le permet <code>greengrass:*</code>, y compris cette autorisation. Pour de plus amples informations, veuillez consulter Authentification et autorisation de l'appareil pour AWS IoT Greengrass. • Ajoute la variable de <code>iot:thingName</code> recette. Vous pouvez utiliser cette variable de recette pour obtenir le nom de l'appareil AWS IoT principal d'une recette. Pour de plus amples informations, veuillez consulter Variables de recette. Corrections de bogues et améliorations • Corrections et améliorations mineures supplémentaires. Pour plus d'informations, consultez les versions sur GitHub.

Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.2.0 le 18 juin 2021

Cette version fournit la version 2.2.0 du composant Greengrass Nucleus, les AWS nouveaux composants fournis et les mises à jour AWS des composants fournis.

Date de sortie : 18 juin 2021

Points forts de la publication

- Prise en charge des appareils clients : les nouveaux composants AWS fournis vous permettent de connecter des appareils clients à vos appareils principaux à l'aide de la découverte du cloud. Vous pouvez synchroniser les appareils clients avec les appareils clients AWS IoT Core et interagir avec eux dans les composants Greengrass. Pour de plus amples informations, veuillez consulter [Interagissez avec les appareils IoT locaux](#).
- Service parallèle local : le nouveau composant du gestionnaire parallèle active le service parallèle local sur vos appareils principaux. Vous pouvez utiliser ce service parallèle pour interagir avec les ombres locales lorsque vous êtes hors ligne à l'aide des bibliothèques de communication interprocessus (IPC) Greengrass du Kit SDK des appareils AWS IoT. Vous pouvez également utiliser le composant Shadow Manager pour synchroniser les états des ombres locaux avec AWS IoT Core. Pour de plus amples informations, veuillez consulter [Interagissez avec les ombres de l'appareil](#).

Détails de la version

- [Mises à jour des composants publics](#)

Mises à jour des composants publics

Le tableau suivant répertorie les composants AWS fournis qui incluent des fonctionnalités nouvelles et mises à jour.

Important

Lorsque vous déployez un composant, AWS IoT Greengrass installe les dernières versions prises en charge de toutes les dépendances de ce composant. De ce fait, les nouvelles versions de correctif des composants publics AWS fournis peuvent être automatiquement déployées sur vos appareils principaux si vous ajoutez de nouveaux appareils à un groupe

d'objets ou si vous mettez à jour le déploiement qui cible ces appareils. Certaines mises à jour automatiques, telles que la mise à jour du noyau, peuvent provoquer le redémarrage inattendu de vos appareils.

Pour éviter les mises à jour involontaires d'un composant en cours d'exécution sur votre appareil, nous vous recommandons d'inclure directement votre version préférée de ce composant lorsque vous [créez un déploiement](#). Pour plus d'informations sur le comportement de mise à jour du logiciel AWS IoT Greengrass Core, consultez [Mettre à jour le logiciel AWS IoT Greengrass principal \(OTA\)](#).

Composant	Détails
Noyau de Greengrass	La version 2.2.0 du noyau Greengrass est disponible. Nouvelles fonctionnalités • Ajoute des opérations IPC pour la gestion des ombres locales. Corrections de bogues et améliorations • Réduit la taille du fichier JAR. • Réduit l'utilisation de la mémoire. • Résout les problèmes liés au fait que la configuration du journal n'était pas mise à jour dans certains cas. • Corrections et améliorations mineures supplémentaires. Pour plus d'informations, consultez les versions sur GitHub.
Shadow Manager	La version 2.0.0 du nouveau composant Shadow Manager est disponible. Nouvelles fonctionnalités • Ajoute la prise en charge des ombres classiques et nommées. • Ajoute la prise en charge de la gestion locale des ombres à l'aide d'IPC. • Ajoute la prise en charge de la synchronisation des ombres avec AWS IoT Core.
Authentification de l'appareil client	La version 2.0.0 du nouveau composant d'authentification de l'appareil client est disponible.

Composant	Détails
	Nouvelles fonctionnalités • Ajoute la prise en charge des appareils clients Greengrass, qui sont des appareils IoT locaux qui se connectent à un appareil principal via MQTT. • Ajoute la prise en charge de l'authentification et de l'autorisation des appareils clients et de leurs actions MQTT.
Courtier MQTT Moquette	La version 2.0.0 du nouveau composant du broker Moquette MQTT est disponible. Nouvelles fonctionnalités • Ajoute la prise en charge d'un broker Moquette MQTT local qui gère les communications avec les appareils clients.
Pont MQTT	La version 2.0.0 du nouveau composant de pont MQTT est disponible. Nouvelles fonctionnalités • Permet de relayer les messages entre le courtier MQTT local, le courtier local de publication/d'abonnement Greengrass et le courtier MQTT. AWS IoT Core
Détecteur IP	La version 2.0.0 du nouveau composant de détection IP est disponible. Nouvelles fonctionnalités • Permet de signaler les points de terminaison du broker MQTT local d'un périphérique principal au service AWS IoT Greengrass cloud afin que les appareils clients puissent se connecter.
Gestionnaire de journaux	La version 2.1.1 du composant du gestionnaire de journaux est disponible. Corrections de bogues et améliorations • Résout un problème en raison duquel la configuration du journal système n'était pas mise à jour dans certains cas.

Composant	Détails
Détection d'objets DLR	La version 2.1.2 de la détection d'objets DLR est disponible. Corrections de bogues et améliorations • Résout un problème de mise à l'échelle de l'image qui entraînait des cadres de délimitation inexacts dans les exemples de résultats d'inférence de détection d'objets DLR.
TensorFlow Détection d'objets allégée	La version 2.1.1 de la détection d'objets TensorFlow Lite est disponible. Corrections de bogues et améliorations • Résout un problème de mise à l'échelle de l'image qui entraînait des cadres de délimitation inexacts dans les résultats d'inférence de détection d'objets de Sample TensorFlow Lite.

Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.1.0 le 26 avril 2021

Cette version fournit la version 2.1.0 du composant Greengrass Nucleus et AWS met à jour les composants fournis.

Date de sortie : 26 avril 2021

Points forts de la publication

- Intégration entre Docker Hub et Amazon Elastic Container Registry (Amazon ECR) — Le nouveau composant du gestionnaire d'applications Docker vous permet de télécharger des images publiques ou privées depuis Amazon ECR. Vous pouvez également utiliser ce composant pour télécharger des images publiques depuis Docker Hub et AWS Marketplace. Pour de plus amples informations, veuillez consulter [Exécuter un conteneur Docker](#).
- Images Dockerfile et Docker pour le logiciel AWS IoT Greengrass principal : vous pouvez utiliser l'image Greengrass Docker pour l'exécuter dans AWS IoT Greengrass un conteneur Docker qui utilise Amazon Linux 2 comme système d'exploitation de base. Vous pouvez également utiliser le AWS IoT Greengrass Dockerfile pour créer votre propre image Greengrass. Pour de plus amples informations, veuillez consulter [Exécuter le logiciel AWS IoT Greengrass Core dans un conteneur Docker](#).

- Support de structures et de plateformes d'apprentissage automatique supplémentaires : vous pouvez déployer des exemples de composants d'inférence d'apprentissage automatique qui utilisent des modèles préentraînés pour effectuer la classification d'images d'échantillons et la détection d'objets à l'aide de TensorFlow Lite 2.5.0 et DLR 1.6.0. Cette version étend également la prise en charge des exemples de machine learning pour les appareils Armv8 (AArch64). Pour de plus amples informations, veuillez consulter [Exécuter l'inférence de Machine Learning](#).

Détails de la version

- [Mises à jour du support technique](#)
- [Mises à jour des composants publics](#)

Mises à jour du support technique

Plateforme	Détails
Docker	Un Dockerfile et une image Docker pour AWS IoT Greengrass sont désormais disponibles. Dockerfile AWS IoT Greengrass fournit un Dockerfile pour créer une image de conteneur sur laquelle le logiciel AWS IoT Greengrass Core et ses dépendances sont installés sur une image de base Amazon Linux 2 (x86_64). Vous pouvez modifier l'image de base dans le Dockerfile pour l'exécuter AWS IoT Greengrass sur une architecture de plate-forme différente. Image Docker AWS IoT Greengrass fournit une image Docker prédéfinie sur laquelle le logiciel AWS IoT Greengrass principal et ses dépendances sont installés sur une image de base Amazon Linux 2 (x86_64). Pour de plus amples informations, veuillez consulter Exécuter le logiciel AWS IoT Greengrass Core dans un conteneur Docker.

Mises à jour des composants publics

Le tableau suivant répertorie les composants AWS fournis qui incluent des fonctionnalités nouvelles et mises à jour.

Important

Lorsque vous déployez un composant, AWS IoT Greengrass installe les dernières versions prises en charge de toutes les dépendances de ce composant. De ce fait, les nouvelles versions de correctif des composants publics AWS fournis peuvent être automatiquement déployées sur vos appareils principaux si vous ajoutez de nouveaux appareils à un groupe d'objets ou si vous mettez à jour le déploiement qui cible ces appareils. Certaines mises à jour automatiques, telles que la mise à jour du noyau, peuvent provoquer le redémarrage inattendu de vos appareils.

Pour éviter les mises à jour involontaires d'un composant en cours d'exécution sur votre appareil, nous vous recommandons d'inclure directement votre version préférée de ce composant lorsque vous [créez un déploiement](#). Pour plus d'informations sur le comportement de mise à jour du logiciel AWS IoT Greengrass Core, consultez [Mettre à jour le logiciel AWS IoT Greengrass principal \(OTA\)](#).

Composant	Détails
Noyau de Greengrass	La version 2.1.0 du noyau Greengrass est disponible. Nouvelles fonctionnalités • Prend en charge le téléchargement d'images Docker à partir de référentiels privés sur Amazon ECR. • Ajoute les paramètres suivants pour personnaliser la configuration MQTT sur les appareils principaux : • <code>maxInFlightPublishes</code> — Le nombre maximum de messages MQTT QoS 1 non reconnus qui peuvent être envoyés en même temps. • <code>maxPublishRetry</code> — Le nombre maximum de fois que vous pouvez réessayer un message qui n'est pas publié.

Composant	Détails
	• Ajoute le paramètre de <code>fleetstatusservice</code> configuration pour configurer l'intervalle auquel le périphérique principal publie l'état du périphérique dans le AWS Cloud. • Corrections et améliorations mineures supplémentaires. Pour plus d'informations, consultez les versions sur GitHub. Corrections de bugs et améliorations • Résout un problème en raison duquel les déploiements fictifs étaient dupliqués lors du redémarrage du noyau. • Résout un problème qui provoquait le blocage du noyau lorsqu'il rencontrait une exception de chargement de service. • Améliore la résolution des dépendances entre les composants en cas d'échec d'un déploiement incluant une dépendance circulaire. • Résout un problème qui empêchait le redéploiement d'un composant de plug-in s'il avait été précédemment supprimé du périphérique principal. • Résolution d'un problème en raison duquel la variable d'HOMEenvironnement était définie <code>/greengrass/v2 /work</code> dans le répertoire des composants Lambda ou des composants exécutés en tant qu'utilisateur root. La HOME variable est désormais correctement définie dans le répertoire de base de l'utilisateur qui exécute le composant. • Corrections et améliorations mineures supplémentaires. Pour plus d'informations, consultez les versions sur GitHub.
Gestionnaire d'applications Docker	La version 2.0.0 du nouveau composant du gestionnaire d'applications Docker est disponible. Nouvelles fonctionnalités • Gère les informations d'identification pour télécharger des images depuis des référentiels privés sur Amazon ECR. • Télécharge des images publiques depuis Amazon ECR, Docker Hub et. AWS Marketplace

Composant	Détails
Lanceur Lambda	La version 2.0.4 du composant du lanceur Lambda est disponible. Corrections de bugs et améliorations • Résout un problème en raison duquel le composant ne passe pas correctement <code>AddGroupOwner</code> au conteneur de fonctions Lambda.
Routeur d'abonnement Legacy	La version 2.1.0 de l'ancien composant routeur d'abonnement est disponible. Corrections de bugs et améliorations • Permet de spécifier les noms des composants au lieu de ARNs for source et target. Si vous spécifiez un nom de composant pour un abonnement, vous n'avez pas besoin de reconfigurer l'abonnement chaque fois que la version de la fonction Lambda change.
Console de débogage locale	La version 2.1.0 du composant de console de débogage local est disponible. Nouvelles fonctionnalités • Utilise le protocole HTTPS pour sécuriser votre connexion à la console de débogage locale. Le protocole HTTPS est activé par défaut. Corrections de bugs et améliorations • Vous pouvez ignorer les messages de la barre flash dans l'éditeur de configuration.

Composant	Détails
Gestionnaire de journaux	La version 2.1.0 du composant du gestionnaire de journaux est disponible. Corrections de bugs et améliorations • Utilisez des valeurs par défaut pour <code>logFileDirectoryPath</code> et <code>logFileRegex</code> qui fonctionnent pour les composants Greengrass qui impriment en sortie standard (stdout) et en erreur standard (stderr). • Acheminez correctement le trafic via un proxy réseau configuré lors du téléchargement des CloudWatch journaux vers Logs. • Gérez correctement les caractères deux-points (:) dans les noms des flux de log. CloudWatch Les noms des flux de journaux ne prennent pas en charge les deux-points. • Simplifiez les noms des flux de journaux en supprimant les noms de groupes d'objets du flux de journaux. • Supprimez un message du journal des erreurs qui s'imprime lors d'un comportement normal.

Composant	Détails
Classification des images DLR	La version 2.1.1 du composant de classification d'images DLR est disponible. Nouvelles fonctionnalités • Utilisez Deep Learning Runtime v1.6.0. • Ajout de la prise en charge de la classification des exemples d'images sur les plateformes Armv8 (AArch64). Cela étend la prise en charge de l'apprentissage automatique pour les appareils principaux de Greengrass exécutant NVIDIA Jetson, tels que le Jetson Nano. • Activez l'intégration de la caméra pour l'inférence d'échantillons. Utilisez le nouveau paramètre <code>UseCamera</code> de configuration pour permettre à l'exemple de code d'inférence d'accéder à la caméra de votre appareil principal Greengrass et d'exécuter l'inférence localement sur l'image capturée. • Ajoutez la prise en charge de la publication des résultats d'inférence au AWS Cloud. Utilisez le nouveau paramètre <code>PublishResultsOnTopic</code> de configuration pour spécifier le sujet sur lequel vous souhaitez publier les résultats. • Ajoutez le nouveau paramètre <code>ImageDirectory</code> de configuration qui vous permet de spécifier un répertoire personnalisé pour l'image sur laquelle vous souhaitez effectuer une inférence. Corrections de bugs et améliorations • Écrivez les résultats d'inférence dans le fichier journal du composant plutôt que dans un fichier d'inférence distinct. • Utilisez le module de journalisation du logiciel AWS IoT Greengrass Core pour enregistrer la sortie des composants. • Utilisez le Kit SDK des appareils AWS IoT pour lire la configuration du composant et appliquer les modifications de configuration.

Composant	Détails
Détection d'objets DLR	La version 2.1.1 du composant de détection d'objets DLR est disponible. Nouvelles fonctionnalités • Utilisez Deep Learning Runtime v1.6.0. • Ajout de la prise en charge de la détection d'échantillons d'objets sur les plateformes Armv8 (AArch64). Cela étend la prise en charge de l'apprentissage automatique pour les appareils principaux de Greengrass exécutant NVIDIA Jetson, tels que le Jetson Nano. • Activez l'intégration de la caméra pour l'inférence d'échantillons. Utilisez le nouveau paramètre <code>UseCamera</code> de configuration pour permettre à l'exemple de code d'inférence d'accéder à la caméra de votre appareil principal Greengrass et d'exécuter l'inférence localement sur l'image capturée. • Ajoutez la prise en charge de la publication des résultats d'inférence au AWS Cloud. Utilisez le nouveau paramètre <code>PublishResultsOnTopic</code> de configuration pour spécifier le sujet sur lequel vous souhaitez publier les résultats. • Ajoutez le nouveau paramètre <code>ImageDirectory</code> de configuration qui vous permet de spécifier un répertoire personnalisé pour l'image sur laquelle vous souhaitez effectuer une inférence. Corrections de bugs et améliorations • Écrivez les résultats d'inférence dans le fichier journal du composant plutôt que dans un fichier d'inférence distinct. • Utilisez le module de journalisation du logiciel AWS IoT Greengrass Core pour enregistrer la sortie des composants. • Utilisez le Kit SDK des appareils AWS IoT pour lire la configuration du composant et appliquer les modifications de configuration.

Composant	Détails
magasin de modèles de classification d'images DLR	La version 2.1.1 du composant Model Store de classification d'images DLR est disponible. Nouvelles fonctionnalités • Ajoutez un exemple de modèle de classification d'images ResNet -50 pour les plateformes Armv8 (AArch64). Cela étend la prise en charge de l'apprentissage automatique pour les appareils principaux de Greengrass exécutant NVIDIA Jetson, tels que le Jetson Nano.
Model Store dédié à la détection d'objets DLR	La version 2.1.1 du composant Model Store de détection d'objets DLR est disponible. Nouvelles fonctionnalités • Ajoutez un exemple de modèle de détection d'YOLOv3 objets pour les plateformes Armv8 (AArch64). Cela étend la prise en charge de l'apprentissage automatique pour les appareils principaux de Greengrass exécutant NVIDIA Jetson, tels que le Jetson Nano.
Installeur DLR	La version 1.6.1 du composant DLR est disponible. Nouvelles fonctionnalités • Installez Deep Learning Runtime v1.6.0 et ses dépendances. • Ajout du support pour l'installation du DLR sur les plateformes Armv8 (AArch64). Cela étend la prise en charge de l'apprentissage automatique pour les appareils principaux de Greengrass exécutant NVIDIA Jetson, tels que le Jetson Nano. Corrections de bugs et améliorations • Installez-le Kit SDK des appareils AWS IoT dans l'environnement virtuel pour lire la configuration des composants et appliquer les modifications de configuration. • Corrections de bogues et améliorations mineures supplémentaires.

Composant	Détails
TensorFlow Classification d'images Lite	La version 2.1.0 du nouveau composant de classification d'images TensorFlow Lite est disponible. Nouvelles fonctionnalités Ajoutez la prise en charge de l'inférence de classification d'exemples d'images à l'aide de TensorFlow Lite.
TensorFlow Détection d'objets allégée	La version 2.1.0 du nouveau composant de détection d'objets TensorFlow Lite est disponible. Nouvelles fonctionnalités Ajoutez la prise en charge de l'inférence de détection d'échantillons d'objets à l'aide de TensorFlow Lite.
TensorFlow Boutique de modèles de classification d'images Lite	La version 2.1.0 du nouveau composant de magasin de modèles de classification d'images TensorFlow Lite est disponible. Nouvelles fonctionnalités Fournissez un modèle quantifié MobileNet v1 pré-entraîné pour l'inférence de classification d'échantillons d'images à l'aide de Lite. TensorFlow
TensorFlow Boutique de modèles de détection d'objets Lite	La version 2.1.0 du nouveau composant TensorFlow Lite de détection d'objets Model Store est disponible. Nouvelles fonctionnalités Fournissez un MobileNet modèle SSD (Single Shot Detection) préentraîné basé sur le jeu de données COCO pour l'inférence de détection d'échantillons d'objets à l'aide TensorFlow de Lite.
TensorFlow Lite	La version 2.5.0 du nouveau composant TensorFlow Lite est disponible. Nouvelles fonctionnalités Installez TensorFlow Lite v1.6.0 et ses dépendances dans un environnement virtuel sur les plateformes Armv7, Armv8 (AArch64) et x86_64.

Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.0.5 le 09 mars 2021

Cette version fournit la version 2.0.5 du composant Greengrass Nucleus et AWS met à jour les composants fournis. Il résout un problème lié à la prise en charge des proxys réseau et un problème lié au point de terminaison du plan de données Greengrass dans les régions de AWS Chine.

Date de sortie : 09 mars 2021

Mises à jour des composants publics

Le tableau suivant répertorie les composants AWS fournis qui incluent des fonctionnalités nouvelles et mises à jour.

Important

Lorsque vous déployez un composant, AWS IoT Greengrass installe les dernières versions prises en charge de toutes les dépendances de ce composant. De ce fait, les nouvelles versions de correctif des composants publics AWS fournis peuvent être automatiquement déployées sur vos appareils principaux si vous ajoutez de nouveaux appareils à un groupe d'objets ou si vous mettez à jour le déploiement qui cible ces appareils. Certaines mises à jour automatiques, telles que la mise à jour du noyau, peuvent provoquer le redémarrage inattendu de vos appareils.

Pour éviter les mises à jour involontaires d'un composant en cours d'exécution sur votre appareil, nous vous recommandons d'inclure directement votre version préférée de ce composant lorsque vous [créez un déploiement](#). Pour plus d'informations sur le comportement de mise à jour du logiciel AWS IoT Greengrass Core, consultez [Mettre à jour le logiciel AWS IoT Greengrass principal \(OTA\)](#).

Composant	Détails
Noyau de Greengrass	La version 2.0.5 du noyau Greengrass est disponible. Corrections de bogues et améliorations Achemine correctement le trafic via un proxy réseau configuré lors du téléchargement des composants AWS fournis.

Composant	Détails
	Utilisez le point de terminaison du plan de données Greengrass approprié dans les régions de AWS Chine.

Sortie : mise à jour logicielle AWS IoT Greengrass Core v2.0.4 le 04 février 2021

Cette version fournit la version 2.0.4 du composant Greengrass nucleus. Il inclut le nouveau `greengrassDataPlanePort` paramètre permettant de configurer la communication HTTPS sur le port 443 et corrige les bogues. La politique IAM minimale nécessite désormais le `iam:GetPolicy` et `sts:GetCallerIdentity` quand le programme d'installation du logiciel AWS IoT Greengrass Core est exécuté avec `--provision true`.

Date de sortie : 04 février 2021

Mises à jour des composants publics

Le tableau suivant répertorie les composants AWS fournis qui incluent des fonctionnalités nouvelles et mises à jour.

Important

Lorsque vous déployez un composant, AWS IoT Greengrass installe les dernières versions prises en charge de toutes les dépendances de ce composant. De ce fait, les nouvelles versions de correctif des composants publics AWS fournis peuvent être automatiquement déployées sur vos appareils principaux si vous ajoutez de nouveaux appareils à un groupe d'objets ou si vous mettez à jour le déploiement qui cible ces appareils. Certaines mises à jour automatiques, telles que la mise à jour du noyau, peuvent provoquer le redémarrage inattendu de vos appareils.

Pour éviter les mises à jour involontaires d'un composant en cours d'exécution sur votre appareil, nous vous recommandons d'inclure directement votre version préférée de ce composant lorsque vous [créez un déploiement](#). Pour plus d'informations sur le comportement de mise à jour du logiciel AWS IoT Greengrass Core, consultez [Mettre à jour le logiciel AWS IoT Greengrass principal \(OTA\)](#).

Composant	Détails
Noyau de Greengrass	La version 2.0.4 du noyau Greengrass est disponible. Nouvelles fonctionnalités • Active le trafic HTTPS sur le port 443. Vous pouvez utiliser le nouveau paramètre de <code>greengrassDataPlanePort</code> configuration de la version 2.0.4 du composant Nucleus pour configurer la communication HTTPS afin qu'elle passe par le port 443 au lieu du port par défaut 8443. Pour de plus amples informations, veuillez consulter Configuration du protocole HTTPS sur le port 443. • Ajoute la variable de recette du chemin de travail. Vous pouvez utiliser cette variable de recette pour obtenir le chemin d'accès aux dossiers de travail des composants, que vous pouvez utiliser pour partager des fichiers entre les composants et leurs dépendances. Pour plus d'informations, consultez la variable de recette du chemin de travail. Corrections de bogues et améliorations • Empêche la création de la politique de rôle d'échange de jetons Gestion des identités et des accès AWS (IAM) si une stratégie de rôle existe déjà. À la suite de cette modification, le programme d'installation nécessite désormais le <code>iam:GetPolicy</code> et <code>sts:GetCallerIdentity</code> lorsqu'il est exécuté avec <code>--provision true</code>. Pour de plus amples informations, veuillez consulter Politique IAM minimale permettant au programme d'installation de provisionner les ressources. • Gère correctement l'annulation d'un déploiement qui n'a pas encore été enregistré avec succès. • Met à jour la configuration pour supprimer les anciennes entrées avec des horodatages plus récents lors de l'annulation d'un déploiement. • Corrections et améliorations mineures supplémentaires. Pour plus d'informations, consultez les communiqués publiés sur GitHub.

Migrer depuis AWS IoT Greengrass la version 1

AWS IoT Greengrass Version 2 est une version majeure du logiciel AWS IoT Greengrass Core et APIs de la console. AWS IoT Greengrass V2 introduit plusieurs améliorations AWS IoT Greengrass V1, telles que les applications modulaires, les déploiements sur de grands parcs d'appareils et la prise en charge de plateformes supplémentaires.

Note

Avis de fin de support : le 7 octobre 2026, AWS le support de AWS IoT Greengrass Version 1. Après le 7 octobre 2026, vous ne pourrez plus accéder à la AWS IoT Greengrass V1 console ni aux AWS IoT Greengrass V1 ressources.

Suivez les instructions de ce guide pour effectuer la migration de AWS IoT Greengrass V1 vers AWS IoT Greengrass V2.

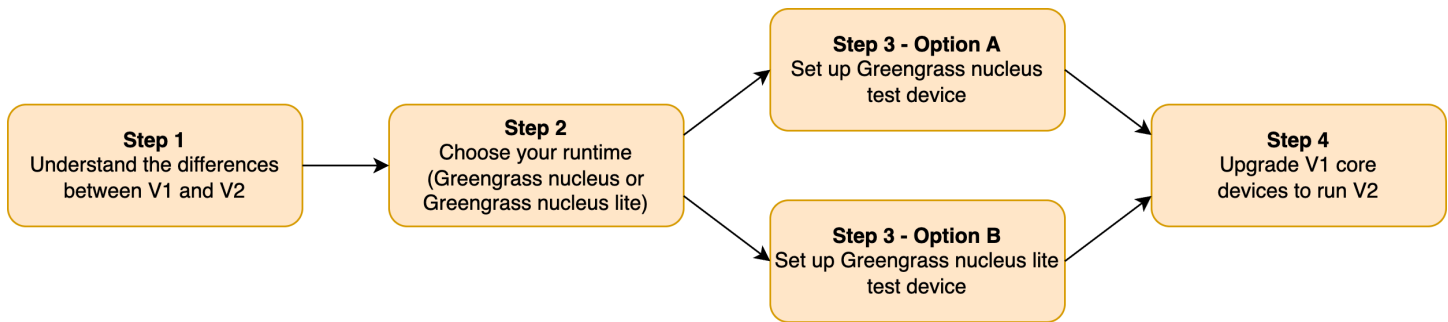
Présentation de la migration

À un niveau supérieur, vous pouvez utiliser la procédure suivante pour mettre à niveau les périphériques principaux de AWS IoT Greengrass V1 vers AWS IoT Greengrass V2.

Avant de procéder à la migration, vous devez choisir entre deux options d'exécution :

- Greengrass nucleus (réduction des efforts de migration, prise en charge complète des fonctionnalités)
- Greengrass Nucleus Lite (effort de migration plus important, conçu pour les appareils aux ressources limitées).

La procédure exacte que vous devez suivre dépend des ressources de votre appareil, des fonctionnalités requises et des exigences environnementales spécifiques.



1. [Comprendre les différences entre la V1 et la V2](#)

AWS IoT Greengrass V2 introduit de nouveaux concepts fondamentaux pour les flottes d'appareils et les logiciels déployables, et la V2 simplifie plusieurs concepts de la V1.

Le service AWS IoT Greengrass V2 cloud et le logiciel AWS IoT Greengrass Core v2.x ne sont pas rétrocompatibles avec le service AWS IoT Greengrass V1 cloud et le logiciel AWS IoT Greengrass Core v1.x. Par conséquent, les mises à jour AWS IoT Greengrass V1 over-the-air (OTA) ne peuvent pas mettre à niveau les appareils principaux de la V1 à la V2.

2. [Choisissez votre environnement d'exécution \(Greengrass nucleus ou Greengrass nucleus lite\)](#)

Choisissez entre Greengrass nucleus ou Greengrass nucleus lite en fonction des ressources de votre appareil et des fonctionnalités requises :

- Trajectoire du noyau de Greengrass : réduction de l'effort de migration. Les fonctions Lambda peuvent être importées en tant que composants Lambda avec des modifications de code minimales. Supporte les fonctionnalités V1 (service parallèle local, appareils clients, connecteurs).
- Trajectoire Greengrass Nucleus Lite : effort de migration accru. Les fonctions Lambda doivent être converties en composants génériques, ce qui nécessite des modifications de code pour utiliser le SDK Kit SDK des appareils AWS IoT V2/AWS IoT Greengrass Component au lieu du SDK Core. AWS IoT Greengrass Ne prend pas en charge le service parallèle local, les appareils clients ou les connecteurs.

3. [Configurer un nouvel appareil pour tester les applications V1 sur V2](#)

Pour minimiser les risques pour vos appareils en production, créez un nouvel appareil pour tester vos applications V1 sur V2. Choisissez le guide de configuration en fonction de votre sélection d'environnement d'exécution :

- Option A - Runtime Greengrass Nucleus : [configurez un nouvel appareil pour tester les applications V1 sur V2](#). Importez des fonctions Lambda en tant que composants Lambda avec des modifications de code minimales.
- Option B - Runtime Greengrass nucleus lite : [configurez un nouvel appareil pour tester les applications V1 sur V2 \(Greengrass nucleus lite\)](#). Convertissez les fonctions Lambda en composants génériques à l'aide de. Kit SDK des appareils AWS IoT

4. [Mettre à niveau les appareils principaux de la V1 pour exécuter la V2](#)

Après le test sur un nouvel appareil, mettez à niveau vos appareils principaux V1 existants pour exécuter le logiciel AWS IoT Greengrass Core v2.x et AWS IoT Greengrass V2 ses composants. Pour migrer un parc d'appareils de la V1 à la V2, vous devez répéter cette étape pour chaque appareil du parc.

Différences entre AWS IoT Greengrass V1 et AWS IoT Greengrass V2

AWS IoT Greengrass V2 introduit de nouveaux concepts fondamentaux pour les appareils, les flottes et les logiciels déployables. Cette section décrit les concepts de la V1 qui sont différents dans la V2.

Concepts et terminologie de Greengrass

Concept	AWS IoT Greengrass V1	AWS IoT Greengrass V2
Code de l'application	Dans AWS IoT Greengrass V1, les fonctions Lambda définissent le logiciel qui s'exécute sur les appareils principaux. Dans chaque groupe Greengrass, vous définissez les abonnements et les ressources locales que la fonction utilise. Pour les fonctions Lambda que le logiciel AWS IoT Greengrass Core exécute dans un environnement d'exécution	Dans AWS IoT Greengrass V2, les composants sont les modules logiciels qui s'exécutent sur les appareils principaux. • Chaque composant possède une recette qui définit les métadonnées, les paramètres, les dépendances et les scripts du composant à exécuter à chaque étape du cycle de vie du composant.

Concept	AWS IoT Greengrass V1	AWS IoT Greengrass V2
	Lambda conteneurisé, vous définissez les paramètres du conteneur, tels que les limites de mémoire.	• La recette définit également les artefacts du composant, qui sont des fichiers binaires, tels que des scripts, du code compilé et des ressources statiques. • Lorsque vous déployez un composant sur un périphérique principal, celui-ci télécharge la recette du composant et les artefacts pour exécuter le composant. Vous pouvez importer vos fonctions Lambda V1 en tant que composants exécutés dans un environnement d'exécution Lambda dans AWS IoT Greengrass V2. Lorsque vous importez la fonction Lambda, vous spécifiez les abonnements, les ressources locales et les paramètres de conteneur pour la fonction. Pour de plus amples informations, veuillez consulter Étape 2 : créer et déployer des AWS IoT Greengrass V2 composants pour migrer AWS IoT Greengrass V1 des applications. Pour plus d'informations sur la création de composant

Concept	AWS IoT Greengrass V1	AWS IoT Greengrass V2
		s personnalisés, consultez Développer des AWS IoT Greengrass composants .

Concept	AWS IoT Greengrass V1	AWS IoT Greengrass V2
AWS IoT Greengrass groupes et déploiements	Dans AWS IoT Greengrass V1, un groupe définit le périphérique principal, les paramètres et le logiciel de ce périphérique principal, ainsi que la liste des AWS IoT éléments pouvant se connecter à ce périphérique principal. Vous créez un déploiement pour envoyer la configuration d'un groupe à un appareil principal.	Dans AWS IoT Greengrass V2, vous utilisez les déploiements pour définir les composants logiciels et les configurations qui s'exécutent sur les appareils principaux. • Chaque déploiement cible un seul périphérique principal (qui est un AWS IoT objet) ou un AWS IoT groupe d'objets pouvant contenir plusieurs périphériques principaux. • Les déploiements vers des groupes d'objets sont continus. Ainsi, lorsque vous ajoutez un appareil principal à un groupe d'objets, il reçoit la configuration logicielle de ce groupe. Pour de plus amples informations, veuillez consulter Déployer AWS IoT Greengrass des composants sur des appareils. Dans AWS IoT Greengrass V2, vous pouvez également créer des déploiements locaux à l'aide de la CLI Greengrass pour tester des composants logiciels personnalisés sur l'appareil sur lequel vous les

Concept	AWS IoT Greengrass V1	AWS IoT Greengrass V2
		développez. Pour de plus amples informations, veuillez consulter Création de AWS IoT Greengrass composants .

Concept	AWS IoT Greengrass V1	AWS IoT Greengrass V2
AWS IoT Greengrass Logiciel de base	Dans AWS IoT Greengrass V1, le logiciel AWS IoT Greengrass Core est un package unique qui contient le logiciel et toutes ses fonctionnalités. L'appareil périphérique sur lequel vous installez le logiciel AWS IoT Greengrass Core est appelé noyau Greengrass.	Dans AWS IoT Greengrass V2, le logiciel AWS IoT Greengrass Core est modulaire, de sorte que vous pouvez choisir ce que vous souhaitez installer pour contrôler l'empreinte mémoire. - Le composant Greengrass nucleus est l'installation minimale requise du logiciel AWS IoT Greengrass Core. Le périphérique périphérique sur lequel vous installez le noyau est appelé périphérique central Greengrass. - Le noyau gère les déploiements, l'orchestration et la gestion du cycle de vie des autres composants du périphérique principal. - Les fonctionnalités telles que le gestionnaire de flux, le gestionnaire de secrets et le gestionnaire de journaux sont des composants que vous déployez uniquement lorsque vous en avez besoin. Pour de plus amples informations, veuillez consulter AWS-composants fournis.

Concept	AWS IoT Greengrass V1	AWS IoT Greengrass V2
Connecteurs	Dans AWS IoT Greengrass V1, les connecteurs sont des modules prédéfinis que vous déployez sur les appareils AWS IoT Greengrass V1 principaux pour interagir avec l'infrastructure locale, les protocoles des appareils et d'autres services cloud. AWS	In AWS IoT Greengrass V2, AWS fournit des composants Greengrass qui implémentent les fonctionnalités fournies par les connecteurs dans la version 1. Les AWS IoT Greengrass V2 composants suivants fournissent les fonctionnalités du connecteur Greengrass V1 : • CloudWatch composant de métriques • AWS IoT Device Defender composant • Composant Firehose • Composant adaptateur de protocole Modbus-RTU • Composant Amazon SNS Pour de plus amples informations, veuillez consulter AWS-composants fournis.

Concept	AWS IoT Greengrass V1	AWS IoT Greengrass V2
Appareils connectés (appareils Greengrass)	Dans AWS IoT Greengrass V1, les appareils connectés sont AWS IoT des éléments que vous ajoutez à un groupe Greengrass pour vous connecter au périphérique principal de ce groupe et communiquer via MQTT. Vous devez déployer ce groupe chaque fois que vous ajoutez ou supprimez un appareil connecté. Vous utilisez des abonnements pour relayer des messages entre les AWS IoT Core appareils connectés et les applications de l'appareil principal.	Dans AWS IoT Greengrass V2, les appareils connectés sont appelés appareils clients Greengrass. • Vous associez les appareils clients aux appareils principaux pour les connecter et communiquer via MQTT. • Pour autoriser les appareils clients à se connecter, vous définissez des politiques d'autorisation qui peuvent s'appliquer à des groupes d'appareils clients. Vous n'avez donc pas besoin de créer un déploiement pour ajouter ou supprimer un appareil client. • Pour relayer des messages entre les appareils clients et les composants Greengrass, vous pouvez configurer un composant de pont MQTT optionnel. AWS IoT Core Dans AWS IoT Greengrass V1 les deux cas AWS IoT Greengrass V2, les appareils peuvent exécuter FreeRTOS ou utiliser l'Kit SDK des appareils AWS IoTAPI de

Concept	AWS IoT Greengrass V1	AWS IoT Greengrass V2
		découverte Greengrass pour obtenir des informations sur les principaux appareils auxquels ils peuvent se connecter. L'API de découverte Greengrass est rétrocompatible. Ainsi, si vous avez des appareils clients qui se connectent à un appareil principal V1, vous pouvez les connecter à un appareil principal V2 sans modifier leur code. Pour plus d'informations sur les appareils clients, consultez Interagissez avec les appareils IoT locaux.

Concept	AWS IoT Greengrass V1	AWS IoT Greengrass V2
Ressources locales	Dans AWS IoT Greengrass V1, les fonctions Lambda exécutées dans des conteneurs peuvent être configurées pour accéder aux volumes et aux périphériques du système de fichiers du périphérique principal. Ces ressources du système de fichiers sont appelées ressources locales.	Dans AWS IoT Greengrass V2, vous pouvez exécuter des composants qui sont des fonctions Lambda, des conteneurs Docker, des processus de système d'exploitation natifs ou des environnements d'exécution personnalisés. • Lorsque vous importez une fonction Lambda conteneurisée en tant que composant, vous devez spécifier les ressources locales utilisées par la fonction. • Les fonctions Lambda non conteneurisées et les composants non Lambda peuvent fonctionner directement avec les ressources locales des appareils principaux. Vous n'avez donc pas besoin de spécifier les ressources locales utilisées par le composant.

Concept	AWS IoT Greengrass V1	AWS IoT Greengrass V2
Service parallèle local	Dans AWS IoT Greengrass V1, le service d'ombre local est activé par défaut et ne prend en charge que les ombres classiques sans nom. Vous utilisez le SDK AWS IoT Greengrass Core dans vos fonctions Lambda pour interagir avec les ombres de vos appareils.	Dans AWS IoT Greengrass V2, vous activez le service fantôme local en déployant le composant Shadow Manager. • Vous pouvez utiliser la Kit SDK des appareils AWS IoT V2 dans les fonctions Lambda et les composants personnalisés pour interagir avec les ombres de vos appareils. • Le service d'ombre local prend en charge les ombres nommées. • Le service local d'ombres vous permet de supprimer les ombres et de synchroniser les ombres supprimées avec AWS IoT Core. Pour de plus amples informations, veuillez consulter Interagissez avec les ombres de l'appareil.

Concept	AWS IoT Greengrass V1	AWS IoT Greengrass V2
Abonnements	Dans AWS IoT Greengrass V1, vous définissez les abonnements d'un groupe Greengrass afin de spécifier les canaux de communication entre les fonctions Lambda, les connecteurs, les appareils connectés, le broker AWS IoT Core MQTT et le service parallèle local. Les abonnements spécifient l'endroit où les fonctions Lambda reçoivent des messages d'événement à consommer sous forme de charges utiles de fonction.	Dans AWS IoT Greengrass V2, vous spécifiez les canaux de communication sans utiliser d'abonnements. • Les composants gèrent leurs propres canaux de communication pour interagir avec les messages locaux de publication/d'abonnement, les messages AWS IoT Core MQTT et le service parallèle local. • Pour développer un composant qui réagit aux messages provenant d'un autre composant ou du broker AWS IoT Core MQTT, vous pouvez utiliser des interfaces de communication interprocessus (IPC) pour les messages locaux de publication/d'abonnement et les messages MQTT.AWS IoT Core • Pour développer un composant qui interagit avec le service parallèle local, vous pouvez utiliser l'interface IPC du service parallèle local. • Dans la configuration du composant, vous

Concept	AWS IoT Greengrass V1	AWS IoT Greengrass V2
		définissez des politiques d'autorisation pour spécifier les sujets et les ombres locales que le composant est autorisé à utiliser. • Pour configurer les canaux de communication entre les appareils clients, le courtier de publication/d'abonnement local et le courtier AWS IoT Core MQTT, vous devez configurer et déployer le composant du pont MQTT. Le composant de pont MQTT vous permet d'interagir avec les appareils clients dans les composants et de relayer des messages entre les appareils clients et AWS IoT Core.

Concept	AWS IoT Greengrass V1	AWS IoT Greengrass V2
Accès à d'autres Services AWS	Dans AWS IoT Greengrass V1, vous attachez un rôle Gestion des identités et des accès AWS (IAM), appelé rôle de groupe, à un groupe Greengrass. Le rôle de groupe définit les autorisations que les fonctions et AWS IoT Greengrass fonctionnalités de Lambda sur l'appareil principal de ce groupe utilisent pour y accéder. Services AWS	Dans AWS IoT Greengrass V2, vous associez un alias de AWS IoT rôle à un appareil principal Greengrass. L'alias de rôle pointe vers un rôle IAM appelé rôle d'échange de jetons. Le rôle d'échange de jetons définit les autorisations que les composants Greengrass du périphérique principal utilisent pour y accéder. Services AWS Pour de plus amples informations, veuillez consulter Autoriser les appareils principaux à interagir avec les AWS services .

Choisissez votre environnement d'exécution (Greengrass nucleus ou Greengrass nucleus lite)

Le choix entre Greengrass nucleus et Greengrass nucleus lite dépend des ressources de votre appareil et des fonctionnalités utilisées par vos fonctions Lambda. Consultez la matrice de compatibilité des sources d'événements dans le tableau suivant, puis utilisez le diagramme du flux décisionnel pour déterminer quel environnement d'exécution convient à votre migration. [Pour une comparaison détaillée des fonctionnalités de Greengrass Nucleus et de Greengrass Nucleus Lite, consultez Choisir votre environnement d'exécution.](#)

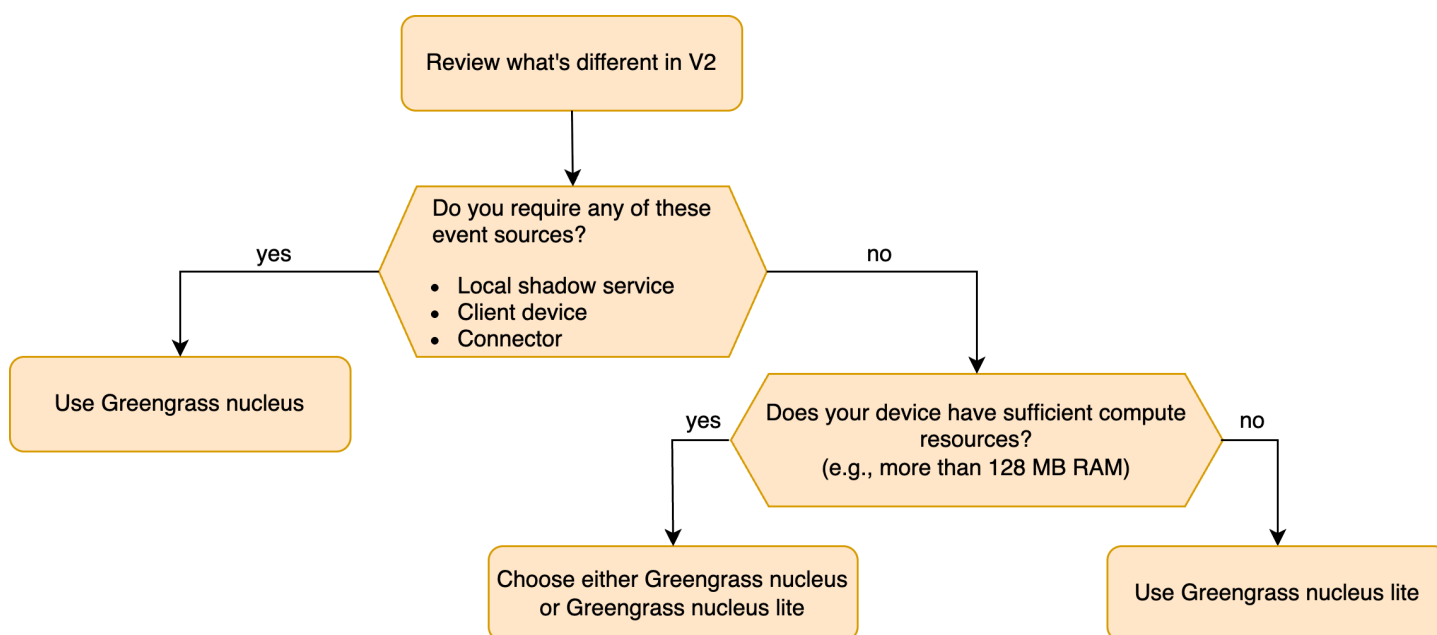
Matrice de compatibilité des sources d'événements

Dans AWS IoT Greengrass V1, les fonctions Lambda peuvent communiquer avec cinq types de sources d'événements : les autres fonctions Lambda, le service parallèle local AWS IoT Core, les appareils clients et les connecteurs. Le tableau suivant indique quelles sources d'événements sont prises en charge dans chaque environnement d'exécution de la version 2.

Remarque : Les noms des sources d'événements utilisent AWS IoT Greengrass V1 une terminologie. Lors de la migration vers la version V2, les fonctions Lambda sont converties en composants Lambda (pris en charge uniquement dans le noyau Greengrass) ou en composants génériques (pris en charge à la fois dans Greengrass nucleus et Greengrass nucleus lite).

Source de l'événement	Noyau de Greengrass	Greengrass Nucleus Lite
Autres fonctions Lambda du groupe	✓ (composants Lambda et composants génériques)	✓ (composants génériques uniquement)
AWS IoT Core service	✓	✓
Service parallèle local	✓	x
Appareil client	✓	x
Connecteur	✓	x

Flux décisionnel de sélection du temps d'exécution



Remarques

1. [Pour connaître les exigences de Greengrass Nucleus Lite et les détails de compatibilité, consultez Greengrass Nucleus Lite.](#) Greengrass nucleus lite nécessite un minimum de 5 Mo de RAM et est conçu pour les appareils aux ressources limitées.
2. Le flux décisionnel fournit des conseils basés sur des cas d'utilisation typiques, mais il ne s'agit pas d'une exigence stricte. Les clients disposant d'appareils dont les ressources sont limitées ou suffisantes peuvent choisir d'utiliser un seul environnement d'exécution sur tous les appareils pour des raisons de simplicité opérationnelle, même si certains appareils peuvent prendre en charge l'un ou l'autre environnement d'exécution.

Étapes suivantes

Après avoir choisi votre environnement d'exécution, procédez à la configuration de votre appareil de test :

- Pour Greengrass Nucleus Runtime : [configurer un nouvel appareil pour tester les applications V1 sur V2](#)
- Pour Greengrass Nucleus Lite Runtime : [configurer un nouvel appareil avec Greengrass](#) Nucleus Lite

Configuration d'un nouvel appareil pour tester les applications V1 sur V2

Pour minimiser les risques pour vos appareils en production, créez un nouvel appareil pour tester vos applications V1 sur V2 avant de mettre à niveau vos appareils de production.

Choisissez l'un des guides de configuration suivants en fonction de votre sélection d'environnement d'exécution à l'étape précédente :

- Option A - Runtime du noyau Greengrass : suivez [Configuration d'un nouvel appareil pour tester les applications V1 sur V2](#) si vous avez choisi Greengrass nucleus. Cette option vous permet d'importer des fonctions Lambda en tant que composants Lambda avec un minimum de modifications de code et de prendre en charge les fonctionnalités de la version 1 telles que le service parallèle local, les appareils clients et les connecteurs.

- Option B - Runtime de Greengrass Nucleus Lite : À suivre [Configurer un nouvel appareil pour tester les applications V1 sur V2 \(Greengrass nucleus lite\)](#) si vous avez choisi Greengrass Nucleus Lite. Cette option nécessite de convertir les fonctions Lambda en composants génériques à l'aide de la Kit SDK des appareils AWS IoT version 2 ou du SDK de AWS IoT Greengrass composants, mais elle est optimisée pour les appareils aux ressources limitées.

Configuration d'un nouvel appareil pour tester les applications V1 sur V2

Configurez un nouveau périphérique AWS IoT Greengrass V2 principal pour déployer et tester les composants et les AWS Lambda fonctions AWS fournis pour vos AWS IoT Greengrass V1 applications. Vous pouvez également utiliser cet appareil principal V2 pour développer et tester des composants Greengrass personnalisés supplémentaires qui exécutent des processus natifs sur les appareils principaux. Après avoir testé vos applications sur un appareil principal V2, vous pouvez mettre à niveau vos appareils principaux V1 existants vers le V2 et déployer les composants V2 qui fournissent les fonctionnalités de votre V1.

Étape 1 : Installation AWS IoT Greengrass V2 sur un nouvel appareil

Installez le logiciel AWS IoT Greengrass Core v2.x sur un nouvel appareil. Vous pouvez suivre le [didacticiel de démarrage](#) pour configurer un appareil et apprendre à développer et à déployer des composants. Ce didacticiel utilise le [provisionnement automatique](#) pour configurer rapidement un appareil. Lorsque vous installez le logiciel AWS IoT Greengrass Core v2.x, spécifiez l'`--deploy-tool` argument pour déployer la [CLI](#) Greengrass, afin de pouvoir développer, tester et déboguer des composants directement sur le périphérique. Pour plus d'informations sur les autres options d'installation, notamment sur la façon d'installer le logiciel AWS IoT Greengrass Core derrière un proxy ou à l'aide d'un module de sécurité matériel (HSM), consultez [Installation du logiciel AWS IoT Greengrass de base](#).

(Facultatif) Activer la connexion à Amazon CloudWatch Logs

Pour permettre à un appareil principal V2 de télécharger des CloudWatch journaux sur Amazon Logs, vous pouvez déployer le [composant de gestionnaire de journaux AWS](#) fourni. Vous pouvez utiliser CloudWatch les journaux pour consulter les journaux des composants, afin de pouvoir déboguer et résoudre les problèmes sans accéder au système de fichiers du périphérique principal. Pour de plus amples informations, veuillez consulter [AWS IoT Greengrass Journaux de surveillance](#).

Étape 2 : créer et déployer des AWS IoT Greengrass V2 composants pour migrer AWS IoT Greengrass V1 des applications

Vous pouvez exécuter la plupart AWS IoT Greengrass V1 des applications sur AWS IoT Greengrass V2. Vous pouvez importer des fonctions Lambda en tant que composants qui s'exécutent sur AWS IoT Greengrass V2, et vous pouvez utiliser des [composants AWS fournis](#) qui offrent les mêmes fonctionnalités que les connecteurs. AWS IoT Greengrass

Vous pouvez également développer des composants personnalisés pour créer n'importe quelle fonctionnalité ou environnement d'exécution à exécuter sur les appareils principaux de Greengrass. Pour plus d'informations sur la façon de développer et de tester des composants localement, consultez [Création de AWS IoT Greengrass composants](#).

Rubriques

- [Importer les fonctions Lambda de la V1](#)
- [Utiliser des connecteurs V1](#)
- [Exécuter des conteneurs Docker](#)
- [Connectez les appareils Greengrass V1](#)
- [Activer le service parallèle local](#)
- [Intégrez avec AWS IoT SiteWise](#)

Importer les fonctions Lambda de la V1

Vous pouvez importer des fonctions Lambda en tant que AWS IoT Greengrass V2 composants. Choisissez l'une des approches suivantes :

- Import V1 Lambda fonctionne directement en tant que composants Greengrass.
- Mettez à jour vos fonctions Lambda pour utiliser les bibliothèques Greengrass de la Kit SDK des appareils AWS IoT version v2, puis importez les fonctions Lambda en tant que composants Greengrass.
- Créez des composants personnalisés qui utilisent du code autre que Lambda et la Kit SDK des appareils AWS IoT version v2 pour implémenter les mêmes fonctionnalités que vos fonctions Lambda.

Si votre fonction Lambda utilise des fonctionnalités, telles que le gestionnaire de flux ou les secrets locaux, vous devez définir les dépendances sur les composants AWS fournis qui regroupent ces

fonctionnalités. Lorsque vous déployez le composant de fonction Lambda, le déploiement inclut également le composant pour chaque fonctionnalité que vous définissez comme dépendance. Lors du déploiement, vous pouvez configurer des paramètres, tels que les secrets à déployer sur le périphérique principal. Toutes les fonctionnalités de la V1 ne nécessitent pas une dépendance de composant pour votre fonction Lambda sur la V2. La liste suivante décrit comment utiliser les fonctionnalités V1 dans votre composant de fonction Lambda V2 :

- Accédez à d'autres AWS services

Si votre fonction Lambda utilise des AWS informations d'identification pour envoyer des demandes à d'autres AWS services, le rôle d'échange de jetons du périphérique principal doit permettre au périphérique principal d'effectuer les AWS opérations utilisées par la fonction Lambda. Pour de plus amples informations, veuillez consulter [Autoriser les appareils principaux à interagir avec les AWS services](#).

- Gestionnaire de flux

Si votre fonction Lambda utilise le gestionnaire de flux, spécifiez-la `aws.greengrass.StreamManager` en tant que dépendance de composant lorsque vous importez la fonction. Lorsque vous déployez le composant du gestionnaire de flux, spécifiez les paramètres du gestionnaire de flux à définir pour les équipements principaux cibles. Le rôle d'échange de jetons du périphérique principal doit permettre à celui-ci d'accéder aux AWS Cloud destinations que vous utilisez avec le gestionnaire de flux. Pour de plus amples informations, veuillez consulter [Gestionnaire de flux](#).

- Secrets locaux

Si votre fonction Lambda utilise des secrets locaux, spécifiez-les `aws.greengrass.SecretManager` comme dépendance de composant lorsque vous importez la fonction. Lorsque vous déployez le composant secret manager, spécifiez les ressources secrètes à déployer sur les équipements principaux cibles. Le rôle d'échange de jetons du périphérique principal doit permettre au périphérique principal de récupérer les ressources secrètes à déployer. Pour de plus amples informations, veuillez consulter [Directeur des services secrets](#).

Lorsque vous déployez votre composant de fonction Lambda, configurez-le pour qu'il dispose d'une [politique d'autorisation IPC](#) qui autorise l'utilisation de l'[opération GetSecretValue IPC](#) dans la V2. Kit SDK des appareils AWS IoT

- Ombres locales

Si votre fonction Lambda interagit avec des ombres locales, vous devez mettre à jour le code de la fonction Lambda pour utiliser la V2. Kit SDK des appareils AWS IoT Vous devez également spécifier `aws.greengrass.ShadowManager` une dépendance de composant lorsque vous importez la fonction. Pour de plus amples informations, veuillez consulter [Interagissez avec les ombres de l'appareil](#).

Lorsque vous déployez votre composant de fonction Lambda, configurez-le pour qu'il dispose d'une [politique d'autorisation IPC](#) autorisant l'utilisation des [opérations IPC parallèles](#) dans la V2. Kit SDK des appareils AWS IoT

- Abonnements
 - Si votre fonction Lambda s'abonne à des messages provenant d'une source cloud, spécifiez ces abonnements comme sources d'événements lorsque vous importez la fonction.
 - Si votre fonction Lambda s'abonne aux messages d'une autre fonction Lambda, ou si votre fonction Lambda publie des messages vers AWS IoT Core d'autres fonctions Lambda, configurez et déployez l'ancien composant [routeur d'abonnement](#) lorsque vous déployez votre fonction Lambda. Lorsque vous déployez l'ancien composant routeur d'abonnement, spécifiez les abonnements utilisés par la fonction Lambda.

 Note

L'ancien composant routeur d'abonnement n'est requis que si votre fonction Lambda utilise la `publish()` fonction du SDK AWS IoT Greengrass Core. Si vous mettez à jour votre code de fonction Lambda pour utiliser l'interface de communication interprocessus (IPC) de la Kit SDK des appareils AWS IoT V2, vous n'avez pas besoin de déployer l'ancien composant routeur d'abonnement. Pour plus d'informations, consultez les services de [communication interprocessus](#) suivants :

- [Publier/souscrire des messages locaux](#)
 - [Publier/souscrire AWS IoT Core des messages MQTT](#)
- Si votre fonction Lambda s'abonne à des messages provenant d'appareils connectés locaux, spécifiez ces abonnements comme sources d'événements lorsque vous importez la fonction. Vous devez également configurer et déployer le [composant du pont MQTT](#) pour relayer les messages des appareils connectés vers les `publish/subscribe` sujets locaux que vous spécifiez comme sources d'événements.

- [Si votre fonction Lambda publie des messages sur des appareils connectés locaux, vous devez mettre à jour le code de la fonction Lambda pour utiliser la Kit SDK des appareils AWS IoT V2 pour publier des messages locaux. publish/subscribe](#) Vous devez également configurer et déployer le [composant du pont MQTT](#) pour relayer les messages du courtier de publish/subscribe messages local vers les appareils connectés.
- Volumes et appareils locaux

Si votre fonction Lambda conteneurisée accède à des volumes ou à des appareils locaux, spécifiez ces volumes et appareils lorsque vous importez la fonction Lambda. Cette fonctionnalité ne nécessite pas de dépendance à un composant.

Pour de plus amples informations, veuillez consulter [Exécuter AWS Lambda des fonctions](#).

Utiliser des connecteurs V1

Vous pouvez déployer des composants AWS fournis qui offrent les mêmes fonctionnalités que certains AWS IoT Greengrass connecteurs. Lorsque vous créez le déploiement, vous pouvez configurer les paramètres des connecteurs.

Les AWS IoT Greengrass V2 composants suivants fournissent les fonctionnalités du connecteur Greengrass V1 :

- [CloudWatch composant de métriques](#)
- [AWS IoT Device Defender composant](#)
- [Composant Firehose](#)
- [Composant adaptateur de protocole Modbus-RTU](#)
- [Composant Amazon SNS](#)

Exécuter des conteneurs Docker

AWS IoT Greengrass V2 ne fournit pas de composant pour remplacer directement le connecteur de déploiement d'applications Docker V1. Cependant, vous pouvez utiliser le composant du gestionnaire d'applications Docker pour télécharger des images Docker, puis créer des composants personnalisés qui exécutent des conteneurs Docker à partir des images téléchargées. Pour plus d'informations, consultez [Exécuter un conteneur Docker](#) et [Gestionnaire d'applications Docker](#).

Connectez les appareils Greengrass V1

Les appareils connectés en AWS IoT Greengrass V1 entrée sont appelés appareils clients en entrée AWS IoT Greengrass V2. AWS IoT Greengrass V2 la prise en charge des appareils clients est rétrocompatible avec AWS IoT Greengrass V1, de sorte que vous pouvez connecter les appareils clients V1 aux périphériques principaux V2 sans modifier leur code d'application. Pour permettre aux appareils clients de se connecter à un périphérique principal V2, déployez les composants Greengrass qui permettent la prise en charge des appareils clients et associez les appareils clients au périphérique principal. [Pour relayer des messages entre les appareils clients, le service AWS IoT Core cloud et les composants de Greengrass \(y compris les fonctions Lambda\), déployez et configurez le composant de pont MQTT.](#) Vous pouvez déployer le [composant de détection IP](#) pour détecter automatiquement les informations de connectivité, ou vous pouvez gérer manuellement les points de terminaison. Pour de plus amples informations, veuillez consulter [Interagissez avec les appareils IoT locaux.](#)

Activer le service parallèle local

Dans AWS IoT Greengrass V2, le service parallèle local est implémenté par le composant shadow manager AWS fourni. AWS IoT Greengrass V2 inclut également le support pour les ombres nommées. Pour permettre à vos composants d'interagir avec les ombres locales et de synchroniser les états des ombres avec celles-ci AWS IoT Core, configurez et déployez le composant Shadow Manager, et utilisez les opérations IPC des ombres dans le code de votre composant. Pour de plus amples informations, veuillez consulter [Interagissez avec les ombres de l'appareil.](#)

Intégrez avec AWS IoT SiteWise

Si vous utilisez votre périphérique principal V1 comme AWS IoT SiteWise passerelle, [suivez les instructions](#) pour configurer votre nouveau périphérique principal V2 en tant que AWS IoT SiteWise passerelle. AWS IoT SiteWise fournit un script d'installation qui déploie les AWS IoT SiteWise composants pour vous.

Étape 3 : Testez vos AWS IoT Greengrass V2 applications

Après avoir créé et déployé des composants V2 sur votre nouveau périphérique principal V2, vérifiez que vos applications répondent à vos attentes. Vous pouvez consulter les journaux du périphérique pour voir les messages de sortie standard (stdout) et d'erreur standard (stderr) de vos composants. Pour de plus amples informations, veuillez consulter [AWS IoT Greengrass Journaux de surveillance.](#)

Si vous avez déployé la [CLI Greengrass](#) sur le périphérique principal, vous pouvez l'utiliser pour déboguer les composants et leurs configurations. Pour de plus amples informations, veuillez consulter [Commandes Greengrass CLI](#).

Après avoir vérifié que vos applications fonctionnent sur un appareil principal V2, vous pouvez déployer les composants Greengrass de votre application sur d'autres appareils principaux. Si vous avez développé des composants personnalisés qui exécutent des processus natifs ou des conteneurs Docker, vous devez d'abord [publier ces composants](#) sur le AWS IoT Greengrass service pour les déployer sur d'autres appareils principaux.

Configurer un nouvel appareil pour tester les applications V1 sur V2 (Greengrass nucleus lite)

Configurez un nouvel appareil avec Greengrass nucleus lite pour tester les composants génériques que vous créez pour migrer vos AWS IoT Greengrass V1 applications vers la version V2. Greengrass Nucleus Lite est un environnement d'exécution léger optimisé pour les appareils aux ressources limitées. Vous pouvez utiliser cet appareil pour développer et tester des composants Greengrass personnalisés qui exécutent des processus natifs. Après avoir testé vos applications sur un appareil Greengrass nucleus lite, vous pouvez déployer les composants V2 sur d'autres appareils exécutant Greengrass nucleus lite ou mettre à niveau vos appareils principaux V1 existants vers la version V2.

Étape 1 : installer Greengrass nucleus lite sur un nouvel appareil

Installez le Greengrass Nucleus Lite sur un nouvel appareil. Suivez le [guide d'installation de Greengrass Nucleus Lite](#) pour configurer un appareil.

Note

Greengrass Nucleus Lite ne prend actuellement pas en charge le service parallèle local, les appareils clients ou les connecteurs. Assurez-vous que vos applications V1 ne s'appuient pas sur ces fonctionnalités avant de suivre ce guide.

Étape 2 : créer et déployer des composants génériques pour migrer les fonctions AWS IoT Greengrass V1 Lambda

Pour reproduire les fonctionnalités de vos fonctions AWS IoT Greengrass V1 Lambda sur Greengrass nucleus lite, vous devez les convertir en composants génériques. Cela implique de réécrire le

code de votre fonction Lambda pour utiliser Kit SDK des appareils AWS IoT le SDK V2 AWS IoT Greengrass ou Component au lieu du AWS IoT Greengrass SDK Core.

Le tableau suivant répertorie les exemples de composants SDKs utilisés dans la version V2 de ce guide :

Kit SDK	Version minimale
Kit SDK des appareils AWS IoT pour Python v2	v1.11.3
Kit SDK des appareils AWS IoT pour Java v2	v1.9.3
Kit SDK des appareils AWS IoT pour JavaScript v2	v1.12.0
AWS IoT Greengrass SDK pour les composants (C/C++) (actuellement en version préliminaire)	v0.4.0

Les exemples ci-dessous montrent les fonctions Lambda qui utilisent le SDK AWS IoT Greengrass V1 Core et ses composants génériques équivalents, avec du code de composant, des recettes et des instructions de construction dans plusieurs langages de programmation pour deux scénarios principaux :

- Communication locale : composants qui communiquent avec d'autres composants du même appareil à l'aide d'un pub/sub local
- Communication dans le cloud : composants qui communiquent avec d'autres services AWS IoT Core ou avec d'autres AWS services

Scénario 1 : communication locale (éditeur → processeur → abonné)

Ce scénario montre comment convertir une fonction Lambda V1 qui utilise la pub/sub communication locale en un composant générique V2.

Architecture d'application

Cet exemple implique trois composants :

- L'éditeur Lambda publie des données de température

- Le processeur Lambda reçoit les données et les traite
- Le processeur Lambda publie le résultat traité à l'abonné Lambda

L'exemple de code ci-dessous se concentre sur le processeur Lambda, qui montre à la fois l'abonnement et la publication de messages pour les communications locales.

Abonnements du groupe V1

Dans AWS IoT Greengrass V1, les abonnements de groupe suivants permettent la communication entre les fonctions Lambda :

Abonnement 1 : éditeur → processeur

- Source : Lambda (éditeur)
- Cible : Lambda (processeur)
- Sujet : capteurs/température

Abonnement 2 : processeur → abonné

- Source : Lambda (processeur)
- Cible : Lambda (abonné)
- Sujet : lambda/alertes

Fonction Lambda du processeur (V1)

Python

```
import greengrasssdk
import json

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

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

```

temperature = event['temperature']

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

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

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

    print(f"Alert sent to subscriber Lambda")

return {'statusCode': 200}

```

Java

```

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

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

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

    public String handleRequest(Map<String, Object> event, Context context) {
        /*

```

```

    * Receives temperature from publisher Lambda,
    * processes it, and forwards to subscriber Lambda
    */

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

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

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

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

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

        iotDataClient.publish(publishRequest);

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

    return "Success";
}
}

```

JavaScript

```

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

const iotClient = new greengrasssdk.IotData();

/**
 * Greengrass v1 Lambda function

```

```

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

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

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

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

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

```

C

```

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

```



```
#include <jansson.h> // For JSON parsing.

static aws_greengrass_iot_data_client *iot_client = NULL;

void on_message_received(const char *topic, const uint8_t *payload, size_t
payload_len, void *user_data) {
    // Parse the incoming message.
    char *payload_str = strdup((char *)payload, payload_len);
    json_error_t error;
    json_t *event = json_loads(payload_str, 0, &error);
    free(payload_str);

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

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

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

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

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

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

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

```

        aws_greengrass_iot_data_publish(iot_client, &params);

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

        free(alert_payload);
        json_decref(alert_data);
    }

    json_decref(event);
}

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

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

    aws_greengrass_iot_data_subscribe(iot_client, &subscribe_params);

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

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

    return 0;
}

```

C++

```

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

```

```

#include <unistd.h>

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

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

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

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

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

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

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

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

```

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

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

json_decref(event);
}

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

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

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

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

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

    free(alert_payload);
    json_decref(alert_data);
}

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

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

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

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

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

    return 0;
}
```

Composant générique (V2)

Pour obtenir les mêmes fonctionnalités dans AWS IoT Greengrass V2, créez un composant générique avec les éléments suivants :

1. Code du composant

Python

Conditions préalables : Avant d'utiliser ce code de composant, installez et vérifiez le code Kit SDK des appareils AWS IoT pour Python sur votre appareil Greengrass :

```
# Install the SDK
pip3 install awsiotsdk

# Verify installation
```

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

Si vous rencontrez des conflits de dépendance lors de l'installation, essayez d'installer une version spécifique du Kit SDK des appareils AWS IoT.

Si la commande de vérification affiche « SDK installé avec succès », vous êtes prêt à utiliser le code du composant :

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

ipc_client = GreengrassCoreIPCClientV2()

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

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

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

            # Publish to another component (AlertHandler).
            ipc_client.publish_to_topic(
                topic='component/alerts',
                publish_message=PublishMessage(
```

```

        json_message=JsonMessage(message=alert_data)
    )
)

print(f"Alert sent to AlertHandler component")

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

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

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

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

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

if __name__ == '__main__':
    main()

```

Java

```

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

public class TemperatureProcessor {
    private static GreengrassCoreIPCClientV2 ipcClient;

```

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

    try (GreengrassCoreIPCCliëntV2 client =
GreengrassCoreIPCCliëntV2.builder().build()) {
        ipcClient = client;

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

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

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

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

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

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

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



```

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

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

```

JavaScript

```

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

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

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

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

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

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

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

```

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

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

await streamingOperation.activate();

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

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

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

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

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

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

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

```

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

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

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

```

C

```

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

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

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

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

```

```

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

    while (true) {
        pthread_mutex_lock(&alert_mutex);
        while (!has_pending_alert) {
            pthread_cond_wait(&alert_cond, &alert_mutex);
        }

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

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

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

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

    return NULL;
}

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

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

```

```

GgMap map = gg_obj_into_map(payload);

GgObject *sensor_id_obj;
if (!gg_map_get(map, GG_STR("sensor_id"), &sensor_id_obj)) {
    fprintf(stderr, "Missing sensor_id field\n");
    return;
}
GgBuffer sensor_id = gg_obj_into_buf(*sensor_id_obj);

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

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

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

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

    gg_sdk_init();

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

```

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

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

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

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

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

return 0;
}
```

C++

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

#include <chrono>
#include <condition_variable>
#include <iostream>
#include <mutex>
#include <string>
```

```

#include <string_view>
#include <thread>

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

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

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

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

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

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

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

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

class SensorCallback : public gg::ipc::LocalTopicCallback {
    void operator()(

```

```

        std::string_view topic,
        gg::Object payload,
        gg::ipc::Subscription& handle
    ) override {
        (void) topic;
        (void) handle;

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

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

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

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

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

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

```



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

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

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

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

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

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

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

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

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

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

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

```
    return 0;
}
```

2. Compiler et emballer le composant

Certains langages nécessitent une création ou un emballage avant le déploiement.

Python

Python ne nécessite pas de compilation. Le composant peut utiliser directement le fichier .py.

Java

Pour créer un fichier JAR exécutable avec toutes les dépendances groupées :

1. Créez un pom.xml fichier dans le répertoire de votre projet :

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

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

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

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

```

        <version>1.25.1</version>
    </dependency>
</dependencies>

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

```

```
</build>
</project>
```

2. Construisez le fichier JAR :

```
mvn clean package
```

Cela crée `target/temperature-processor-1.0.0.jar` avec toutes les dépendances incluses.

3. Téléchargez le fichier JAR dans votre compartiment S3 pour le déploiement.

JavaScript

Pour emballer votre composant Node.js avec toutes les dépendances, procédez comme suit :

1. Créez un `package.json` fichier :

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

2. Installez les dépendances sur votre machine de développement :

```
npm install
```

Cela crée un `node_modules` dossier contenant la AWS Kit SDK des appareils AWS IoT v2.

3. Package pour le déploiement :

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

4. Téléchargez le fichier zip dans votre compartiment S3 pour le déploiement.

Note

Le moteur d'exécution Node.js doit être installé sur le périphérique Greengrass (version 14 ou ultérieure). Vous n'avez pas besoin de l'exécuter `npm install` sur le périphérique principal de Greengrass puisque l'artefact du composant inclut toutes les dépendances dans le dossier fourni. `node_modules`

C

Prérequis :

Pour créer le SDK et le composant, vous aurez besoin des dépendances de construction suivantes :

- GCC ou Clang
- CMake (au moins la version 3.22)
- Marque ou Ninja

Installez les dépendances de construction :

Sur Ubuntu/Debian :

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

Sur Amazon Linux :

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

Créez un fichier CMake Lists.txt pour votre composant :

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

set(CMAKE_C_STANDARD 11)

# Add AWS Greengrass Component SDK
```

```
add_subdirectory(aws-greengrass-component-sdk)

# Build your component executable
add_executable(temperature_processor temperature_processor.c)
target_link_libraries(temperature_processor gg-sdk)
```

Étapes de construction :

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

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

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

C++

Prérequis :

Pour créer le SDK et le composant, vous aurez besoin des dépendances de construction suivantes :

- GCC ou Clang avec support C++20
- CMake (au moins la version 3.22)
- Make ou Ninja

Installez les dépendances de construction :

Sur Ubuntu/Debian :

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

Sur Amazon Linux :

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

Créez un fichier CMake Lists.txt pour votre composant :

```

cmake_minimum_required(VERSION 3.10)
project(TemperatureProcessor CXX)

set(CMAKE_CXX_STANDARD 20)

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

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

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

```

Étapes de construction :

```

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

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

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

```

3. Recette de composants

Mettez à jour le tableau « ressources » avec les sujets réels utilisés par votre composant.

Python

```

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

```

```

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

```


Java

```

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

```

```

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

```

JavaScript

```

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

```

```

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

```

C/C++

```

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

```

```

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

```

Scénario 2 : communication dans le cloud

Ce scénario montre comment convertir une fonction Lambda V1 qui communique avec en un AWS IoT Core composant générique V2.

Architecture d'application

Cet exemple utilise une architecture connectée au cloud :

- AWS IoT Core envoie des commandes à l'appareil
- Le contrôleur Lambda reçoit les commandes et les traite
- Le contrôleur Lambda renvoie les données de télémétrie à AWS IoT Core

Cet exemple se concentre sur le contrôleur Lambda, qui montre à la fois la réception de messages et la publication de messages vers. AWS IoT Core

Abonnements du groupe V1

Dans AWS IoT Greengrass V1, les abonnements de groupe suivants permettent la communication entre la fonction Lambda et : AWS IoT Core

Abonnement 1 : IoT Cloud → Lambda

- Source : IoT Cloud
- Cible : Lambda () DeviceController
- Sujet : commandes/device1

Abonnement 2 : Lambda → IoT Cloud

- Source : Lambda () DeviceController
- Cible : IoT Cloud
- Sujet : télémétrie/appareil1

Fonction Lambda du contrôleur (V1)

Python

```
import greengrasssdk
import json
import time

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

```
def lambda_handler(event, context):
    """
    Receives commands from IoT Core,
    processes them, and sends telemetry back to cloud
    """

    # Receive command from IoT Core.
    command = event.get('command')
    device_id = event.get('device_id', 'device1')

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

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

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

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

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

    return {'statusCode': 200}

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

Java

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

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

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

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

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

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

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

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

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

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

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

            iotDataClient.publish(publishRequest);

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

```
    }

    return "Success";
}

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

JavaScript

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

const iotClient = new greengrasssdk.IotData();

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

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

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

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

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

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



```

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

```

C

```

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

static aws_greengrass_iot_data_client *iot_client = NULL;

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

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

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

```

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

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

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

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

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

    char *telemetry_payload = json_dumps(telemetry_data, JSON_COMPACT);

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

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

    aws_greengrass_iot_data_publish(iot_client, &params);

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

    free(telemetry_payload);
    json_decref(telemetry_data);
}

json_decref(event);
}
```

```

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

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

    aws_greengrass_iot_data_subscribe(iot_client, &subscribe_params);

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

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

    return 0;
}

```

C++

```

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

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

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

```

```

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

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

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

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

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

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

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

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

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

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

```

```

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

        char *telemetry_payload = json_dumps(telemetry_data, JSON_COMPACT);

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

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

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

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

        free(telemetry_payload);
        json_decref(telemetry_data);
    }

    json_decref(event);
}

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

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

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

```

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

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

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

    return 0;
}
```

Composant générique (V2)

Pour obtenir les mêmes fonctionnalités dans AWS IoT Greengrass V2, créez un composant générique avec les éléments suivants :

1. Code du composant

Python

Conditions préalables : Avant d'utiliser ce code de composant, installez et vérifiez le code Kit SDK des appareils AWS IoT pour Python sur votre appareil Greengrass :

```
# Install the SDK
pip3 install awsiotsdk

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

Si vous rencontrez des conflits de dépendance lors de l'installation, essayez d'installer une version spécifique du Kit SDK des appareils AWS IoT.

Si la commande de vérification affiche « SDK installé avec succès », vous êtes prêt à utiliser le code du composant :

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

ipc_client = GreengrassCoreIPCClientV2()

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

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

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

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

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

```

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

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

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

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

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

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

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

if __name__ == '__main__':
    main()

```

Java

```

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

public class DeviceController {

```



```
private static GreengrassCoreIPCClientV2 ipcClient;

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

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

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

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

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

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

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

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

```

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

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

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

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

            ipcClient.publishToIoTCore(publishRequest);

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

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

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

```

JavaScript

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

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

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

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

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

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

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

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

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

      await streamingOperation.activate();

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

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

```
    }  
  }  
  
  async onCommand(message) {  
    try {  
      const payload = message.message.payload.toString('utf-8');  
      const data = JSON.parse(payload);  
  
      const command = data.command;  
      const deviceId = data.device_id || 'device1';  
  
      console.log(`Received command from cloud: ${command}`);  
  
      if (command === 'get_status') {  
        const status = this.getDeviceStatus();  
  
        const telemetryData = {  
          device_id: deviceId,  
          status: status,  
          timestamp: Date.now() / 1000  
        };  
  
        const telemetryJson = JSON.stringify(telemetryData);  
  
        const publishRequest = {  
          topicName: `telemetry/${deviceId}`,  
          qos: 1,  
          payload: Buffer.from(telemetryJson, 'utf-8')  
        };  
  
        await this.ipcClient.publishToIoTCore(publishRequest);  
        console.log(`Telemetry sent to cloud: ${telemetryJson}`);  
      }  
  
    } catch (error) {  
      console.error('Error processing command:', error);  
    }  
  }  
  
  getDeviceStatus() {  
    return 'online';  
  }  
}
```

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

C

```
#include <gg/buffer.h>  
#include <gg/error.h>  
#include <gg/ipc/client.h>  
#include <gg/map.h>  
#include <gg/object.h>  
#include <gg/sdk.h>  
#include <unistd.h>  
#include <stdbool.h>  
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <time.h>  
#include <pthread.h>  
  
#define COMMAND_TOPIC "commands/device1"  
#define TELEMETRY_TOPIC "telemetry/device1"  
  
typedef struct {  
    char device_id[64];  
    char command[64];  
} CommandData;  
  
static pthread_mutex_t command_mutex = PTHREAD_MUTEX_INITIALIZER;  
static pthread_cond_t command_cond = PTHREAD_COND_INITIALIZER;  
static CommandData pending_command;  
static bool has_pending_command = false;  
  
const char* get_device_status(void) {  
    // Simulate getting device status.  
    return "online";  
}  
  
static void *telemetry_publisher_thread(void *arg) {  
    (void) arg;  
  
    while (true) {  
        pthread_mutex_lock(&command_mutex);
```

```

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

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

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

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

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

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

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

return NULL;
}

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

```

```

(void) handle;

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

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

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

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

if (command_start) {
    pthread_mutex_lock(&command_mutex);

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

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

```

```

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

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

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

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

    gg_sdk_init();

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

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

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

```



```

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

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

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

    while (true) {
        sleep(1);
    }

    return 0;
}

```

C++

```

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

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

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

struct CommandData {

```

```

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

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

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

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

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

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

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

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

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

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

            if (error) {

```

```

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

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

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

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

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

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

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

```

```

    }

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

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

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

};

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

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

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

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

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

    // Start telemetry publisher thread.
    std::thread telemetry_thread(telemetry_publisher_thread);
    telemetry_thread.detach();

```

```
// Subscribe to commands from IoT Core.
static CloudCommandCallback handler;
error = client.subscribe_to_iot_core(COMMAND_TOPIC, handler);

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

std::cout << "Subscribed to " << COMMAND_TOPIC << std::endl;
std::cout << "Waiting for commands from IoT Core..." << std::endl;

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

return 0;
}
```

2. Compiler et emballer le composant

Certains langages nécessitent une création ou un emballage avant le déploiement.

Python

Python ne nécessite pas de compilation. Le composant peut utiliser directement le fichier .py.

Java

Pour créer un fichier JAR exécutable avec toutes les dépendances groupées :

1. Créez un pom.xml fichier dans le répertoire de votre projet :

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

```

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

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

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

    <build>
        <plugins>
            <!-- Maven Shade Plugin - creates a standalone JAR with all
dependencies included for Greengrass deployment -->
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-shade-plugin</artifactId>
                <version>3.2.4</version>
                <executions>
                    <execution>
                        <phase>package</phase>
                        <goals>
                            <goal>shade</goal>
                        </goals>
                        <configuration>
                            <transformers>
                                <!-- Set the main class for the executable JAR
-->
                                <transformer
implementation="org.apache.maven.plugins.shade.resource.ManifestResourceTransformer">
                                    <mainClass>DeviceController</mainClass>

```

```

        </transformer>
      </transformers>
      <filters>
        <!-- Exclude signature files to avoid security
exceptions -->
        <filter>
          <artifact>*:*</artifact>
          <excludes>
            <exclude>META-INF/*.SF</exclude>
            <exclude>META-INF/*.DSA</exclude>
            <exclude>META-INF/*.RSA</exclude>
          </excludes>
        </filter>
      </filters>
    </configuration>
  </execution>
</executions>
</plugin>
</plugins>
</build>
</project>

```

2. Construisez le fichier JAR :

```
mvn clean package
```

Cela crée `target/device-controller-1.0.0.jar` avec toutes les dépendances incluses.

3. Téléchargez le fichier JAR dans votre compartiment S3 pour le déploiement.

JavaScript

Pour emballer votre composant Node.js avec toutes les dépendances, procédez comme suit :

1. Créez un `package.json` fichier :

```

{
  "name": "device-controller",
  "version": "1.0.0",
  "description": "Device controller component for Greengrass v2",
  "main": "DeviceController.js",
  "dependencies": {

```

```
"aws-iot-device-sdk-v2": "^1.21.0"
},
"engines": {
  "node": ">=14.0.0"
}
}
```

2. Installez les dépendances sur votre machine de développement :

```
npm install
```

Cela crée un `node_modules` dossier contenant la AWS Kit SDK des appareils AWS IoT v2.

3. Package pour le déploiement :

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

4. Téléchargez le fichier zip dans votre compartiment S3 pour le déploiement.

Note

Le moteur d'exécution Node.js doit être installé sur le périphérique Greengrass (version 14 ou ultérieure). Vous n'avez pas besoin de l'exécuter `npm install` sur le périphérique principal de Greengrass puisque l'artefact du composant inclut toutes les dépendances dans le dossier fourni. `node_modules`

C

Prérequis :

Pour créer le SDK et le composant, vous aurez besoin des dépendances de construction suivantes :

- GCC ou Clang
- CMake (au moins la version 3.22)
- Marque ou Ninja

Installez les dépendances de construction :

Sur Ubuntu/Debian :

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

Sur Amazon Linux :

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

Créez un fichier CMake Lists.txt pour votre composant :

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

set(CMAKE_C_STANDARD 11)

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

# Build your component executable
add_executable(device_controller device_controller.c)
target_link_libraries(device_controller gg-sdk)
```

Étapes de construction :

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

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

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

C++

Prérequis :

Pour créer le SDK et le composant, vous aurez besoin des dépendances de construction suivantes :

- GCC ou Clang avec support C++20

- CMake (au moins la version 3.22)
- Marque ou Ninja

Installez les dépendances de construction :

Sur Ubuntu/Debian :

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

Sur Amazon Linux :

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

Créez un fichier CMake Lists.txt pour votre composant :

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

set(CMAKE_CXX_STANDARD 20)

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

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

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

Étapes de construction :

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

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

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

3. Recette de composants

Mettez à jour le tableau « ressources » avec les sujets réels utilisés par votre composant.

Python

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.DeviceController",
  "ComponentVersion": "1.0.0",
  "ComponentType": "aws.greengrass.generic",
  "ComponentDescription": "Receives commands from IoT Core and sends telemetry back
to cloud",
  "ComponentPublisher": "[Your Company]",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.mqttproxy": {
          "com.example.DeviceController:mqttproxy:1": {
            "policyDescription": "Allows access to subscribe to IoT Core topics",
            "operations": [
              "aws.greengrass#SubscribeToIoTCore"
            ],
            "resources": [
              "commands/device1"
            ]
          },
          "com.example.DeviceController:mqttproxy:2": {
            "policyDescription": "Allows access to publish to IoT Core topics",
            "operations": [
              "aws.greengrass#PublishToIoTCore"
            ],
            "resources": [
              "telemetry/device1"
            ]
          }
        }
      }
    }
  },
  "ComponentDependencies": {
    "aws.greengrass.TokenExchangeService": {
      "VersionRequirement": ">=2.0.0",
      "DependencyType": "HARD"
    }
  }
}
```

```

    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux",
        "runtime": "*"
      },
      "Lifecycle": {
        "run": "python3 -u {artifacts:path}/device_controller.py"
      },
      "Artifacts": [
        {
          "Uri": "s3://YOUR-BUCKET/artifacts/com.example.DeviceController/1.0.0/
device_controller.py"
        }
      ]
    }
  ]
}

```

Java

```

{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.DeviceController",
  "ComponentVersion": "1.0.0",
  "ComponentType": "aws.greengrass.generic",
  "ComponentDescription": "Receives commands from IoT Core and sends telemetry back
to cloud",
  "ComponentPublisher": "[Your Company]",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.mqttproxy": {
          "com.example.DeviceController:mqttproxy:1": {
            "policyDescription": "Allows access to subscribe to IoT Core topics",
            "operations": [
              "aws.greengrass#SubscribeToIoTCore"
            ],
            "resources": [
              "commands/device1"
            ]
          }
        }
      }
    }
  }
}

```

```

    },
    "com.example.DeviceController:mqttproxy:2": {
      "policyDescription": "Allows access to publish to IoT Core topics",
      "operations": [
        "aws.greengrass#PublishToIoTCore"
      ],
      "resources": [
        "telemetry/device1"
      ]
    }
  }
},
"ComponentDependencies": {
  "aws.greengrass.TokenExchangeService": {
    "VersionRequirement": ">=2.0.0",
    "DependencyType": "HARD"
  }
},
"Manifests": [
  {
    "Platform": {
      "os": "linux",
      "runtime": "*"
    },
    "Lifecycle": {
      "run": "java -jar {artifacts:path}/DeviceController.jar"
    },
    "Artifacts": [
      {
        "Uri": "s3://YOUR-BUCKET/artifacts/com.example.DeviceController/1.0.0/DeviceController.jar"
      }
    ]
  }
]
}

```

JavaScript

```

{
  "RecipeFormatVersion": "2020-01-25",

```

```

"ComponentName": "com.example.DeviceController",
"ComponentVersion": "1.0.0",
"ComponentType": "aws.greengrass.generic",
"ComponentDescription": "Receives commands from IoT Core and sends telemetry back
to cloud",
"ComponentPublisher": "[Your Company]",
"ComponentConfiguration": {
  "DefaultConfiguration": {
    "accessControl": {
      "aws.greengrass.ipc.mqttproxy": {
        "com.example.DeviceController:mqttproxy:1": {
          "policyDescription": "Allows access to subscribe to command topics from
IoT Core",
          "operations": [
            "aws.greengrass#SubscribeToIoTCore"
          ],
          "resources": [
            "commands/device1"
          ]
        },
        "com.example.DeviceController:mqttproxy:2": {
          "policyDescription": "Allows access to publish telemetry to IoT Core",
          "operations": [
            "aws.greengrass#PublishToIoTCore"
          ],
          "resources": [
            "telemetry/*"
          ]
        }
      }
    }
  }
},
"Manifests": [
  {
    "Platform": {
      "os": "linux",
      "runtime": "*"
    },
    "Lifecycle": {
      "run": "cd {artifacts:decompressedPath}/DeviceController && node
DeviceController.js"
    },
    "Artifacts": [

```

```

    {
      "Uri": "s3://YOUR-BUCKET/artifacts/com.example.DeviceController/1.0.0/DeviceController.zip",
      "Unarchive": "ZIP"
    }
  ]
}
]
}

```

C/C++

```

{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.DeviceController",
  "ComponentVersion": "1.0.0",
  "ComponentType": "aws.greengrass.generic",
  "ComponentDescription": "Receives commands from IoT Core and sends telemetry back to cloud",
  "ComponentPublisher": "[Your Company]",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.mqttproxy": {
          "com.example.DeviceController:mqttproxy:1": {
            "policyDescription": "Allows access to subscribe to IoT Core topics",
            "operations": ["aws.greengrass#SubscribeToIoTCore"],
            "resources": ["commands/device1"]
          },
          "com.example.DeviceController:mqttproxy:2": {
            "policyDescription": "Allows access to publish to IoT Core topics",
            "operations": ["aws.greengrass#PublishToIoTCore"],
            "resources": ["telemetry/device1"]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux",
        "runtime": ""
      }
    }
  ]
}

```

```
    },
    "Lifecycle": {
      "run": "{artifacts:path}/device_controller"
    },
    "Artifacts": [
      {
        "Uri": "s3://YOUR-BUCKET/artifacts/com.example.DeviceController/1.0.0/
device_controller"
      }
    ]
  }
}
```

Mettez à niveau les appareils principaux de Greengrass V1 vers Greengrass V2

Après avoir vérifié que vos applications et composants fonctionnent sur un appareil AWS IoT Greengrass V2 principal, vous pouvez installer le logiciel AWS IoT Greengrass Core v2.x sur les appareils qui exécutent actuellement la version v1.x, tels que les appareils de production. Déployez ensuite les composants Greengrass V2 pour exécuter vos applications Greengrass sur les appareils.

Pour mettre à niveau un parc d'appareils de la V1 à la V2, procédez comme suit pour chaque appareil à mettre à niveau. Vous pouvez utiliser des groupes d'objets pour déployer des composants V2 sur un parc d'appareils principaux.

Tip

Nous vous recommandons de créer un script pour automatiser le processus de mise à niveau d'un parc d'appareils. Si vous gérez votre flotte, vous pouvez utiliser Systems Manager pour exécuter ce script sur chaque appareil afin de faire passer votre flotte de la V1 à la V2.

[AWS Systems Manager](#)

Vous pouvez contacter votre représentant du Support aux AWS entreprises si vous avez des questions sur la meilleure façon d'automatiser le processus de mise à niveau.

Étape 1 : Installation du logiciel AWS IoT Greengrass Core v2.x

Choisissez l'une des options suivantes pour installer le logiciel AWS IoT Greengrass Core v2.x sur un périphérique principal V1 :

- [Mise à niveau en moins d'étapes](#)

Pour effectuer la mise à niveau en moins d'étapes, vous pouvez désinstaller le logiciel v1.x avant d'installer le logiciel v2.x.

- [Mise à niveau avec un temps d'arrêt minimal](#)

Pour effectuer la mise à niveau avec un temps d'arrêt minimal, vous pouvez installer les deux versions du logiciel AWS IoT Greengrass Core en même temps. Après avoir installé le logiciel AWS IoT Greengrass Core v2.x et vérifié que vos applications V2 fonctionnent correctement, vous désinstallez le logiciel AWS IoT Greengrass Core v1.x. Avant de choisir cette option, considérez la quantité de RAM supplémentaire requise pour exécuter les deux versions du logiciel AWS IoT Greengrass Core en même temps.

Désinstallez AWS IoT Greengrass Core v1.x avant d'installer la v2.x

Si vous souhaitez effectuer une mise à niveau séquentielle, désinstallez le logiciel AWS IoT Greengrass Core v1.x avant d'installer la version 2.x sur votre appareil.

Pour désinstaller le logiciel AWS IoT Greengrass Core v1.x

1. Si le logiciel AWS IoT Greengrass Core v1.x est exécuté en tant que service, vous devez arrêter, désactiver et supprimer le service.

- a. Arrêtez le service AWS IoT Greengrass Core Software v1.x en cours d'exécution.

```
sudo systemctl stop greengrass
```

- b. Attendez que le service s'arrête. Vous pouvez utiliser la `list` commande pour vérifier l'état du service.

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

- c. Désactivez le service.

```
sudo systemctl disable greengrass
```

- d. Supprimez le service.

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

2. Si le logiciel AWS IoT Greengrass Core v1.x n'est pas exécuté en tant que service, utilisez la commande suivante pour arrêter le démon. *greengrass-root* Remplacez-le par le nom de votre dossier racine Greengrass. L'emplacement par défaut est /greengrass.

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

3. (Facultatif) Sauvegardez votre dossier racine Greengrass et, le cas échéant, votre [dossier d'écriture personnalisé, dans](#) un autre dossier de votre appareil.

- a. Utilisez la commande suivante pour copier le dossier racine de Greengrass actuel dans un autre dossier, puis supprimez le dossier racine.

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

- b. Utilisez la commande suivante pour déplacer le dossier d'écriture vers un autre dossier, puis supprimez le dossier d'écriture.

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

Pour le noyau Greengrass : vous pouvez utiliser les [instructions d'installation pour AWS IoT Greengrass V2](#) installer le noyau Greengrass sur votre appareil.

Pour Greengrass nucleus lite : vous pouvez utiliser les [instructions d'installation de Greengrass nucleus lite pour installer Greengrass nucleus lite](#).

Tip

Pour réutiliser l'identité d'un appareil principal lorsque vous le migrez de la V1 à la V2, suivez les instructions d'[installation du logiciel AWS IoT Greengrass Core avec un provisionnement manuel](#). Supprimez d'abord le logiciel principal V1 de l'appareil, puis réutilisez l'élément

principal et le certificat du AWS IoT périphérique V1, et mettez à jour les AWS IoT politiques du certificat pour accorder les autorisations requises par le logiciel v2.x.

Installez le logiciel AWS IoT Greengrass Core v2.x sur un appareil exécutant déjà la version v1.x

Si vous installez le logiciel AWS IoT Greengrass Core v2.x sur un appareil qui exécute déjà le logiciel AWS IoT Greengrass Core v1.x, gardez à l'esprit les points suivants :

- Le nom de l' AWS IoT objet de votre appareil principal V2 doit être unique. N'utilisez pas le même nom que votre appareil principal V1.
- Les ports que vous utilisez pour le logiciel AWS IoT Greengrass Core v2.x doivent être différents de ceux que vous utilisez pour la v1.x.
 - Configurez le gestionnaire de flux V1 pour utiliser un port autre que le port 8088. Pour plus d'informations, consultez [Configurer le gestionnaire de flux](#).
 - Configurez le broker MQTT V1 pour utiliser un port autre que 8883. Pour plus d'informations, voir [Configuration du port MQTT pour la messagerie locale](#).
- AWS IoT Greengrass V2 ne permet pas de renommer le service système Greengrass. Si vous exécutez Greengrass en tant que service système, vous devez effectuer l'une des opérations suivantes pour éviter tout conflit entre les noms de service système :
 - Renommez le service Greengrass pour la version v1.x avant d'installer la version 2.x.
 - Installez le logiciel AWS IoT Greengrass Core v2.x sans service système, puis [configurez manuellement le logiciel en tant que service système](#) avec un nom autre que. greengrass

Pour renommer le service Greengrass pour la version v1.x

1. Arrêtez le service AWS IoT Greengrass Core software v1.x.

```
sudo systemctl stop greengrass
```

2. Attendez que le service s'arrête. L'arrêt du service peut prendre jusqu'à quelques minutes. Vous pouvez utiliser la `list-units` commande pour vérifier si le service s'est arrêté.

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

3. Désactivez le service.

```
sudo systemctl disable greengrass
```

4. Renommez le service.

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

5. Rechargez le service et démarrez-le.

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

Vous pouvez ensuite utiliser les [instructions d'installation AWS IoT Greengrass V2 pour](#) installer le logiciel sur votre appareil.

 Tip

Pour réutiliser l'identité d'un appareil principal lorsque vous le migrez de la V1 à la V2, suivez les instructions d'[installation du logiciel AWS IoT Greengrass Core avec un provisionnement manuel](#). Supprimez d'abord le logiciel principal V1 de l'appareil, puis réutilisez l'élément principal et le certificat du AWS IoT périphérique V1, et mettez à jour les AWS IoT politiques du certificat pour accorder les autorisations requises par le logiciel v2.x.

Étape 2 : Déployer AWS IoT Greengrass V2 des composants sur les appareils principaux

Après avoir installé le logiciel AWS IoT Greengrass Core v2.x sur votre appareil, déployez les composants en fonction du runtime que vous avez choisi.

Pour Greengrass nucleus :

Créez un déploiement qui inclut les ressources suivantes. Pour déployer des composants sur un parc d'appareils similaires, créez un déploiement pour un groupe d'objets contenant ces appareils.

- Composants de fonctions Lambda que vous avez créés à partir de vos fonctions Lambda V1. Pour de plus amples informations, veuillez consulter [Exécuter AWS Lambda des fonctions](#).
- Si vous utilisez des abonnements V1, l'[ancien composant routeur d'abonnement](#).
- Si vous utilisez le gestionnaire de flux, le [composant du gestionnaire de flux](#). Pour de plus amples informations, veuillez consulter [Gérez les flux de données sur les appareils principaux de Greengrass](#).
- Si vous utilisez des secrets locaux, le [composant du gestionnaire de secrets](#).
- Si vous utilisez des connecteurs V1, les [composants du connecteur AWS fournis](#).
- Si vous utilisez des conteneurs Docker, le composant du [gestionnaire d'applications Docker](#). Pour de plus amples informations, veuillez consulter [Exécuter un conteneur Docker](#).
- Si vous utilisez des appareils connectés, les [composants de l'appareil client sont pris en charge](#). Vous devez également activer la prise en charge des appareils clients et associer les appareils clients à votre appareil principal. Pour de plus amples informations, veuillez consulter [Interagissez avec les appareils IoT locaux](#).
- Si vous utilisez les ombres de l'appareil, le [composant Shadow Manager](#). Pour de plus amples informations, veuillez consulter [Interagissez avec les ombres de l'appareil](#).
- Si vous téléchargez des journaux depuis les appareils principaux de Greengrass vers Amazon CloudWatch Logs, le composant du [gestionnaire](#) de journaux. Pour de plus amples informations, veuillez consulter [AWS IoT Greengrass Journaux de surveillance](#).
- Si vous effectuez l'intégration avec AWS IoT SiteWise, [suivez les instructions](#) pour configurer le périphérique principal V2 en tant que AWS IoT SiteWise passerelle. AWS IoT SiteWise fournit un script d'installation qui déploie les AWS IoT SiteWise composants pour vous.
- Composants définis par l'utilisateur que vous avez développés pour implémenter des fonctionnalités personnalisées.

Pour plus d'informations sur la création et la révision de déploiements, consultez. [Déployer AWS IoT Greengrass des composants sur des appareils](#)

Pour Greengrass Nucleus Lite :

Déployez les composants génériques que vous avez créés à l'[étape 2 du guide de migration sur](#) vos appareils Greengrass Nucleus Lite :

1. Créez vos composants en utilisant les recettes de composants que vous avez créées

2. Créez un déploiement ciblant vos appareils Greengrass Nucleus Lite qui inclut vos composants génériques
3. Vérifiez que vos composants fonctionnent correctement

Tutoriel : Débuter avec AWS IoT Greengrass V2

Vous pouvez suivre ce didacticiel de mise en route pour découvrir les fonctionnalités de base de AWS IoT Greengrass V2. Dans ce didacticiel, vous allez effectuer les opérations suivantes :

1. Installez et configurez le logiciel AWS IoT Greengrass Core sur un appareil Linux, tel qu'un Raspberry Pi ou un appareil Windows. Cet appareil est un appareil de base de Greengrass.
2. Développez un composant Hello World sur votre appareil principal Greengrass. Les composants sont des modules logiciels qui s'exécutent sur les appareils principaux de Greengrass.
3. Téléchargez ce composant AWS IoT Greengrass V2 dans le AWS Cloud.
4. Déployez ce composant depuis votre appareil principal Greengrass. AWS Cloud

Note

Ce didacticiel explique comment configurer un environnement de développement et explore les fonctionnalités de AWS IoT Greengrass. Pour plus d'informations sur l'installation et la configuration des appareils de production, consultez les rubriques suivantes :

- [Configuration des appareils AWS IoT Greengrass principaux](#)
- [Installation du logiciel AWS IoT Greengrass de base](#)

Vous pouvez vous attendre à consacrer 20 à 30 minutes à ce didacticiel.

Rubriques

- [Prérequis](#)
- [Étape 1 : créer un AWS compte](#)
- [Étape 2 : Configuration de votre environnement](#)
- [Étape 3 : Installation du logiciel AWS IoT Greengrass principal](#)
- [Étape 4 : développer et tester un composant sur votre appareil](#)
- [Étape 5 : Créez votre composant dans le AWS IoT Greengrass service](#)
- [Étape 6 : Déployez votre composant](#)
- [Étapes suivantes](#)

Prérequis

Pour suivre ce didacticiel de démarrage, vous avez besoin des éléments suivants :

- Un Compte AWS. Si vous n'en avez pas, veuillez consulter [Étape 1 : créer un AWS compte](#).
- L'utilisation d'un ventilateur [Région AWS](#) qui supporte AWS IoT Greengrass V2. Pour obtenir la liste des régions prises en charge, consultez [Points de terminaison et quotas AWS IoT Greengrass V2](#) dans le Références générales AWS.
- Un utilisateur Gestion des identités et des accès AWS (IAM) doté d'autorisations d'administrateur.
- Un appareil à configurer en tant qu'appareil principal de Greengrass, tel qu'un Raspberry Pi avec le système d'[exploitation Raspberry Pi](#) (précédemment appelé Raspbian) ou un appareil Windows 10. Vous devez disposer d'autorisations d'administrateur sur cet appareil ou être en mesure d'acquérir des privilèges d'administrateur, par exemple via sudo. Cet appareil doit disposer d'une connexion Internet.

Vous pouvez également choisir d'utiliser un autre appareil répondant aux exigences d'installation et d'exécution du logiciel AWS IoT Greengrass Core.

Si votre ordinateur de développement répond à ces exigences, vous pouvez le configurer comme périphérique principal de Greengrass dans ce didacticiel.

- [Python](#) 3.5 ou version ultérieure installé pour tous les utilisateurs de l'appareil et ajouté à la variable d'PATH environnement. Sous Windows, le lanceur Python pour Windows doit également être installé pour tous les utilisateurs.

Important

Sous Windows, Python ne s'installe pas pour tous les utilisateurs par défaut. Lorsque vous installez Python, vous devez personnaliser l'installation pour la configurer afin que le logiciel AWS IoT Greengrass Core exécute des scripts Python. Par exemple, si vous utilisez le programme d'installation graphique Python, procédez comme suit :

1. Sélectionnez Installer le lanceur pour tous les utilisateurs (recommandé).
2. Choisissez Customize installation.
3. Choisissez Next.
4. Sélectionnez Install for all users.
5. Sélectionnez Add Python to environment variables.

6. Choisissez Installer.

Pour plus d'informations, consultez la section [Utilisation de Python sous Windows](#) dans la documentation de Python 3.

- AWS Command Line Interface (AWS CLI) installé et configuré avec des informations d'identification sur votre ordinateur de développement et sur votre appareil. Assurez-vous de l'utiliser Région AWS pour configurer le AWS CLI sur votre ordinateur de développement et sur votre appareil. Pour l'utiliser AWS IoT Greengrass V2 avec le AWS CLI, vous devez disposer de l'une des versions suivantes ou d'une version ultérieure :
 - Version AWS CLI V1 minimale : v1.18.197
 - Version AWS CLI V2 minimale : v2.1.11

Tip

Vous pouvez exécuter la commande suivante pour vérifier la version AWS CLI dont vous disposez.

```
aws --version
```

Pour plus d'informations, consultez les [sections Installation, mise à jour et désinstallation du AWS CLI](#) et [Configuration du AWS CLI dans le](#) guide de l'AWS Command Line Interface utilisateur.

Note

Si vous utilisez un appareil ARM 32 bits, tel qu'un Raspberry Pi avec un système d'exploitation 32 bits, installez AWS CLI la version 1. AWS CLI La V2 n'est pas disponible pour les appareils ARM 32 bits. Pour plus d'informations, voir [Installation, mise à jour et désinstallation de la AWS CLI version 1](#).

Étape 1 : créer un AWS compte

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 l'<https://portal.aws.amazon.com/billing/inscription>.
2. Suivez les instructions en ligne.

Dans le cadre de la procédure d'inscription, vous recevrez un appel téléphonique ou un SMS et vous saisissez 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 AWS est créé. Par défaut, seul l'utilisateur racine a accès à l'ensemble des Services AWS et des ressources de ce compte. La meilleure pratique de sécurité consiste à attribuer un accès administratif à un utilisateur, et à utiliser uniquement l'utilisateur racine pour effectuer les [tâches nécessitant un accès utilisateur racine](#).

AWS vous envoie un e-mail de confirmation une fois le processus d'inscription terminé. À tout moment, vous pouvez consulter l'activité actuelle de votre compte et gérer votre compte 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 Console](#) tant 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, octroyez 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.

Connexion en tant qu'utilisateur doté 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.

Attribution d'un accès à d'autres utilisateurs

1. Dans IAM Identity Center, créez un ensemble d'autorisations qui respecte la bonne pratique consistant à appliquer les autorisations de moindre privilège.

Pour obtenir des instructions, consultez [Création d'un ensemble d'autorisations](#) dans le Guide de l'utilisateur AWS IAM Identity Center .

2. Attribuez des utilisateurs à un groupe, puis attribuez un accès par authentification unique au groupe.

Pour obtenir des instructions, consultez [Ajout de groupes](#) dans le Guide de l'utilisateur AWS IAM Identity Center .

Étape 2 : Configuration de votre environnement

Note

Ces étapes ne s'appliquent pas à Nucleus Lite.

Suivez les étapes décrites dans cette section pour configurer un appareil Linux ou Windows à utiliser comme périphérique AWS IoT Greengrass principal.

Configuration d'un appareil Linux (Raspberry Pi)

Ces étapes supposent que vous utilisez un Raspberry Pi avec le système d'exploitation Raspberry Pi. Si vous utilisez un autre appareil ou un autre système d'exploitation, consultez la documentation correspondante à votre appareil.

Pour configurer un Raspberry Pi pour AWS IoT Greengrass V2

1. Activez SSH sur votre Raspberry Pi pour vous y connecter à distance. Pour plus d'informations, consultez [SSH \(Secure Shell\)](#) dans la documentation du Raspberry Pi.
2. Trouvez l'adresse IP de votre Raspberry Pi pour vous y connecter via SSH. Pour cela, vous pouvez exécuter la commande suivante sur votre Raspberry Pi.

```
hostname -I
```

3. Connectez-vous à votre Raspberry Pi via SSH.

Sur votre ordinateur de développement, exécutez la commande suivante. Remplacez *username* par le nom de l'utilisateur auquel vous souhaitez vous connecter et *pi-ip-address* par l'adresse IP que vous avez trouvée à l'étape précédente.

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

Important

Si votre ordinateur de développement utilise une version antérieure de Windows, il se peut que vous ne disposiez pas de cette ssh commande ou que vous ssh ne puissiez pas vous connecter à votre Raspberry Pi. Pour vous connecter à votre Raspberry Pi,

vous pouvez installer et configurer [PuTTY](#), un client SSH open source gratuit. Consultez la [documentation PuTTY](#) pour vous connecter à votre Raspberry Pi.

4. Installez le moteur d'exécution Java, dont le logiciel AWS IoT Greengrass Core a besoin pour fonctionner. Sur votre Raspberry Pi, utilisez les commandes suivantes pour installer Java 11.

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

Lorsque l'installation est terminée, exécutez la commande suivante pour vérifier que Java fonctionne sur votre Raspberry Pi.

```
java -version
```

La commande affiche la version de Java exécutée sur le périphérique. Le résultat peut ressembler à celui de l'exemple suivant.

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

Conseil : définissez les paramètres du noyau sur un Raspberry Pi

Si votre appareil est un Raspberry Pi, vous pouvez suivre les étapes suivantes pour afficher et mettre à jour les paramètres de son noyau Linux :

1. Ouvrez le fichier `/boot/cmdline.txt`. Ce fichier indique les paramètres du noyau Linux à appliquer au démarrage du Raspberry Pi.

Par exemple, sur un système basé sur Linux, vous pouvez exécuter la commande suivante pour utiliser GNU nano pour ouvrir le fichier.

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

2. Vérifiez que le `/boot/cmdline.txt` fichier contient les paramètres de noyau suivants. Le `systemd.unified_cgroup_hierarchy=0` paramètre indique d'utiliser cgroups v1 au lieu de cgroups v2.

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

Si le `/boot/cmdline.txt` fichier ne contient pas ces paramètres ou s'il contient des valeurs différentes, mettez-le à jour pour qu'il contienne ces paramètres et valeurs.

3. Si vous avez mis à jour le `/boot/cmdline.txt` fichier, redémarrez le Raspberry Pi pour appliquer les modifications.

```
sudo reboot
```

Configuration d'un appareil Linux (autre)

Pour configurer un appareil Linux pour AWS IoT Greengrass V2

1. Installez le moteur d'exécution Java, dont le logiciel AWS IoT Greengrass Core a besoin pour fonctionner. Nous vous recommandons d'utiliser les versions de support à long terme d'[Amazon Corretto](#) ou d'OpenJDK. La version 8 ou supérieure est requise. Les commandes suivantes vous montrent comment installer OpenJDK sur votre appareil.

- Pour les distributions basées sur Debian ou Ubuntu :

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

- Pour les distributions basées sur Red Hat :

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

- Dans Amazon Linux 2 :

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

- Pour Amazon Linux 2023 :

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

Lorsque l'installation est terminée, exécutez la commande suivante pour vérifier que Java s'exécute sur votre appareil Linux.

```
java -version
```

La commande affiche la version de Java exécutée sur le périphérique. Par exemple, sur une distribution basée sur Debian, le résultat peut ressembler à celui de l'exemple suivant.

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

2. (Facultatif) Créez l'utilisateur système et le groupe par défaut qui exécutent les composants sur le périphérique. Vous pouvez également choisir de laisser le programme d'installation du logiciel AWS IoT Greengrass Core créer cet utilisateur et ce groupe lors de l'installation avec l'argument `--component-default-user` `install`. Pour de plus amples informations, veuillez consulter [Arguments d'installation](#).

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

3. Vérifiez que l'utilisateur qui exécute le logiciel AWS IoT Greengrass Core (généralement `root`) est autorisé à exécuter `sudo` avec n'importe quel utilisateur et n'importe quel groupe.
 - a. Exécutez la commande suivante pour ouvrir le `/etc/sudoers` fichier.

```
sudo visudo
```

- b. Vérifiez que l'autorisation accordée à l'utilisateur ressemble à l'exemple suivant.

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

4. (Facultatif) Pour [exécuter des fonctions Lambda conteneurisées](#), vous devez activer [cgroups](#) v1, et vous devez activer et monter les cgroups de mémoire et de périphériques. Si vous ne prévoyez pas d'exécuter des fonctions Lambda conteneurisées, vous pouvez ignorer cette étape.

Pour activer ces options cgroups, démarrez le périphérique avec les paramètres du noyau Linux suivants.

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

Pour plus d'informations sur l'affichage et la définition des paramètres du noyau de votre appareil, consultez la documentation de votre système d'exploitation et de votre chargeur de démarrage. Suivez les instructions pour définir définitivement les paramètres du noyau.

5. Installez toutes les autres dépendances requises sur votre appareil, comme indiqué dans la liste des exigences figurant dans [Exigences relatives aux dispositifs](#).

Configuration d'un appareil Windows

Pour configurer un appareil Windows pour AWS IoT Greengrass V2

1. Installez le moteur d'exécution Java, dont le logiciel AWS IoT Greengrass Core a besoin pour fonctionner. Nous vous recommandons d'utiliser les versions de support à long terme d'[Amazon Corretto](#) ou d'OpenJDK. La version 8 ou supérieure est requise.
2. Vérifiez si Java est disponible sur la variable système [PATH](#), et ajoutez-le dans le cas contraire. Le LocalSystem compte exécute le logiciel AWS IoT Greengrass Core. Vous devez donc ajouter Java à la variable système PATH au lieu de la variable utilisateur PATH pour votre utilisateur. Procédez comme suit :
 - a. Appuyez sur la touche Windows pour ouvrir le menu de démarrage.
 - b. Tapez **environment variables** pour rechercher les options du système dans le menu Démarrer.
 - c. Dans les résultats de recherche du menu Démarrer, choisissez Modifier les variables d'environnement du système pour ouvrir la fenêtre des propriétés du système.
 - d. Choisissez les variables d'environnement... pour ouvrir la fenêtre des variables d'environnement.
 - e. Sous Variables système, sélectionnez Chemin, puis Modifier. Dans la fenêtre Modifier la variable d'environnement, vous pouvez afficher chaque chemin sur une ligne distincte.
 - f. Vérifiez si le chemin d'accès au bin dossier d'installation de Java est présent. Le chemin peut ressembler à celui de l'exemple suivant.

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

- g. Si le bin dossier de l'installation Java est absent de Path, choisissez Nouveau pour l'ajouter, puis OK.
3. Ouvrez l'invite de commande Windows (cmd.exe) en tant qu'administrateur.

4. Créez l'utilisateur par défaut dans le LocalSystem compte sur l'appareil Windows.

password Remplacez-le par un mot de passe sécurisé.

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

 Tip

En fonction de votre configuration Windows, le mot de passe de l'utilisateur peut être configuré pour expirer dans le futur. Pour vous assurer que vos applications Greengrass continuent de fonctionner, suivez la date d'expiration du mot de passe et mettez-le à jour avant son expiration. Vous pouvez également définir le mot de passe de l'utilisateur pour qu'il n'expire jamais.

- Pour vérifier la date d'expiration d'un utilisateur et de son mot de passe, exécutez la commande suivante.

```
net user ggc_user | findstr /C:expires
```

- Pour définir le mot de passe d'un utilisateur afin qu'il n'expire jamais, exécutez la commande suivante.

```
wmic UserAccount where "Name='ggc_user'" set PasswordExpires=False
```

- Si vous utilisez Windows 10 ou une version ultérieure où la [wmic commande est obsolète](#), exécutez la commande suivante PowerShell .

```
Get-CimInstance -Query "SELECT * from Win32_UserAccount WHERE name = 'ggc_user'" | Set-CimInstance -Property @{PasswordExpires="False"}
```

5. Téléchargez et installez l'[PsExecutilitaire](#) de Microsoft sur l'appareil.
6. Utilisez l' PsExec utilitaire pour stocker le nom d'utilisateur et le mot de passe de l'utilisateur par défaut dans l'instance Credential Manager du LocalSystem compte. *password* Remplacez-le par le mot de passe utilisateur que vous avez défini précédemment.

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

Si l'icône PsExec License Agreement ouvre, choisissez Accept pour accepter la licence et exécuter la commande.

 Note

Sur les appareils Windows, le LocalSystem compte exécute le noyau Greengrass, et vous devez utiliser l' PsExec utilitaire pour stocker les informations utilisateur par défaut dans le LocalSystem compte. L'application Credential Manager stocke ces informations dans le compte Windows de l'utilisateur actuellement connecté, plutôt que dans le LocalSystem compte.

Étape 3 : Installation du logiciel AWS IoT Greengrass principal

 Tip

Nous vous recommandons d'essayer le [pack contextuel AWS IoT Greengrass AI Agents](#) pour configurer et expérimenter rapidement IoT AWS Greengrass. Le pack de contexte des agents permettra aux agents d'intelligence artificielle de configurer Greengrass Nucleus et Nucleus Lite, de déployer des composants et de résoudre les problèmes courants.

Suivez les étapes décrites dans cette section pour configurer votre Raspberry Pi en tant que périphérique AWS IoT Greengrass principal que vous pouvez utiliser pour le développement local. Dans cette section, vous allez télécharger et exécuter un programme d'installation qui effectue les opérations suivantes pour configurer le logiciel AWS IoT Greengrass Core pour votre appareil :

- Installe le composant Greengrass nucleus. Le noyau est un composant obligatoire et constitue le minimum requis pour exécuter le logiciel AWS IoT Greengrass Core sur un appareil. Pour plus d'informations, consultez la section Composant du [noyau Greengrass](#).
- Enregistre votre appareil en tant qu' AWS IoT objet et télécharge un certificat numérique qui permet à votre appareil de se connecter à AWS. Pour de plus amples informations, veuillez consulter [Authentification et autorisation de l'appareil pour AWS IoT Greengrass](#).
- Ajoute l' AWS IoT objet de l'appareil à un groupe d'objets, qui est un groupe ou un parc d' AWS IoT objets. Les groupes d'objets vous permettent de gérer des flottes d'appareils principaux de Greengrass. Lorsque vous déployez des composants logiciels sur vos appareils, vous pouvez choisir de les déployer sur des appareils individuels ou sur des groupes d'appareils. Pour plus d'informations, consultez [la section Gestion des appareils AWS IoT](#) dans le Guide du AWS IoT Core développeur.

- Crée le rôle IAM qui permet à votre appareil principal Greengrass d'interagir avec AWS les services. Par défaut, ce rôle permet à votre appareil d'interagir avec Amazon Logs AWS IoT et de lui envoyer CloudWatch des journaux. Pour de plus amples informations, veuillez consulter [Autoriser les appareils principaux à interagir avec les AWS services](#).
- Installe l'interface de ligne de AWS IoT Greengrass commande (`greengrass-cli`), que vous pouvez utiliser pour tester les composants personnalisés que vous développez sur le périphérique principal. Pour de plus amples informations, veuillez consulter [Interface de ligne de commande Greengrass](#).

Installation du logiciel AWS IoT Greengrass Core (console)

1. Connectez-vous à la [console AWS IoT Greengrass](#).
2. Sous Commencer avec Greengrass, choisissez Configurer l'appareil principal.
3. Dans Étape 1 : Enregistrer un appareil Greengrass Core, dans le champ Nom de l'appareil Core, entrez le nom de l' AWS IoT appareil Greengrass Core. Si l'objet n'existe pas, le programme d'installation le crée.
4. Dans Étape 2 : Ajouter à un groupe d'objets pour appliquer un déploiement continu, pour le groupe d'objets, choisissez le AWS IoT groupe d'objets auquel vous souhaitez ajouter votre appareil principal.
 - Si vous sélectionnez Entrez un nouveau nom de groupe, dans Nom du groupe d'objets, entrez le nom du nouveau groupe à créer. Le programme d'installation crée le nouveau groupe pour vous.
 - Si vous sélectionnez Sélectionner un groupe existant, dans Nom du groupe d'objets, choisissez le groupe existant que vous souhaitez utiliser.
 - Si vous sélectionnez Aucun groupe, le programme d'installation n'ajoute pas le périphérique principal à un groupe d'objets.
5. Dans Étape 3 : Installation du logiciel Greengrass Core, effectuez les étapes suivantes.

Nucleus classic

1. Choisissez Nucleus classic comme moteur d'exécution du logiciel principal de votre appareil.
2. Choisissez le système d'exploitation de votre appareil principal : Linux ou Windows.

3. Fournissez vos AWS informations d'identification à l'appareil afin que le programme d'installation puisse fournir les ressources AWS IoT et IAM pour votre appareil principal. Pour renforcer la sécurité, nous vous recommandons d'obtenir des informations d'identification temporaires pour un rôle IAM qui n'accorde que les autorisations minimales nécessaires au provisionnement. Pour de plus amples informations, veuillez consulter [Politique IAM minimale permettant au programme d'installation de provisionner les ressources](#).

 Note

Le programme d'installation n'enregistre ni ne stocke vos informations d'identification.

Sur votre appareil, effectuez l'une des opérations suivantes pour récupérer les informations d'identification et les mettre à la disposition du programme d'installation du logiciel AWS IoT Greengrass Core :

- (Recommandé) Utilisez des informations d'identification temporaires provenant de AWS IAM Identity Center
 - a. Fournissez l'ID de clé d'accès, la clé d'accès secrète et le jeton de session provenant du centre d'identité IAM. Pour plus d'informations, voir Actualisation manuelle des informations d'identification dans la section Obtenir et actualiser [des informations d'identification temporaires](#) dans le guide de l'utilisateur d'IAM Identity Center.
 - b. Exécutez les commandes suivantes pour fournir les informations d'identification au logiciel AWS IoT Greengrass Core.

Linux or Unix

```
export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/
bPxRfiCYEXAMPLEKEY
export AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o50ytwEXAMPLE=
```

Windows Command Prompt (CMD)

```
set AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
```

```
set AWS_SECRET_ACCESS_KEY=wJa1rXUtnFEMI/K7MDENG/  
bPxRfiCYEXAMPLEKEY  
set AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o50ytwEXAMPLE=
```

PowerShell

```
$env:AWS_ACCESS_KEY_ID="AKIAIOSFODNN7EXAMPLE"  
$env:AWS_SECRET_ACCESS_KEY="wJa1rXUtnFEMI/K7MDENG/  
bPxRfiCYEXAMPLEKEY"  
$env:AWS_SESSION_TOKEN="AQoDYXdzEJr1K...o50ytwEXAMPLE="
```

- Utilisez les informations d'identification de sécurité temporaires issues d'un rôle IAM :
 - a. Fournissez l'ID de clé d'accès, la clé d'accès secrète et le jeton de session correspondant au rôle IAM que vous assumez. Pour plus d'informations sur la façon de récupérer ces informations d'identification, consultez la section [Demande d'informations d'identification de sécurité temporaires](#) dans le guide de l'utilisateur IAM.
 - b. Exécutez les commandes suivantes pour fournir les informations d'identification au logiciel AWS IoT Greengrass Core.

Linux or Unix

```
export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE  
export AWS_SECRET_ACCESS_KEY=wJa1rXUtnFEMI/K7MDENG/  
bPxRfiCYEXAMPLEKEY  
export AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o50ytwEXAMPLE=
```

Windows Command Prompt (CMD)

```
set AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE  
set AWS_SECRET_ACCESS_KEY=wJa1rXUtnFEMI/K7MDENG/  
bPxRfiCYEXAMPLEKEY  
set AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o50ytwEXAMPLE=
```

PowerShell

```
$env:AWS_ACCESS_KEY_ID="AKIAIOSFODNN7EXAMPLE"
```

```
$env:AWS_SECRET_ACCESS_KEY="wJa1rXUtnFEMI/K7MDENG/  
bPxRfiCYEXAMPLEKEY"  
$env:AWS_SESSION_TOKEN="AQoDYXdzEJr1K...o50ytwEXAMPLE="
```

- Utilisez les informations d'identification à long terme d'un utilisateur IAM :
 - a. Fournissez l'ID de clé d'accès et la clé d'accès secrète pour votre utilisateur IAM. Vous pouvez créer un utilisateur IAM pour le provisionnement, que vous supprimerez ultérieurement. Pour connaître la politique IAM à communiquer à l'utilisateur, consultez [Politique IAM minimale permettant au programme d'installation de provisionner les ressources](#). Pour plus d'informations sur la façon de récupérer des informations d'identification à long terme, consultez [la section Gestion des clés d'accès pour les utilisateurs IAM](#) dans le guide de l'utilisateur IAM.
 - b. Exécutez les commandes suivantes pour fournir les informations d'identification au logiciel AWS IoT Greengrass Core.

Linux or Unix

```
export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE  
export AWS_SECRET_ACCESS_KEY=wJa1rXUtnFEMI/K7MDENG/  
bPxRfiCYEXAMPLEKEY
```

Windows Command Prompt (CMD)

```
set AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE  
set AWS_SECRET_ACCESS_KEY=wJa1rXUtnFEMI/K7MDENG/  
bPxRfiCYEXAMPLEKEY
```

PowerShell

```
$env:AWS_ACCESS_KEY_ID="AKIAIOSFODNN7EXAMPLE"  
$env:AWS_SECRET_ACCESS_KEY="wJa1rXUtnFEMI/K7MDENG/  
bPxRfiCYEXAMPLEKEY"
```

- c. (Facultatif) Si vous avez créé un utilisateur IAM pour approvisionner votre appareil Greengrass, supprimez-le.
- d. (Facultatif) Si vous avez utilisé l'ID de clé d'accès et la clé d'accès secrète d'un utilisateur IAM existant, mettez à jour les clés de cet utilisateur afin qu'elles ne

soient plus valides. Pour plus d'informations, consultez la section [Mise à jour des clés d'accès](#) dans le guide de Gestion des identités et des accès AWS l'utilisateur.

4. Sous Exécuter le programme d'installation, effectuez les étapes suivantes.
 - a. Sous Télécharger le programme d'installation, choisissez Copier et exécutez la commande copiée sur votre appareil principal. Cette commande télécharge la dernière version du logiciel AWS IoT Greengrass Core et la décompresse sur votre appareil.
 - b. Sous Exécuter le programme d'installation, choisissez Copier, puis exécutez la commande copiée sur votre appareil principal. Cette commande utilise les AWS IoT noms d'objets et de groupes d'objets que vous avez spécifiés précédemment pour exécuter le programme d'installation du logiciel AWS IoT Greengrass Core et configurer les AWS ressources pour votre périphérique principal.

Cette commande effectue également les opérations suivantes :

- Configurez le logiciel AWS IoT Greengrass Core en tant que service système qui s'exécute au démarrage. Sur les appareils Linux, cela nécessite le [système d'initialisation Systemd](#).

 Important

Sur les appareils Windows Core, vous devez configurer le logiciel AWS IoT Greengrass Core en tant que service système.

- Déployez le [composant AWS IoT Greengrass CLI](#), qui est un outil de ligne de commande qui vous permet de développer des composants Greengrass personnalisés sur le périphérique principal.
- Spécifiez l'utilisation de l'utilisateur `ggc_user` du système pour exécuter les composants logiciels sur le périphérique principal. Sur les appareils Linux, cette commande indique également d'utiliser le groupe `ggc_group` système, et le programme d'installation crée l'utilisateur et le groupe système pour vous.

Lorsque vous exécutez cette commande, les messages suivants devraient s'afficher pour indiquer que le programme d'installation a réussi.

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

 Note

Si vous possédez un appareil Linux et qu'il n'est pas doté de [systemd](#), le programme d'installation ne configurera pas le logiciel en tant que service système et vous ne verrez pas le message de confirmation de configuration du noyau en tant que service système.

Nucleus lite

1. Choisissez Nucleus lite comme moteur d'exécution du logiciel principal de votre appareil.
2. Sélectionnez la méthode de configuration de votre appareil pour connecter votre appareil à un appareil Greengrass Core.

Option 1 : configurer un appareil avec téléchargement du package (environ 1 Mo)

1. Créez un AWS IoT objet et un rôle pour Greengrass.
2. Téléchargez le fichier zip contenant AWS IoT les ressources auxquelles votre appareil doit se connecter AWS IoT :
 - Un certificat et une clé privée générés à l'aide AWS IoT de l'autorité de certification.
 - Un fichier de schéma pour lancer l'installation de Greengrass sur votre appareil.
3. Téléchargez le package qui installera le dernier runtime Greengrass Nucleus lite sur votre Raspberry Pi.
4. Provisionnez votre appareil pour qu'il devienne un appareil AWS IoT Greengrass Core et connectez-le à AWS IoT :
 - a. Transférez le package Greengrass et le kit de connexion sur votre appareil à l'aide d'une clé USB, d'un SCP/FTP ou d'une carte SD.
 - b. Décompressez le fichier greengrass-package.zip dans le GreengrassInstaller répertoire/de l'appareil.

- c. Décompressez le fichier zip du kit de connexion dans le répertoire /de l'appareil.
 - d. Exécutez la commande fournie sur le périphérique à installer AWS IoT Greengrass
5. Choisissez ensuite Afficher les principaux appareils.

Option 2 : configurer un appareil avec un téléchargement d'exemple d'image préconfiguré sur l'ensemble du disque (environ 100 Mo)

1. Créez un AWS IoT objet et un rôle pour Greengrass.
2. Téléchargez le fichier zip contenant AWS IoT les ressources auxquelles votre appareil doit se connecter AWS IoT :
 - Un certificat et une clé privée générés à l'aide AWS IoT de l'autorité de certification.
 - Un fichier de schéma pour lancer l'installation de Greengrass sur votre appareil.
3. Téléchargez l'image d'exemple de disque entier préconfigurée qui contient Greengrass et le système d'exploitation.
 - a. Pour transférer le kit de connexion et flasher l'image sur votre appareil, suivez le fichier readme téléchargé avec l'image.
 - b. Pour démarrer l'installation de Greengrass, allumez et démarrez l'appareil à partir de l'image clignotée
4. Choisissez ensuite Afficher les principaux appareils.

Option 3 : configurer un appareil avec votre propre version personnalisée

1. Créez un AWS IoT objet et un rôle pour Greengrass.
2. Téléchargez le fichier zip contenant AWS IoT les ressources auxquelles votre appareil doit se connecter AWS IoT :
 - Un certificat et une clé privée générés à l'aide AWS IoT de l'autorité de certification.
 - Un fichier de schéma pour lancer l'installation de Greengrass sur votre appareil.
3. Pour personnaliser et créer votre propre image à l'aide de Yocto à partir du code source, puis utiliser le kit de connexion pour installer Nucleus Lite, suivez les instructions ci-dessous. GitHub
 - Choisissez ensuite Afficher les principaux appareils.

Installation du logiciel AWS IoT Greengrass principal (CLI)

Note

Ces étapes ne s'appliquent pas à Nucleus Lite.

Pour installer et configurer le logiciel AWS IoT Greengrass Core

1. Sur votre appareil principal Greengrass, exécutez la commande suivante pour accéder au répertoire de base.

Linux or Unix

```
cd ~
```

Windows Command Prompt (CMD)

```
cd %USERPROFILE%
```

PowerShell

```
cd ~
```

2. Sur votre appareil principal, téléchargez le logiciel AWS IoT Greengrass Core dans un fichier nommé `greengrass-nucleus-latest.zip`.

Linux or Unix

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

Windows Command Prompt (CMD)

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

PowerShell

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

Si vous téléchargez ce logiciel, vous acceptez le [contrat de licence du logiciel Greengrass Core](#).

3. Décompressez le logiciel AWS IoT Greengrass Core dans un dossier de votre appareil. *GreengrassInstaller* Remplacez-le par le dossier que vous souhaitez utiliser.

Linux or Unix

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

Windows Command Prompt (CMD)

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

PowerShell

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

4. Exécutez la commande suivante pour lancer le programme d'installation du logiciel AWS IoT Greengrass Core. Cette commande exécute les opérations suivantes :
 - Créez les AWS ressources dont le périphérique principal a besoin pour fonctionner.
 - Configurez le logiciel AWS IoT Greengrass Core en tant que service système qui s'exécute au démarrage. Sur les appareils Linux, cela nécessite le [système d'initialisation Systemd](#).

Important

Sur les appareils Windows Core, vous devez configurer le logiciel AWS IoT Greengrass Core en tant que service système.

- Déployez le [composant AWS IoT Greengrass CLI](#), qui est un outil de ligne de commande qui vous permet de développer des composants Greengrass personnalisés sur le périphérique principal.
- Spécifiez d'utiliser l'utilisateur `ggc_user` du système pour exécuter les composants logiciels sur le périphérique principal. Sur les appareils Linux, cette commande indique également d'utiliser le groupe `ggc_group` système, et le programme d'installation crée l'utilisateur et le groupe système pour vous.

Remplacez les valeurs des arguments dans votre commande comme suit.

- a. */greengrass/v2* ou *C:\greengrass\v2* : chemin d'accès au dossier racine à utiliser pour installer le logiciel AWS IoT Greengrass Core.
- b. *GreengrassInstaller*. Le chemin d'accès au dossier dans lequel vous avez décompressé le programme d'installation du logiciel AWS IoT Greengrass Core.
- c. *region*. L' Région AWS endroit dans lequel trouver ou créer des ressources.
- d. *MyGreengrassCore*. Le nom de l' AWS IoT appareil principal de votre Greengrass. Si l'objet n'existe pas, le programme d'installation le crée. Le programme d'installation télécharge les certificats pour s'authentifier en tant qu' AWS IoT objet. Pour de plus amples informations, veuillez consulter [Authentification et autorisation de l'appareil pour AWS IoT Greengrass](#).

 Note

Le nom de l'objet ne peut pas contenir de caractères deux-points (:).

- e. *MyGreengrassCoreGroup*. Le nom du AWS IoT groupe d'objets de votre appareil Greengrass principal. Si le groupe d'objets n'existe pas, le programme d'installation le crée et y ajoute l'objet. Si le groupe d'objets existe et fait l'objet d'un déploiement actif, le périphérique principal télécharge et exécute le logiciel spécifié par le déploiement.

 Note

Le nom du groupe d'objets ne peut pas contenir de deux-points (:).

- f. *GreengrassV2IoTThingPolicy*. Le nom de la AWS IoT politique qui permet aux appareils principaux de Greengrass de communiquer avec AWS IoT et. AWS IoT

Greengrass Si la AWS IoT politique n'existe pas, le programme d'installation crée une AWS IoT politique permissive portant ce nom. Vous pouvez restreindre les autorisations de cette politique pour votre cas d'utilisation. Pour de plus amples informations, veuillez consulter [AWS IoT Politique minimale pour les appareils AWS IoT Greengrass V2 principaux](#).

- g. ***GreengrassV2TokenExchangeRole***. Nom du rôle IAM qui permet au périphérique principal de Greengrass d'obtenir AWS des informations d'identification temporaires. Si le rôle n'existe pas, le programme d'installation le crée, puis crée et attache une politique nommée ***GreengrassV2TokenExchangeRoleAccess***. Pour de plus amples informations, veuillez consulter [Autoriser les appareils principaux à interagir avec les AWS services](#).
- h. ***GreengrassCoreTokenExchangeRoleAlias***. Alias du rôle IAM qui permet au périphérique principal de Greengrass d'obtenir des informations d'identification temporaires ultérieurement. Si l'alias de rôle n'existe pas, le programme d'installation le crée et le pointe vers le rôle IAM que vous spécifiez. Pour de plus amples informations, veuillez consulter [Autoriser les appareils principaux à interagir avec les AWS services](#).

Linux or Unix

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

Windows Command Prompt (CMD)

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

```
--tes-role-alias-name GreengrassCoreTokenExchangeRoleAlias ^  
--component-default-user ggc_user ^  
--provision true ^  
--setup-system-service true ^  
--deploy-dev-tools true
```

PowerShell

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

Note

Si vous utilisez AWS IoT Greengrass un appareil dont la mémoire est limitée, vous pouvez contrôler la quantité de mémoire utilisée par le logiciel AWS IoT Greengrass Core. Pour contrôler l'allocation de mémoire, vous pouvez définir les options de taille de segment de mémoire JVM dans le paramètre de `jvmOptions` configuration de votre composant Nucleus. Pour de plus amples informations, veuillez consulter [Contrôlez l'allocation de mémoire grâce aux options JVM](#).

Lorsque vous exécutez cette commande, les messages suivants devraient s'afficher pour indiquer que le programme d'installation a réussi.

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

 Note

Si vous possédez un appareil Linux et qu'il n'est pas doté de [systemd](#), le programme d'installation ne configurera pas le logiciel en tant que service système et vous ne verrez pas le message de confirmation de configuration du noyau en tant que service système.

(Facultatif) Exécutez le logiciel Greengrass (Linux)

 Note

Ces étapes ne s'appliquent pas à Nucleus Lite.

Si vous avez installé le logiciel en tant que service système, le programme d'installation exécute le logiciel pour vous. Dans le cas contraire, vous devez exécuter le logiciel. Pour savoir si le programme d'installation a configuré le logiciel en tant que service système, recherchez la ligne suivante dans le résultat du programme d'installation.

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

Si ce message ne s'affiche pas, procédez comme suit pour exécuter le logiciel :

1. Exécutez la commande suivante pour exécuter le logiciel.

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

Le logiciel imprime le message suivant s'il démarre correctement.

```
Launched Nucleus successfully.
```

2. Vous devez laisser l'interface de commande actuelle ouverte pour que le logiciel AWS IoT Greengrass Core continue de fonctionner. Si vous utilisez SSH pour vous connecter au périphérique principal, exécutez la commande suivante sur votre ordinateur de développement pour ouvrir une deuxième session SSH que vous pouvez utiliser pour exécuter des commandes supplémentaires sur le périphérique principal. Remplacez *username* par le nom de l'utilisateur auquel vous souhaitez vous connecter et *pi-ip-address* par l'adresse IP de l'appareil.

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

Pour plus d'informations sur la manière d'interagir avec le service système Greengrass, consultez.

[Configurer le noyau Greengrass en tant que service système](#)

Vérifiez l'installation de Greengrass CLI sur l'appareil

Note

Ces étapes ne s'appliquent pas à Nucleus Lite.

Le déploiement de la CLI Greengrass peut prendre jusqu'à une minute. Exécutez la commande suivante pour vérifier l'état du déploiement. Remplacez *MyGreengrassCore* par le nom de votre appareil principal.

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

`coreDeviceExecutionStatus` indique l'état du déploiement sur le périphérique principal. Lorsque le statut est défini `SUCCEEDED`, exécutez la commande suivante pour vérifier que la CLI Greengrass est installée et fonctionne. Remplacez */greengrass/v2* par le chemin d'accès au dossier racine.

Linux or Unix

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

Windows Command Prompt (CMD)

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

PowerShell

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


La commande génère des informations d'aide pour la CLI Greengrass. S'il `greengrass-cli` n'est pas trouvé, le déploiement n'a peut-être pas réussi à installer la CLI Greengrass. Pour de plus amples informations, veuillez consulter [Résolution des problèmes AWS IoT Greengrass V2](#).

Vous pouvez également exécuter la commande suivante pour déployer manuellement la AWS IoT Greengrass CLI sur votre appareil.

- **region** Remplacez-le par Région AWS celui que vous utilisez. Assurez-vous d'utiliser le même Région AWS que celui que vous avez utilisé pour configurer le AWS CLI sur votre appareil.
- **account-id** Remplacez-le par votre Compte AWS identifiant.
- Remplacez **MyGreengrassCore** par le nom de votre appareil principal.

Linux, macOS, or Unix

```
aws greengrassv2 create-deployment \  
  --target-arn "arn:aws:iot:region:account-id:thing/MyGreengrassCore" \  
  --components '{  
    "aws.greengrass.Cli": {  
      "componentVersion": "2.16.1"  
    }  
  }'
```

Windows Command Prompt (CMD)

```
aws greengrassv2 create-deployment ^  
  --target-arn "arn:aws:iot:region:account-id:thing/MyGreengrassCore" ^  
  --components "{\"aws.greengrass.Cli\":{\"componentVersion\":\"2.16.1\"}}"
```

PowerShell

```
aws greengrassv2 create-deployment `  
  --target-arn "arn:aws:iot:region:account-id:thing/MyGreengrassCore" `  
  --components '{"aws.greengrass.Cli":{"componentVersion":"2.16.1"}}'
```

 Tip

Vous pouvez ajouter `/greengrass/v2/bin` (Linux) ou `C:\greengrass\v2\bin` (Windows) à votre variable d'PATH environnement pour qu'elle s'exécute `greengrass-cli` sans son chemin absolu.

Le logiciel AWS IoT Greengrass principal et les outils de développement local s'exécutent sur votre appareil. Vous pouvez ensuite développer un AWS IoT Greengrass composant Hello World sur votre appareil.

Étape 4 : développer et tester un composant sur votre appareil

Un composant est un module logiciel qui s'exécute sur les appareils AWS IoT Greengrass principaux. Les composants vous permettent de créer et de gérer des applications complexes sous forme de composants distincts que vous pouvez réutiliser d'un appareil principal de Greengrass à un autre. Chaque composant est composé d'une recette et d'artefacts.

- Recettes

Chaque composant contient un fichier de recette qui définit ses métadonnées. La recette spécifie également les paramètres de configuration, les dépendances des composants, le cycle de vie et la compatibilité de la plate-forme du composant. Le cycle de vie du composant définit les commandes qui installent, exécutent et arrêtent le composant. Pour de plus amples informations, veuillez consulter [AWS IoT Greengrass référence de recette de composant](#).

Vous pouvez définir des recettes au format [JSON](#) ou [YAML](#).

- Artefacts

Les composants peuvent avoir un nombre illimité d'artefacts, qui sont des binaires de composants. Les artefacts peuvent inclure des scripts, du code compilé, des ressources statiques et tout autre fichier consommé par un composant. Les composants peuvent également consommer des artefacts issus de leurs dépendances.

Avec AWS IoT Greengrass, vous pouvez utiliser la CLI Greengrass pour développer et tester des composants localement sur un appareil principal de Greengrass sans interaction avec le Cloud. AWS Lorsque vous avez terminé votre composant local, vous pouvez utiliser la recette et les artefacts

du composant pour créer ce composant dans le AWS IoT Greengrass service dans le AWS Cloud, puis le déployer sur tous vos appareils principaux Greengrass. Pour plus d'informations sur les composants, consultez [Développer des AWS IoT Greengrass composants](#).

Dans cette section, vous apprendrez à créer et à exécuter un composant Hello World de base en local sur votre appareil principal.

Pour développer un composant Hello World sur votre appareil

1. Créez un dossier pour vos composants avec des sous-dossiers pour les recettes et les artefacts. Exécutez les commandes suivantes sur votre appareil principal Greengrass pour créer ces dossiers et accéder au dossier des composants. Remplacez `~/greengrassv2` ou `%USERPROFILE%\greengrassv2` par le chemin d'accès au dossier à utiliser pour le développement local.

Linux or Unix

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

Windows Command Prompt (CMD)

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

PowerShell

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

2. Utilisez un éditeur de texte pour créer un fichier de recette qui définit les métadonnées, les paramètres, les dépendances, le cycle de vie et les capacités de la plateforme de votre composant. Incluez la version du composant dans le nom du fichier de recette afin de pouvoir identifier quelle recette reflète quelle version de composant. Vous pouvez choisir le format YAML ou JSON pour votre recette.

Par exemple, sur un système basé sur Linux, vous pouvez exécuter la commande suivante pour utiliser GNU nano pour créer le fichier.

JSON

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

YAML

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

Note

AWS IoT Greengrass utilise des versions sémantiques pour les composants. Les versions sémantiques suivent une majeure. mineur. système de numéro de patch. Par exemple, la version 1.0.0 représente la première version majeure d'un composant. Pour plus d'informations, consultez la [spécification de version sémantique](#).

3. Collez la recette suivante dans le fichier.

JSON

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

```

    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "run": "py -3 -u {artifacts:path}/hello_world.py {configuration:/
Message}"
      }
    }
  ]
}

```

YAML

```

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

```

La `ComponentConfiguration` section de cette recette définit un paramètre `Message`, dont la valeur par défaut est `world`. La `Manifests` section définit un manifeste, qui est un ensemble d'instructions de cycle de vie et d'artefacts pour une plate-forme. Vous pouvez définir plusieurs manifestes pour spécifier différentes instructions d'installation pour différentes plateformes, par exemple. Dans le manifeste, la `Lifecycle` section indique au périphérique principal de

Greengrass d'exécuter le script Hello World avec Message la valeur du paramètre comme argument.

4. Exécutez la commande suivante pour créer un dossier pour les artefacts du composant.

Linux or Unix

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

Windows Command Prompt (CMD)

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

PowerShell

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

Important

Vous devez utiliser le format suivant pour le chemin du dossier d'artefacts. Incluez le nom et la version du composant que vous spécifiez dans la recette.

```
artifacts/componentName/componentVersion/
```

5. Utilisez un éditeur de texte pour créer un fichier d'artefact de script Python pour votre composant Hello World.

Par exemple, sur un système basé sur Linux, vous pouvez exécuter la commande suivante pour utiliser GNU nano pour créer le fichier.

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

Copiez et collez le script Python suivant dans le fichier.

```
import sys

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

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

6. Utilisez la AWS IoT Greengrass CLI locale pour gérer les composants de votre appareil principal Greengrass.

Exécutez la commande suivante pour déployer le composant sur le AWS IoT Greengrass noyau. Remplacez `/greengrass/v2` ou `C:\greengrass\v2` par votre dossier AWS IoT Greengrass V2 racine, et remplacez `~/greengrassv2` ou `%USERPROFILE%\greengrassv2` par le dossier de développement de vos composants.

Linux or Unix

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

Windows Command Prompt (CMD)

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

PowerShell

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

Cette commande ajoute le composant qui utilise la recette dans `recipes` et le script Python dans `artifacts`. L'option `--merge` ajoute ou met à jour le composant et la version que vous spécifiez.

7. Le logiciel AWS IoT Greengrass Core enregistre la sortie standard du processus du composant dans les fichiers journaux du logs dossier. Exécutez la commande suivante pour vérifier que le composant Hello World s'exécute et imprime les messages.

Linux or Unix

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

Windows Command Prompt (CMD)

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

La type commande écrit le contenu du fichier sur le terminal. Exécutez cette commande plusieurs fois pour observer les modifications apportées au fichier.

PowerShell

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

Vous devriez voir des messages similaires à ceux de l'exemple suivant.

```
Hello, world!
```

Note

Si le fichier n'existe pas, le déploiement local n'est peut-être pas encore terminé. Si le fichier n'existe pas dans les 15 secondes, le déploiement a probablement échoué. Cela peut se produire si votre recette n'est pas valide, par exemple. Exécutez la commande suivante pour afficher le fichier journal AWS IoT Greengrass principal. Ce fichier inclut les journaux du service de déploiement de l'appareil principal Greengrass.

Linux or Unix

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

Windows Command Prompt (CMD)

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

La type commande écrit le contenu du fichier sur le terminal. Exécutez cette commande plusieurs fois pour observer les modifications apportées au fichier.

PowerShell

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

8. Modifiez le composant local pour itérer et tester votre code. Ouvrez `hello_world.py` dans un éditeur de texte et ajoutez le code suivant à la ligne 4 pour modifier le message enregistré par le AWS IoT Greengrass noyau.

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

Le `hello_world.py` script doit maintenant avoir le contenu suivant.

```
import sys

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

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

9. Exécutez la commande suivante pour mettre à jour le composant avec vos modifications.

Linux or Unix

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

Windows Command Prompt (CMD)

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

PowerShell

```
C:\greengrass\v2\bin\greengrass-cli deployment create `
```

```
--recipeDir ~/greengrassv2/recipes `
--artifactDir ~/greengrassv2/artifacts `
--merge "com.example.HelloWorld=1.0.0"
```

Cette commande met à jour le `com.example.HelloWorld` composant avec le dernier artefact Hello World.

10. Exécutez la commande suivante pour redémarrer le composant. Lorsque vous redémarrez un composant, le périphérique principal utilise les dernières modifications.

Linux or Unix

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

Windows Command Prompt (CMD)

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

PowerShell

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

11. Consultez à nouveau le journal pour vérifier que le composant Hello World imprime le nouveau message.

Linux or Unix

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

Windows Command Prompt (CMD)

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

La `type` commande écrit le contenu du fichier sur le terminal. Exécutez cette commande plusieurs fois pour observer les modifications apportées au fichier.

PowerShell

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

Vous devriez voir des messages similaires à ceux de l'exemple suivant.

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

12. Vous pouvez mettre à jour les paramètres de configuration du composant pour tester différentes configurations. Lorsque vous déployez un composant, vous pouvez spécifier une mise à jour de configuration, qui définit comment modifier la configuration du composant sur le périphérique principal. Vous pouvez spécifier les valeurs de configuration à rétablir aux valeurs par défaut et les nouvelles valeurs de configuration à fusionner sur le périphérique principal. Pour de plus amples informations, veuillez consulter [Mettre à jour les configurations des composants](#).

Procédez comme suit :

- a. Utilisez un éditeur de texte pour créer un fichier appelé `hello-world-config-update.json` pour contenir la mise à jour de configuration

Par exemple, sur un système basé sur Linux, vous pouvez exécuter la commande suivante pour utiliser GNU nano pour créer le fichier.

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

- b. Copiez et collez l'objet JSON suivant dans le fichier. Cet objet JSON définit une mise à jour de configuration qui fusionne la valeur `friend` avec le Message paramètre pour mettre à jour sa valeur. Cette mise à jour de configuration ne spécifie aucune valeur à réinitialiser. Il n'est pas nécessaire de réinitialiser le Message paramètre car la mise à jour de fusion remplace la valeur existante.

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

- c. Exécutez la commande suivante pour déployer la mise à jour de configuration sur le composant Hello World.

Linux or Unix

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

Windows Command Prompt (CMD)

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

PowerShell

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

- d. Consultez à nouveau le journal pour vérifier que le composant Hello World produit le nouveau message.

Linux or Unix

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

Windows Command Prompt (CMD)

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

La type commande écrit le contenu du fichier sur le terminal. Exécutez cette commande plusieurs fois pour observer les modifications apportées au fichier.

PowerShell

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

Vous devriez voir des messages similaires à ceux de l'exemple suivant.

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

13. Une fois que vous avez terminé de tester votre composant, supprimez-le de votre appareil principal. Exécutez la commande suivante.

Linux or Unix

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

Windows Command Prompt (CMD)

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

PowerShell

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

Important

Cette étape est nécessaire pour que vous puissiez redéployer le composant sur le périphérique principal après l'avoir chargé sur AWS IoT Greengrass. Dans le cas contraire, le déploiement échoue avec une erreur de compatibilité de version car le déploiement local spécifie une version différente du composant.

Exécutez la commande suivante et vérifiez que le `com.example.HelloWorld` composant n'apparaît pas dans la liste des composants de votre appareil.

Linux or Unix

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

Windows Command Prompt (CMD)

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

PowerShell

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

Votre composant Hello World est terminé et vous pouvez désormais le télécharger sur le service AWS IoT Greengrass cloud. Vous pouvez ensuite déployer le composant sur les appareils principaux de Greengrass.

Étape 5 : Créez votre composant dans le AWS IoT Greengrass service

Lorsque vous avez terminé de développer un composant sur votre appareil principal, vous pouvez le télécharger vers le AWS IoT Greengrass service dans le AWS Cloud. Vous pouvez également créer le composant directement dans la [AWS IoT Greengrass console](#). AWS IoT Greengrass fournit un service de gestion des composants qui héberge vos composants afin que vous puissiez les déployer sur des appareils individuels ou des flottes d'appareils. Pour télécharger un composant vers le AWS IoT Greengrass service, vous devez suivre les étapes suivantes :

- Téléchargez les artefacts des composants dans un compartiment S3.
- Ajoutez l'URI Amazon Simple Storage Service (Amazon S3) de chaque artefact à la recette du composant.
- Créez un composant à AWS IoT Greengrass partir de la recette du composant.

Dans cette section, vous devez effectuer ces étapes sur votre appareil principal Greengrass pour télécharger votre composant Hello World sur le AWS IoT Greengrass service.

Créez votre composant dans AWS IoT Greengrass (console)

1. Utilisez un compartiment S3 dans votre AWS compte pour héberger les artefacts des AWS IoT Greengrass composants. Lorsque vous déployez le composant sur un périphérique principal, celui-ci télécharge les artefacts du composant depuis le compartiment.

Vous pouvez utiliser un compartiment S3 existant ou en créer un nouveau.

- a. Dans la [console Amazon S3](#), sous Buckets, choisissez Create bucket.
- b. Pour Nom du compartiment, entrez un nom de compartiment unique. Par exemple, vous pouvez utiliser **greengrass-component-artifacts-*region*-123456789012**.
123456789012 Remplacez-le par votre identifiant de AWS compte et *region* par celui Région AWS que vous utilisez pour ce didacticiel.
- c. Pour AWS la région, sélectionnez la AWS région que vous utilisez pour ce didacticiel.
- d. Choisissez Créer un compartiment.
- e. Sous Buckets, choisissez le bucket que vous avez créé, téléchargez le `hello_world.py` script dans le `artifacts/com.example.HelloWorld/1.0.0` dossier du bucket. Pour plus d'informations sur le téléchargement d'objets vers des compartiments S3, consultez la section [Chargement d'objets](#) dans le guide de l'utilisateur d'Amazon Simple Storage Service.
- f. Copiez l'URI S3 de l'`hello_world.py` objet dans le compartiment S3. Cette URI doit ressembler à l'exemple suivant. Remplacez `amzn-s3-demo-bucket` par le nom du compartiment S3.

```
s3://amzn-s3-demo-bucket/artifacts/com.example.HelloWorld/1.0.0/hello_world.py
```

2. Autorisez le périphérique principal à accéder aux artefacts des composants dans le compartiment S3.

Chaque appareil principal possède un [rôle IAM principal](#) qui lui permet d'interagir avec le AWS cloud AWS IoT et d'envoyer des journaux vers celui-ci. Ce rôle de périphérique n'autorise pas l'accès aux compartiments S3 par défaut. Vous devez donc créer et associer une politique permettant au périphérique principal de récupérer les artefacts des composants du compartiment S3.

Si le rôle de votre appareil autorise déjà l'accès au compartiment S3, vous pouvez ignorer cette étape. Sinon, créez une politique IAM autorisant l'accès et associez-la au rôle, comme suit :

- a. Dans le menu de navigation de [la console IAM](#), choisissez Politiques, puis Create policy.
- b. Sur l'onglet JSON, remplacez le contenu de l'espace réservé par la stratégie suivante. Remplacez `amzn-s3-demo-bucket` par le nom du compartiment S3 contenant les artefacts des composants à télécharger par le périphérique principal.

```
{
```

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

- c. Choisissez Suivant.
 - d. Dans la section Détails de la politique, pour Nom, entrez **MyGreengrassV2ComponentArtifactPolicy**.
 - e. Choisissez Create Policy (Créer une politique).
 - f. Dans le menu de navigation de [la console IAM](#), choisissez Role, puis choisissez le nom du rôle pour le périphérique principal. Vous avez spécifié ce nom de rôle lors de l'installation du logiciel AWS IoT Greengrass Core. Si vous n'avez pas spécifié de nom, le nom par défaut est `GreengrassV2TokenExchangeRole`.
 - g. Sous Autorisations, choisissez Ajouter des autorisations, puis choisissez Joindre des politiques.
 - h. Sur la page Ajouter des autorisations, cochez la case à côté de la `MyGreengrassV2ComponentArtifactPolicy` politique que vous avez créée, puis choisissez Ajouter des autorisations.
3. Utilisez la recette du composant pour créer un composant dans la [AWS IoT Greengrass console](#).
- a. Dans le menu de navigation de la [AWS IoT Greengrass console](#), choisissez Composants, puis sélectionnez Créer un composant.
 - b. Sous Informations sur le composant, choisissez Enter recipe as JSON. La recette de remplacement doit ressembler à l'exemple suivant.

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.HelloWorld",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "My first AWS IoT Greengrass component.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
```



```

    "DefaultConfiguration": {
      "Message": "world"
    },
    "Manifests": [
      {
        "Platform": {
          "os": "linux"
        },
        "Lifecycle": {
          "Run": "python3 -u {artifacts:path}/hello_world.py \"{configuration:/
Message}\""
        },
        "Artifacts": [
          {
            "URI": "s3://amzn-s3-demo-bucket/artifacts/
com.example.HelloWorld/1.0.0/hello_world.py"
          }
        ]
      },
      {
        "Platform": {
          "os": "windows"
        },
        "Lifecycle": {
          "Run": "py -3 -u {artifacts:path}/hello_world.py \"{configuration:/
Message}\""
        },
        "Artifacts": [
          {
            "URI": "s3://amzn-s3-demo-bucket/artifacts/
com.example.HelloWorld/1.0.0/hello_world.py"
          }
        ]
      }
    ]
  }
}

```

- c. Remplacez l'URI de remplacement dans chaque Artifacts section par l'URI S3 de votre `hello_world.py` objet.
- d. Choisissez Créer un composant.
- e. Sur le `com.example.HelloWorld` page du composant, vérifiez que l'état du composant est déployable.

Créez votre composant dans AWS IoT Greengrass (AWS CLI)

Pour télécharger votre composant Hello World

1. Utilisez un compartiment S3 dans votre compartiment Compte AWS pour héberger les artefacts des AWS IoT Greengrass composants. Lorsque vous déployez le composant sur un périphérique principal, celui-ci télécharge les artefacts du composant depuis le compartiment.

Vous pouvez utiliser un compartiment S3 existant ou exécuter la commande suivante pour créer un compartiment. Cette commande crée un compartiment avec votre Compte AWS identifiant et Région AWS pour former un nom de compartiment unique. `123456789012` Remplacez-le par votre Compte AWS identifiant et `region` par celui Région AWS que vous avez utilisé pour ce didacticiel.

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

La commande affiche les informations suivantes si la demande aboutit.

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

2. Autorisez le périphérique principal à accéder aux artefacts des composants dans le compartiment S3.

Chaque appareil principal possède un [rôle IAM principal](#) qui lui permet AWS IoT d'interagir avec le AWS Cloud. Ce rôle de périphérique n'autorise pas l'accès aux compartiments S3 par défaut. Vous devez donc créer et associer une politique permettant au périphérique principal de récupérer les artefacts des composants du compartiment S3.

Si le rôle du périphérique principal autorise déjà l'accès au compartiment S3, vous pouvez ignorer cette étape. Sinon, créez une politique IAM autorisant l'accès et associez-la au rôle, comme suit :

- a. Créez un fichier appelé `component-artifact-policy.json` et copiez-y le code JSON suivant. Cette politique permet d'accéder à tous les fichiers d'un compartiment S3. Remplacez `amzn-s3-demo-bucket` par le nom du compartiment S3.

```
{
  "Version": "2012-10-17",
  "Statement": [
```

```
{
  "Effect": "Allow",
  "Action": [
    "s3:GetObject"
  ],
  "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*"
}
```

- b. Exécutez la commande suivante pour créer la politique à partir du document de stratégie dans `component-artifact-policy.json`.

Linux or Unix

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

Windows Command Prompt (CMD)

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

PowerShell

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

Copiez le nom Amazon Resource Name (ARN) de la politique à partir des métadonnées de la politique dans la sortie. Vous utilisez cet ARN pour associer cette politique au rôle principal de l'appareil à l'étape suivante.

- c. Exécutez la commande suivante pour associer la politique au rôle principal de l'appareil. Remplacez *GreengrassV2TokenExchangeRole* par le nom du rôle du périphérique principal. Vous avez spécifié ce nom de rôle lors de l'installation du logiciel AWS IoT Greengrass Core. Remplacez l'ARN de la politique par l'ARN de l'étape précédente.

Linux or Unix

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

Windows Command Prompt (CMD)

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

PowerShell

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

Si la commande n'a aucune sortie, elle a réussi. Le périphérique principal peut désormais accéder aux artefacts que vous téléchargez dans ce compartiment S3.

3. Téléchargez l'artefact du script Python Hello World dans le compartiment S3.

Exécutez la commande suivante pour télécharger le script sur le même chemin dans le compartiment où le script existe sur votre AWS IoT Greengrass cœur. Remplacez `amzn-s3-demo-bucket` par le nom du compartiment S3.

Linux or Unix

```
aws s3 cp \  
  artifacts/com.example.HelloWorld/1.0.0/hello_world.py \  
  s3://amzn-s3-demo-bucket/artifacts/com.example.HelloWorld/1.0.0/hello_world.py
```

Windows Command Prompt (CMD)

```
aws s3 cp ^
```

```
artifacts/com.example.HelloWorld/1.0.0/hello_world.py ^  
s3://amzn-s3-demo-bucket/artifacts/com.example.HelloWorld/1.0.0/hello_world.py
```

PowerShell

```
aws s3 cp `artifacts/com.example.HelloWorld/1.0.0/hello_world.py`  
s3://amzn-s3-demo-bucket/artifacts/com.example.HelloWorld/1.0.0/hello_world.py
```

La commande affiche une ligne qui commence par `upload` : si la demande aboutit.

4. Ajoutez l'URI Amazon S3 de l'artefact à la recette du composant.

L'URI Amazon S3 est composé du nom du compartiment et du chemin d'accès à l'objet artefact contenu dans le compartiment. L'URI Amazon S3 de votre artefact de script est l'URI vers lequel vous avez chargé l'artefact à l'étape précédente. Cette URI doit ressembler à l'exemple suivant. Remplacez `amzn-s3-demo-bucket` par le nom du compartiment S3.

```
s3://amzn-s3-demo-bucket/artifacts/com.example.HelloWorld/1.0.0/hello_world.py
```

Pour ajouter l'artefact à la recette, ajoutez une liste `Artifacts` contenant une structure avec l'URI Amazon S3.

JSON

```
"Artifacts": [  
  {  
    "URI": "s3://amzn-s3-demo-bucket/artifacts/com.example.HelloWorld/1.0.0/  
hello_world.py"  
  }  
]
```

Ouvrez le fichier de recette dans un éditeur de texte.

Par exemple, sur un système basé sur Linux, vous pouvez exécuter la commande suivante pour utiliser GNU nano pour créer le fichier.

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

Ajoutez l'artefact à la recette. Votre fichier de recette doit ressembler à l'exemple suivant.

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

```

    }
  ]
}
```

YAML

```

Artifacts:
  - URI: s3://amzn-s3-demo-bucket/artifacts/com.example.HelloWorld/1.0.0/
    hello_world.py
```

Ouvrez le fichier de recette dans un éditeur de texte.

Par exemple, sur un système basé sur Linux, vous pouvez exécuter la commande suivante pour utiliser GNU nano pour créer le fichier.

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

Ajoutez l'artefact à la recette. Votre fichier de recette doit ressembler à l'exemple suivant.

```

---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.HelloWorld
ComponentVersion: '1.0.0'
ComponentDescription: My first AWS IoT Greengrass component.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    Message: world
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      Run: |
        python3 -u {artifacts:path}/hello_world.py "{configuration:/Message}"
  Artifacts:
    - URI: s3://amzn-s3-demo-bucket/artifacts/com.example.HelloWorld/1.0.0/
hello_world.py
  - Platform:
      os: windows
    Lifecycle:
      Run: |
```

```
py -3 -u {artifacts:path}/hello_world.py "{configuration:/Message}"  
Artifacts:  
- URI: s3://amzn-s3-demo-bucket/artifacts/com.example.HelloWorld/1.0.0/  
hello_world.py
```

5. Créez une ressource de composant à AWS IoT Greengrass partir de la recette. Exécutez la commande suivante pour créer le composant à partir de la recette, que vous fournissez sous forme de fichier binaire.

JSON

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

YAML

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

La réponse ressemble à l'exemple suivant si la demande aboutit.

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

Copiez le arn depuis la sortie pour vérifier l'état du composant à l'étape suivante.

 Note

Vous pouvez également voir votre composant Hello World dans la [AWS IoT Greengrass console](#) sur la page Composants.

6. Vérifiez que le composant est créé et qu'il est prêt à être déployé. Lorsque vous créez un composant, son état est REQUESTED. AWS IoT Greengrass Vérifie ensuite que le composant est déployable. Vous pouvez exécuter la commande suivante pour connaître l'état du composant et vérifier que celui-ci est déployable. Remplacez le arn par l'ARN de l'étape précédente.

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

Si le composant est validé, la réponse indique que l'état du composant est DEPLOYABLE.

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

Votre composant Hello World est désormais disponible en AWS IoT Greengrass. Vous pouvez le redéployer sur cet appareil principal de Greengrass ou sur d'autres appareils principaux.

Étape 6 : Déployez votre composant

Avec AWS IoT Greengrass, vous pouvez déployer des composants sur des appareils individuels ou des groupes d'appareils. Lorsque vous déployez un composant, il AWS IoT Greengrass installe et exécute le logiciel de ce composant sur chaque machine cible. Vous spécifiez les composants à déployer et la mise à jour de configuration à déployer pour chaque composant. Vous pouvez également contrôler la manière dont le déploiement est déployé sur les appareils ciblés par le déploiement. Pour de plus amples informations, veuillez consulter [Déployer AWS IoT Greengrass des composants sur des appareils](#).

Dans cette section, vous allez redéployer votre composant Hello World sur votre appareil principal Greengrass.

Déployez votre composant (console)

1. Dans le menu de navigation de la [AWS IoT Greengrass console](#), sélectionnez Composants.
2. Sur la page Composants, sous l'onglet Mes composants, choisissez com.example.HelloWorld.
3. Dans la page com.example.HelloWorldpage, choisissez Déployer.
4. Dans Ajouter au déploiement, choisissez Créer un nouveau déploiement, puis Suivant.
5. Sur la page Specify target (Spécifier une cible), procédez comme suit :
 - a. Dans la case Nom, saisissez **Deployment for MyGreengrassCore**.
 - b. Pour Cible de déploiement, choisissez l'appareil principal et le nom de l' AWS IoT appareil principal. La valeur par défaut de ce didacticiel est **MyGreengrassCore**.
 - c. Choisissez Suivant.
6. Sur la page Sélectionner les composants, sous Mes composants, vérifiez que com.example.HelloWorldle composant est sélectionné, puis choisissez Next.
7. Sur la page Configurer les composants, sélectionnez com.example.HelloWorld, puis procédez comme suit :
 - a. Choisissez Configure component (Configurer un composant).
 - b. Sous Configuration update (Mise à jour de la configuration), dans Configuration to merge (Configuration à fusionner), saisissez la configuration suivante.

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

```
}
```

Cette mise à jour de configuration définit le Message paramètre Hello World sur universe pour l'appareil dans ce déploiement.

- c. Choisissez Confirmer.
- d. Choisissez Suivant.
8. Sur la page Configure advanced settings (Configurer les paramètres avancés), conservez les paramètres de configuration par défaut et choisissez Next (Suivant).
9. Sur la page Review (Révision), choisissez Deploy (Déployer).
10. Vérifiez que le déploiement est terminé correctement. L'exécution du déploiement peut prendre plusieurs minutes. Consultez le journal Hello World pour vérifier la modification. Exécutez la commande suivante sur votre appareil principal Greengrass.

Linux or Unix

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

Windows Command Prompt (CMD)

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

PowerShell

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

Vous devriez voir des messages similaires à ceux de l'exemple suivant.

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

Note

Si les messages du journal ne changent pas, le déploiement a échoué ou n'a pas atteint le périphérique principal. Cela peut se produire si votre appareil principal n'est pas connecté à Internet ou n'est pas autorisé à récupérer des artefacts de votre compartiment S3. Exécutez la commande suivante sur votre appareil principal pour

afficher le fichier journal du logiciel AWS IoT Greengrass Core. Ce fichier inclut les journaux du service de déploiement de l'appareil principal Greengrass.

Linux or Unix

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

Windows Command Prompt (CMD)

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

La type commande écrit le contenu du fichier sur le terminal. Exécutez cette commande plusieurs fois pour observer les modifications apportées au fichier.

PowerShell

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

Pour de plus amples informations, veuillez consulter [Résolution des problèmes AWS IoT Greengrass V2](#).

Déployez votre composant (AWS CLI)

Pour déployer votre composant Hello World

1. Sur votre ordinateur de développement, créez un fichier appelé `hello-world-deployment.json` et copiez-y le code JSON suivant. Ce fichier définit les composants et les configurations à déployer.

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

```
}
```

Ce fichier de configuration indique de déployer la version 1.0.0 du composant Hello World que vous avez développée et publiée lors de la procédure précédente. configurationUpdateSpécifie de fusionner la configuration du composant dans une chaîne codée en JSON. Cette mise à jour de configuration définit le Message paramètre Hello World sur universe pour l'appareil dans ce déploiement.

2. Exécutez la commande suivante pour déployer le composant sur votre appareil principal Greengrass. Vous pouvez effectuer un déploiement sur des objets, qui sont des appareils individuels, ou sur des groupes d'objets, qui sont des groupes d'appareils. *MyGreengrassCore* Remplacez-le par le nom de l' AWS IoT objet correspondant à votre appareil principal.

Linux or Unix

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

Windows Command Prompt (CMD)

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

PowerShell

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

La commande produit une réponse similaire à l'exemple suivant.

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

```
}
```

3. Vérifiez que le déploiement est terminé correctement. L'exécution du déploiement peut prendre plusieurs minutes. Consultez le journal Hello World pour vérifier la modification. Exécutez la commande suivante sur votre appareil principal Greengrass.

Linux or Unix

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

Windows Command Prompt (CMD)

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

PowerShell

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

Vous devriez voir des messages similaires à ceux de l'exemple suivant.

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

Note

Si les messages du journal ne changent pas, le déploiement a échoué ou n'a pas atteint le périphérique principal. Cela peut se produire si votre appareil principal n'est pas connecté à Internet ou n'est pas autorisé à récupérer des artefacts de votre compartiment S3. Exécutez la commande suivante sur votre appareil principal pour afficher le fichier journal du logiciel AWS IoT Greengrass Core. Ce fichier inclut les journaux du service de déploiement de l'appareil principal Greengrass.

Linux or Unix

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

Windows Command Prompt (CMD)

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

La type commande écrit le contenu du fichier sur le terminal. Exécutez cette commande plusieurs fois pour observer les modifications apportées au fichier.

PowerShell

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

Pour de plus amples informations, veuillez consulter [Résolution des problèmes AWS IoT Greengrass V2](#).

Étapes suivantes

Vous avez terminé ce didacticiel. Le logiciel AWS IoT Greengrass principal et votre composant Hello World s'exécutent sur votre appareil. Votre composant Hello World est également disponible dans le service AWS IoT Greengrass cloud pour être déployé sur d'autres appareils. Pour plus d'informations sur les sujets abordés dans ce didacticiel, consultez les rubriques suivantes :

- [Création de AWS IoT Greengrass composants](#)
- [Publiez des composants à déployer sur vos appareils principaux](#)
- [Déployer AWS IoT Greengrass des composants sur des appareils](#)

Configuration des appareils AWS IoT Greengrass principaux

Effectuez les tâches décrites dans cette section pour installer, configurer et exécuter le logiciel AWS IoT Greengrass Core.

Note

Cette section décrit l'installation et la configuration avancées du logiciel AWS IoT Greengrass Core. Ces étapes ne s'appliquent pas à Nucleus Lite. Si vous utilisez Nucleus Lite pour la première fois AWS IoT Greengrass V2, nous vous recommandons de suivre le [didacticiel de démarrage](#) pour configurer un appareil principal et explorer les fonctionnalités de. AWS IoT Greengrass

Rubriques

- [Plateformes prises en charge](#)
- [Exigences relatives aux dispositifs](#)
- [Exigences relatives à la fonction Lambda](#)
- [Configurez un Compte AWS](#)
- [Installation du logiciel AWS IoT Greengrass de base](#)
- [Exécutez le logiciel AWS IoT Greengrass Core](#)
- [Exécuter le logiciel AWS IoT Greengrass Core dans un conteneur Docker](#)
- [Configuration du logiciel AWS IoT Greengrass de base](#)
- [Mettre à jour le logiciel AWS IoT Greengrass principal \(OTA\)](#)
- [Désinstallez le logiciel AWS IoT Greengrass Core](#)

Plateformes prises en charge

- [Plateformes prises en charge par Greengrass Nucleus](#)
- [Plateformes prises en charge par Greengrass Nucleus Lite](#)

Exigences relatives aux dispositifs

- [Exigences relatives à l'appareil Greengrass Nucleus](#)
- [Configuration requise pour les appareils Greengrass Nucleus Lite](#)

Exigences relatives à la fonction Lambda

Important

Les fonctions Greengrass Lambda ne sont actuellement pas prises en charge par Greengrass nucleus lite.

Votre appareil doit répondre aux exigences suivantes pour exécuter les fonctions Lambda :

- Système d'exploitation basé sur Linux.
- Votre appareil doit disposer de la commande `mkfifo shell`.
- Votre appareil doit exécuter les bibliothèques de langage de programmation requises par une fonction Lambda. Vous devez installer les bibliothèques requises sur le périphérique et les ajouter à la variable d'PATH d'environnement. Greengrass prend en charge toutes les versions compatibles avec Lambda des environnements d'exécution Python, Node.js et Java. Greengrass n'applique aucune restriction supplémentaire aux versions d'exécution Lambda obsolètes. Pour plus d'informations sur la AWS IoT Greengrass prise en charge des environnements d'exécution Lambda, consultez [Exécuter AWS Lambda des fonctions](#)
- Pour exécuter des fonctions Lambda conteneurisées, votre appareil doit répondre aux exigences suivantes :
 - Noyau Linux 4.4 ou version ultérieure.
 - Le noyau doit prendre en charge [les cgroups](#) v1, et vous devez activer et monter les cgroups suivants :
 - Le groupe de mémoire permettant de définir la limite de mémoire AWS IoT Greengrass pour les fonctions Lambda conteneurisées.
 - Le groupe de périphériques pour les fonctions Lambda conteneurisées permettant d'accéder aux périphériques ou aux volumes du système.

Le logiciel AWS IoT Greengrass Core ne prend pas en charge les cgroups v2.

Pour répondre à cette exigence, démarrez le périphérique avec les paramètres du noyau Linux suivants.

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

 Tip

Sur un Raspberry Pi, modifiez le `/boot/cmdline.txt` fichier pour définir les paramètres du noyau de l'appareil.

- Vous devez activer les configurations de noyau Linux suivantes 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
 - Autres :
 - CONFIG_POSIX_MQUEUE
 - CONFIG_OVERLAY_FS
 - CONFIG_HAVE_ARCH_SECCOMP_FILTER
 - CONFIG_SECCOMP_FILTER
 - CONFIG_KEYS
 - CONFIG_SECCOMP
 - CONFIG_SHMEM

 Tip

Consultez la documentation de votre distribution Linux pour savoir comment vérifier et définir les paramètres du noyau Linux. Vous pouvez également utiliser AWS IoT Device

Tester for AWS IoT Greengrass pour vérifier que votre appareil répond à ces exigences. Pour de plus amples informations, veuillez consulter [Utilisation AWS IoT Device Tester pour la AWS IoT Greengrass V2](#).

Configurez 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 l'<https://portal.aws.amazon.com/billing/inscription>.
2. Suivez les instructions en ligne.

Dans le cadre de la procédure d'inscription, vous recevrez un appel téléphonique ou un SMS 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 AWS est créé. Par défaut, seul l'utilisateur racine a accès à l'ensemble des Services AWS et des ressources de ce compte. La meilleure pratique de sécurité consiste à attribuer un accès administratif à un utilisateur, et à utiliser uniquement l'utilisateur racine pour effectuer les [tâches nécessitant un accès utilisateur racine](#).

Afin de créer un utilisateur administrateur, choisissez l'une des options suivantes :

Choisissez un moyen de gérer votre administrateur	Pour	En	Vous pouvez également
Dans IAM Identity Center	Utiliser des identifiants à court terme pour accéder à AWS.	Suivant les instructions fournies dans Mise en route dans le Guide de	Configurez l'accès par programmation en configurant le AWS CLI à utiliser AWS IAM Identity Center dans le

Choisissez un moyen de gérer votre administrateur	Pour	En	Vous pouvez également
(Recommandé)	C'est conforme aux bonnes pratiques en matière de sécurité. Pour plus d'informations sur les bonnes pratiques, consultez Bonnes pratiques de sécurité dans IAM dans le Guide de l'utilisateur IAM.	l'utilisateur AWS IAM Identity Center .	guide de l'AWS Command Line Interface utilisateur.
Dans IAM (Non recommandé)	Utiliser les informations d'identification à long terme pour accéder à AWS.	Suivant les instructions fournies dans Création d'un utilisateur IAM pour l'accès d'urgence dans le Guide de l'utilisateur IAM.	Configurer l'accès par programmation en suivant les instructions fournies dans Gestion des clés d'accès pour les utilisateurs IAM dans le Guide de l'utilisateur IAM.

Installation du logiciel AWS IoT Greengrass de base

AWS IoT Greengrass s'étend AWS aux appareils périphériques afin qu'ils puissent agir sur les données qu'ils génèrent, tout en les utilisant à des AWS Cloud fins de gestion, d'analyse et de stockage durable. Installez le logiciel AWS IoT Greengrass Core sur les appareils de pointe pour l'intégrer AWS IoT Greengrass et le AWS Cloud.

 Important

Avant de télécharger et d'installer le logiciel AWS IoT Greengrass Core, vérifiez que votre appareil principal répond à la [configuration requise](#) pour installer et exécuter le logiciel AWS IoT Greengrass Core v2.0.

Le logiciel AWS IoT Greengrass Core inclut un programme d'installation qui configure votre appareil en tant qu'appareil principal Greengrass. Lorsque vous exécutez le programme d'installation, vous pouvez configurer des options, telles que le dossier racine et le dossier Région AWS à utiliser. Vous pouvez demander au programme d'installation de créer les ressources requises AWS IoT et IAM pour vous. Vous pouvez également choisir de déployer des outils de développement locaux pour configurer un appareil que vous utilisez pour le développement de composants personnalisés.

Le logiciel de AWS IoT Greengrass base nécessite les ressources suivantes AWS IoT et IAM pour se connecter AWS Cloud et fonctionner :

- N'importe AWS IoT quel truc. Lorsque vous enregistrez un appareil en tant qu' AWS IoT objet, celui-ci peut utiliser un certificat numérique pour s'authentifier. AWS Ce certificat permet à l'appareil de communiquer avec AWS IoT et AWS IoT Greengrass. Pour de plus amples informations, veuillez consulter [Authentification et autorisation de l'appareil pour AWS IoT Greengrass](#).
- (Facultatif) Un AWS IoT groupe d'objets. Vous utilisez des groupes d'objets pour gérer des flottes d'appareils principaux de Greengrass. Lorsque vous déployez des composants logiciels sur vos appareils, vous pouvez choisir de les déployer sur des appareils individuels ou sur des groupes d'appareils. Vous pouvez ajouter un appareil à un groupe d'objets pour déployer les composants logiciels de ce groupe d'objets sur l'appareil. Pour de plus amples informations, veuillez consulter [Déployer AWS IoT Greengrass des composants sur des appareils](#).
- Un rôle IAM. Les appareils Greengrass Core utilisent le fournisseur AWS IoT Core d'informations d'identification pour autoriser les appels aux AWS services dotés d'un rôle IAM. Ce rôle permet à votre appareil d'interagir avec Amazon Logs AWS IoT, d'envoyer des CloudWatch journaux à Amazon Logs et de télécharger des artefacts de composants personnalisés depuis Amazon Simple Storage Service (Amazon S3). Pour de plus amples informations, veuillez consulter [Autoriser les appareils principaux à interagir avec les AWS services](#).
- Un alias de AWS IoT rôle. Les appareils Greengrass Core utilisent l'alias de rôle pour identifier le rôle IAM à utiliser. L'alias de rôle vous permet de modifier le rôle IAM tout en conservant la même configuration de l'appareil. Pour plus d'informations, consultez la section [Autorisation des appels directs vers AWS des services](#) dans le Guide du AWS IoT Core développeur.

Choisissez l'une des options suivantes pour installer le logiciel AWS IoT Greengrass Core sur votre appareil principal.

- Installation rapide

Choisissez cette option pour configurer un appareil principal Greengrass en un minimum d'étapes. Le programme d'installation crée les ressources AWS IoT et IAM requises pour vous. Cette option nécessite que vous fournissiez des AWS informations d'identification au programme d'installation pour créer des ressources dans votre Compte AWS.

Vous ne pouvez pas utiliser cette option pour effectuer une installation derrière un pare-feu ou un proxy réseau. Si vos appareils se trouvent derrière un pare-feu ou un proxy réseau, envisagez une [installation manuelle](#).

Pour de plus amples informations, veuillez consulter [Installation AWS IoT Greengrass du logiciel Core avec provisionnement automatique des ressources](#).

- Installation manuelle

Choisissez cette option pour créer les AWS ressources requises manuellement ou pour les installer derrière un pare-feu ou un proxy réseau. En utilisant une installation manuelle, vous n'avez pas besoin d'autoriser le programme d'installation à créer des ressources dans votre ordinateur Compte AWS, car c'est vous qui créez les ressources requises AWS IoT et les ressources IAM. Vous pouvez également configurer votre appareil pour qu'il se connecte sur le port 443 ou via un proxy réseau. Vous pouvez également configurer le logiciel AWS IoT Greengrass Core pour utiliser une clé privée et un certificat que vous stockez dans un module de sécurité matériel (HSM), un module de plateforme sécurisée (TPM) ou un autre élément cryptographique.

Pour de plus amples informations, veuillez consulter [Installation AWS IoT Greengrass du logiciel Core avec provisionnement manuel des ressources](#).

- Installation avec approvisionnement AWS IoT de flotte

Choisissez cette option pour créer les AWS ressources requises à partir d'un modèle de provisionnement de AWS IoT flotte. Vous pouvez choisir cette option pour créer des appareils similaires dans un parc, ou si vous fabriquez des appareils que vos clients activeront ultérieurement, tels que des véhicules ou des appareils domotiques. Les appareils utilisent des certificats de réclamation pour authentifier et approvisionner les AWS ressources, y compris un certificat client X.509 que l'appareil utilise AWS Cloud pour se connecter à des fins de fonctionnement normal. Vous pouvez intégrer ou flasher les certificats de réclamation dans

le matériel de l'appareil pendant la fabrication, et vous pouvez utiliser le même certificat de réclamation et la même clé pour approvisionner plusieurs appareils. Vous pouvez également configurer les appareils pour qu'ils se connectent sur le port 443 ou via un proxy réseau.

Pour de plus amples informations, veuillez consulter [Installation AWS IoT Greengrass du logiciel de base avec provisionnement du AWS IoT parc](#).

- Installation avec provisionnement personnalisé

Choisissez cette option pour développer une application Java personnalisée qui fournit les AWS ressources requises. Vous pouvez choisir cette option si vous [créez vos propres certificats clients X.509](#) ou si vous souhaitez mieux contrôler le processus de provisionnement. AWS IoT Greengrass fournit une interface que vous pouvez implémenter pour échanger des informations entre votre application de provisionnement personnalisée et le programme d'installation du logiciel AWS IoT Greengrass Core.

Pour de plus amples informations, veuillez consulter [Installez le logiciel AWS IoT Greengrass Core avec un provisionnement personnalisé des ressources](#).

AWS IoT Greengrass fournit également des environnements conteneurisés qui exécutent le logiciel AWS IoT Greengrass Core. Vous pouvez utiliser un Dockerfile pour [l'exécuter AWS IoT Greengrass dans un conteneur Docker](#).

Rubriques

- [Installation AWS IoT Greengrass du logiciel Core avec provisionnement automatique des ressources](#)
- [Installation AWS IoT Greengrass du logiciel Core avec provisionnement manuel des ressources](#)
- [Installation AWS IoT Greengrass du logiciel de base avec provisionnement du AWS IoT parc](#)
- [Installez le logiciel AWS IoT Greengrass Core avec un provisionnement personnalisé des ressources](#)
- [Arguments d'installation](#)

Installation AWS IoT Greengrass du logiciel Core avec provisionnement automatique des ressources

Le logiciel AWS IoT Greengrass Core inclut un programme d'installation qui configure votre appareil en tant qu'appareil principal Greengrass. Pour configurer rapidement un appareil, le programme

d'installation peut configurer l' AWS IoT AWS IoT objet, le groupe d'objets, le rôle IAM et l'alias de AWS IoT rôle dont le périphérique principal a besoin pour fonctionner. Le programme d'installation peut également déployer les outils de développement locaux sur le périphérique principal, afin que vous puissiez utiliser l'appareil pour développer et tester des composants logiciels personnalisés. Le programme d'installation a besoin AWS d'informations d'identification pour provisionner ces ressources et créer le déploiement.

Si vous ne pouvez pas fournir AWS d'informations d'identification à l'appareil, vous pouvez fournir les AWS ressources dont le périphérique principal a besoin pour fonctionner. Vous pouvez également déployer les outils de développement sur un appareil principal pour l'utiliser comme périphérique de développement. Cela vous permet de fournir moins d'autorisations à l'appareil lorsque vous exécutez le programme d'installation. Pour de plus amples informations, veuillez consulter [Installation AWS IoT Greengrass du logiciel Core avec provisionnement manuel des ressources](#).

Important

Avant de télécharger le logiciel AWS IoT Greengrass Core, vérifiez que votre appareil principal répond à la [configuration requise](#) pour installer et exécuter le logiciel AWS IoT Greengrass Core v2.0.

Rubriques

- [Configuration de l'environnement de l'appareil](#)
- [Fournir des AWS informations d'identification à l'appareil](#)
- [Téléchargez le logiciel AWS IoT Greengrass de base](#)
- [Installation du logiciel AWS IoT Greengrass de base](#)

Configuration de l'environnement de l'appareil

Suivez les étapes décrites dans cette section pour configurer un appareil Linux ou Windows à utiliser comme périphérique AWS IoT Greengrass principal.

Configuration d'un appareil Linux

Pour configurer un appareil Linux pour AWS IoT Greengrass V2

1. Installez le moteur d'exécution Java, dont le logiciel AWS IoT Greengrass Core a besoin pour fonctionner. Nous vous recommandons d'utiliser les versions de support à long terme d'[Amazon](#)

[Corretto](#) ou d'OpenJDK. La version 8 ou supérieure est requise. Les commandes suivantes vous montrent comment installer OpenJDK sur votre appareil.

- Pour les distributions basées sur Debian ou Ubuntu :

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

- Pour les distributions basées sur Red Hat :

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

- Dans Amazon Linux 2 :

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

- Pour Amazon Linux 2023 :

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

Lorsque l'installation est terminée, exécutez la commande suivante pour vérifier que Java s'exécute sur votre appareil Linux.

```
java -version
```

La commande affiche la version de Java exécutée sur le périphérique. Par exemple, sur une distribution basée sur Debian, le résultat peut ressembler à celui de l'exemple suivant.

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

2. (Facultatif) Créez l'utilisateur système et le groupe par défaut qui exécutent les composants sur le périphérique. Vous pouvez également choisir de laisser le programme d'installation du logiciel AWS IoT Greengrass Core créer cet utilisateur et ce groupe lors de l'installation avec l'argument `--component-default-user` installer. Pour de plus amples informations, veuillez consulter [Arguments d'installation](#).

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

```
sudo groupadd --system ggc_group
```

3. Vérifiez que l'utilisateur qui exécute le logiciel AWS IoT Greengrass Core (généralement `root`) est autorisé à exécuter `sudo` avec n'importe quel utilisateur et n'importe quel groupe.

- a. Exécutez la commande suivante pour ouvrir le `/etc/sudoers` fichier.

```
sudo visudo
```

- b. Vérifiez que l'autorisation accordée à l'utilisateur ressemble à l'exemple suivant.

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

4. (Facultatif) Pour [exécuter des fonctions Lambda conteneurisées](#), vous devez activer [cgroups](#) v1, et vous devez activer et monter les cgroups de mémoire et de périphériques. Si vous ne prévoyez pas d'exécuter des fonctions Lambda conteneurisées, vous pouvez ignorer cette étape.

Pour activer ces options cgroups, démarrez le périphérique avec les paramètres du noyau Linux suivants.

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

Pour plus d'informations sur l'affichage et la définition des paramètres du noyau de votre appareil, consultez la documentation de votre système d'exploitation et de votre chargeur de démarrage. Suivez les instructions pour définir définitivement les paramètres du noyau.

5. Installez toutes les autres dépendances requises sur votre appareil, comme indiqué dans la liste des exigences figurant dans [Exigences relatives aux dispositifs](#).

Configuration d'un appareil Windows

Note

Cette fonctionnalité est disponible pour les versions 2.5.0 et ultérieures du composant [Greengrass](#) nucleus.

Pour configurer un appareil Windows pour AWS IoT Greengrass V2

1. Installez le moteur d'exécution Java, dont le logiciel AWS IoT Greengrass Core a besoin pour fonctionner. Nous vous recommandons d'utiliser les versions de support à long terme d'[Amazon Corretto](#) ou d'OpenJDK. La version 8 ou supérieure est requise.
2. Vérifiez si Java est disponible sur la variable système [PATH](#), et ajoutez-le dans le cas contraire. Le LocalSystem compte exécute le logiciel AWS IoT Greengrass Core. Vous devez donc ajouter Java à la variable système PATH au lieu de la variable utilisateur PATH pour votre utilisateur. Procédez comme suit :
 - a. Appuyez sur la touche Windows pour ouvrir le menu de démarrage.
 - b. Tapez **environment variables** pour rechercher les options du système dans le menu Démarrer.
 - c. Dans les résultats de recherche du menu Démarrer, choisissez Modifier les variables d'environnement du système pour ouvrir la fenêtre des propriétés du système.
 - d. Choisissez les variables d'environnement... pour ouvrir la fenêtre des variables d'environnement.
 - e. Sous Variables système, sélectionnez Chemin, puis Modifier. Dans la fenêtre Modifier la variable d'environnement, vous pouvez afficher chaque chemin sur une ligne distincte.
 - f. Vérifiez si le chemin d'accès au bin dossier d'installation de Java est présent. Le chemin peut ressembler à celui de l'exemple suivant.

```
C:\\Program Files\\Amazon Corretto\\jdk11.0.13_8\\bin
```
 - g. Si le bin dossier de l'installation Java est absent de Path, choisissez Nouveau pour l'ajouter, puis cliquez sur OK.
3. Ouvrez l'invite de commande Windows (cmd . exe) en tant qu'administrateur.
4. Créez l'utilisateur par défaut dans le LocalSystem compte sur l'appareil Windows. *password* Remplacez-le par un mot de passe sécurisé.

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

Tip

En fonction de votre configuration Windows, le mot de passe de l'utilisateur peut être configuré pour expirer à une date ultérieure. Pour vous assurer que vos applications

Greengrass continue de fonctionner, suivez la date d'expiration du mot de passe et mettez-le à jour avant son expiration. Vous pouvez également définir le mot de passe de l'utilisateur pour qu'il n'expire jamais.

- Pour vérifier la date d'expiration d'un utilisateur et de son mot de passe, exécutez la commande suivante.

```
net user ggc_user | findstr /C:expires
```

- Pour définir le mot de passe d'un utilisateur de manière à ce qu'il n'expire jamais, exécutez la commande suivante.

```
wmic UserAccount where "Name='ggc_user'" set PasswordExpires=False
```

- Si vous utilisez Windows 10 ou une version ultérieure où la [wmic commande est obsolète](#), exécutez la commande suivante PowerShell .

```
Get-CimInstance -Query "SELECT * from Win32_UserAccount WHERE name = 'ggc_user'" | Set-CimInstance -Property @{PasswordExpires="False"}
```

5. Téléchargez et installez l'[PsExec utilitaire](#) de Microsoft sur l'appareil.
6. Utilisez l' PsExec utilitaire pour stocker le nom d'utilisateur et le mot de passe de l'utilisateur par défaut dans l'instance Credential Manager du LocalSystem compte. *password* Remplacez-le par le mot de passe utilisateur que vous avez défini précédemment.

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

S'il PsExec License Agreements'ouvre, choisissez Acceptd'accepter la licence et exécutez la commande.

Note

Sur les appareils Windows, le LocalSystem compte exécute le noyau Greengrass, et vous devez utiliser l' PsExec utilitaire pour stocker les informations utilisateur par défaut dans le LocalSystem compte. L'application Credential Manager stocke ces informations dans le compte Windows de l'utilisateur actuellement connecté, plutôt que dans le LocalSystem compte.

Fournir des AWS informations d'identification à l'appareil

Fournissez vos AWS informations d'identification à votre appareil afin que le programme d'installation puisse fournir les AWS ressources nécessaires. Pour plus d'informations sur les autorisations requises, consultez [Politique IAM minimale permettant au programme d'installation de provisionner les ressources](#).

Pour fournir des AWS informations d'identification à l'appareil

- Fournissez vos AWS informations d'identification à l'appareil afin que le programme d'installation puisse fournir les ressources AWS IoT et IAM pour votre appareil principal. Pour renforcer la sécurité, nous vous recommandons d'obtenir des informations d'identification temporaires pour un rôle IAM qui n'accorde que les autorisations minimales nécessaires au provisionnement. Pour de plus amples informations, veuillez consulter [Politique IAM minimale permettant au programme d'installation de provisionner les ressources](#).

Note

Le programme d'installation n'enregistre ni ne stocke vos informations d'identification.

Sur votre appareil, effectuez l'une des opérations suivantes pour récupérer les informations d'identification et les mettre à la disposition du programme d'installation du logiciel AWS IoT Greengrass Core :

- (Recommandé) Utilisez des informations d'identification temporaires provenant de AWS IAM Identity Center
 - Fournissez l'ID de clé d'accès, la clé d'accès secrète et le jeton de session provenant du centre d'identité IAM. Pour plus d'informations, voir Actualisation manuelle des informations d'identification dans la section Obtenir et actualiser [des informations d'identification temporaires](#) dans le guide de l'utilisateur d'IAM Identity Center.
 - Exécutez les commandes suivantes pour fournir les informations d'identification au logiciel AWS IoT Greengrass Core.

Linux or Unix

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

```
export AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o50ytwEXAMPLE=
```

Windows Command Prompt (CMD)

```
set AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE  
set AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY  
set AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o50ytwEXAMPLE=
```

PowerShell

```
$env:AWS_ACCESS_KEY_ID="AKIAIOSFODNN7EXAMPLE"  
$env:AWS_SECRET_ACCESS_KEY="wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY"  
$env:AWS_SESSION_TOKEN="AQoDYXdzEJr1K...o50ytwEXAMPLE="
```

- Utilisez les informations d'identification de sécurité temporaires issues d'un rôle IAM :
 - a. Fournissez l'ID de clé d'accès, la clé d'accès secrète et le jeton de session correspondant au rôle IAM que vous assumez. Pour plus d'informations sur la façon de récupérer ces informations d'identification, consultez la section [Demande d'informations d'identification de sécurité temporaires](#) dans le guide de l'utilisateur IAM.
 - b. Exécutez les commandes suivantes pour fournir les informations d'identification au logiciel AWS IoT Greengrass Core.

Linux or Unix

```
export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE  
export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY  
export AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o50ytwEXAMPLE=
```

Windows Command Prompt (CMD)

```
set AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE  
set AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY  
set AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o50ytwEXAMPLE=
```

PowerShell

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

```
$env:AWS_SESSION_TOKEN="AQoDYXdzEJr1K...o50ytwEXAMPLE="
```

- Utilisez les informations d'identification à long terme d'un utilisateur IAM :
 - a. Fournissez l'ID de clé d'accès et la clé d'accès secrète pour votre utilisateur IAM. Vous pouvez créer un utilisateur IAM pour le provisionnement, que vous supprimerez ultérieurement. Pour connaître la politique IAM à communiquer à l'utilisateur, consultez [Politique IAM minimale permettant au programme d'installation de provisionner les ressources](#). Pour plus d'informations sur la façon de récupérer des informations d'identification à long terme, consultez [la section Gestion des clés d'accès pour les utilisateurs IAM](#) dans le guide de l'utilisateur IAM.
 - b. Exécutez les commandes suivantes pour fournir les informations d'identification au logiciel AWS IoT Greengrass Core.

Linux or Unix

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

Windows Command Prompt (CMD)

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

PowerShell

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

- c. (Facultatif) Si vous avez créé un utilisateur IAM pour approvisionner votre appareil Greengrass, supprimez-le.
- d. (Facultatif) Si vous avez utilisé l'ID de clé d'accès et la clé d'accès secrète d'un utilisateur IAM existant, mettez à jour les clés de cet utilisateur afin qu'elles ne soient plus valides. Pour plus d'informations, consultez la section [Mise à jour des clés d'accès](#) dans le guide de Gestion des identités et des accès AWS l'utilisateur.

Téléchargez le logiciel AWS IoT Greengrass de base

Vous pouvez télécharger la dernière version du logiciel AWS IoT Greengrass Core à l'adresse suivante :

- <https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest.zip>

Note

Vous pouvez télécharger une version spécifique du logiciel AWS IoT Greengrass Core à l'emplacement suivant. Remplacez *version* par la version à télécharger.

```
https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-version.zip
```

Pour télécharger le logiciel AWS IoT Greengrass Core

1. Sur votre appareil principal, téléchargez le logiciel AWS IoT Greengrass Core dans un fichier nommé `greengrass-nucleus-latest.zip`.

Linux or Unix

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

Windows Command Prompt (CMD)

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

PowerShell

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

Si vous téléchargez ce logiciel, vous acceptez le [contrat de licence du logiciel Greengrass Core](#).

2. (Facultatif) Pour vérifier la signature du logiciel Greengrass Nucleus

 Note

Cette fonctionnalité est disponible avec Greengrass nucleus version 2.9.5 et versions ultérieures.

- a. Utilisez la commande suivante pour vérifier la signature de votre artefact Greengrass nucleus :

Linux or Unix

```
jarsigner -verify -certs -verbose greengrass-nucleus-latest.zip
```

Windows Command Prompt (CMD)

Le nom du fichier peut être différent selon la version du JDK que vous installez.
jdk17.0.6_10 Remplacez-le par la version du JDK que vous avez installée.

```
"C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe" -  
verify -certs -verbose greengrass-nucleus-latest.zip
```

PowerShell

Le nom du fichier peut être différent selon la version du JDK que vous installez.
jdk17.0.6_10 Remplacez-le par la version du JDK que vous avez installée.

```
'C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe' -  
verify -certs -verbose greengrass-nucleus-latest.zip
```

- b. L'`jarsigner` invocation produit une sortie qui indique les résultats de la vérification.
 - i. Si le fichier zip Greengrass nucleus est signé, le résultat contient l'instruction suivante :

```
jar verified.
```

- ii. Si le fichier zip Greengrass nucleus n'est pas signé, le résultat contient l'instruction suivante :

```
jar is unsigned.
```

- c. Si vous avez fourni l'-certsoption Jarsigner en même temps que les -verbose options -verify et, le résultat inclut également des informations détaillées sur le certificat du signataire.
3. Décompressez le logiciel AWS IoT Greengrass Core dans un dossier de votre appareil. *GreengrassInstaller* Remplacez-le par le dossier que vous souhaitez utiliser.

Linux or Unix

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

Windows Command Prompt (CMD)

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

PowerShell

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

4. (Facultatif) Exécutez la commande suivante pour voir la version du logiciel AWS IoT Greengrass Core.

```
java -jar ./ GreengrassInstaller/lib/Greengrass.jar --version
```

Important

Si vous installez une version du noyau Greengrass antérieure à la version 2.4.0, ne supprimez pas ce dossier après avoir installé le logiciel Core. AWS IoT Greengrass Le logiciel AWS IoT Greengrass Core utilise les fichiers de ce dossier pour s'exécuter.

Si vous avez téléchargé la dernière version du logiciel, vous devez installer la version 2.4.0 ou ultérieure, et vous pouvez supprimer ce dossier après avoir installé le logiciel AWS IoT Greengrass Core.

Installation du logiciel AWS IoT Greengrass de base

Exécutez le programme d'installation avec des arguments spécifiant les actions suivantes :

- Créez les AWS ressources dont le périphérique principal a besoin pour fonctionner.
- Spécifiez d'utiliser l'utilisateur `ggc_user` du système pour exécuter les composants logiciels sur le périphérique principal. Sur les appareils Linux, cette commande indique également d'utiliser le groupe `ggc_group` système, et le programme d'installation crée l'utilisateur et le groupe système pour vous.
- Configurez le logiciel AWS IoT Greengrass Core en tant que service système qui s'exécute au démarrage. Sur les appareils Linux, cela nécessite le [système d'initialisation Systemd](#).

Important

Sur les appareils Windows Core, vous devez configurer le logiciel AWS IoT Greengrass Core en tant que service système.

Pour configurer un dispositif de développement avec des outils de développement locaux, spécifiez `--deploy-dev-tools true` argument. Le déploiement des outils de développement local peut prendre jusqu'à une minute une fois l'installation terminée.

Pour plus d'informations sur les arguments que vous pouvez spécifier, consultez [Arguments d'installation](#).

Note

Si vous utilisez AWS IoT Greengrass un appareil dont la mémoire est limitée, vous pouvez contrôler la quantité de mémoire utilisée par le logiciel AWS IoT Greengrass Core. Pour contrôler l'allocation de mémoire, vous pouvez définir les options de taille de segment de mémoire JVM dans le paramètre de `jvmOptions` configuration de votre composant Nucleus.

Pour de plus amples informations, veuillez consulter [Contrôlez l'allocation de mémoire grâce aux options JVM](#).

Pour installer le logiciel AWS IoT Greengrass Core

1. Exécutez le programme d'installation AWS IoT Greengrass Core. Remplacez les valeurs des arguments dans votre commande comme suit.

 Note

Pour Windows, la limitation de la longueur du chemin est de 260 caractères. Si vous utilisez Windows, utilisez un dossier racine tel que C:\greengrass\v2 ou D:\greengrass\v2 pour conserver les chemins des composants Greengrass en dessous de la limite de 260 caractères.

- a. */greengrass/v2* ou *C:\greengrass\v2* : chemin d'accès au dossier racine à utiliser pour installer le logiciel AWS IoT Greengrass Core.
- b. *GreengrassInstaller*. Le chemin d'accès au dossier dans lequel vous avez décompressé le programme d'installation du logiciel AWS IoT Greengrass Core.
- c. *region*. L' Région AWS endroit dans lequel trouver ou créer des ressources.
- d. *MyGreengrassCore*. Le nom de l' AWS IoT appareil principal de votre Greengrass. Si l'objet n'existe pas, le programme d'installation le crée. Le programme d'installation télécharge les certificats pour s'authentifier en tant qu' AWS IoT objet. Pour de plus amples informations, veuillez consulter [Authentification et autorisation de l'appareil pour AWS IoT Greengrass](#).

 Note

Le nom de l'objet ne peut pas contenir de caractères deux-points (:).

- e. *MyGreengrassCoreGroup*. Le nom du AWS IoT groupe d'objets de votre appareil Greengrass principal. Si le groupe d'objets n'existe pas, le programme d'installation le crée et y ajoute l'objet. Si le groupe d'objets existe et fait l'objet d'un déploiement actif, le périphérique principal télécharge et exécute le logiciel spécifié par le déploiement.

 Note

Le nom du groupe d'objets ne peut pas contenir de deux-points (:).

- f. *GreengrassV2IoTThingPolicy*. Le nom de la AWS IoT politique qui permet aux appareils principaux de Greengrass de communiquer avec AWS IoT et. AWS IoT Greengrass Si la AWS IoT politique n'existe pas, le programme d'installation crée une AWS IoT politique permissive portant ce nom. Vous pouvez restreindre les autorisations de cette politique pour votre cas d'utilisation. Pour de plus amples informations, veuillez consulter [AWS IoT Politique minimale pour les appareils AWS IoT Greengrass V2 principaux](#).
- g. *GreengrassV2TokenExchangeRole*. Nom du rôle IAM qui permet au périphérique principal de Greengrass d'obtenir AWS des informations d'identification temporaires. Si le rôle n'existe pas, le programme d'installation le crée, puis crée et attache une politique nommée *GreengrassV2TokenExchangeRoleAccess*. Pour de plus amples informations, veuillez consulter [Autoriser les appareils principaux à interagir avec les AWS services](#).
- h. *GreengrassCoreTokenExchangeRoleAlias*. Alias du rôle IAM qui permet au périphérique principal de Greengrass d'obtenir des informations d'identification temporaires ultérieurement. Si l'alias de rôle n'existe pas, le programme d'installation le crée et le pointe vers le rôle IAM que vous spécifiez. Pour de plus amples informations, veuillez consulter [Autoriser les appareils principaux à interagir avec les AWS services](#).

Linux or Unix

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

Windows Command Prompt (CMD)

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

PowerShell

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

Important

Sur les appareils Windows Core, vous devez spécifier `--setup-system-service true` pour configurer le logiciel AWS IoT Greengrass Core en tant que service système.

Le programme d'installation affiche les messages suivants en cas de succès :

- Si vous le spécifiez `--provision`, le programme d'installation imprime `Successfully configured Nucleus with provisioned resource details` s'il a correctement configuré les ressources.

- Si vous le spécifiez `--deploy-dev-tools`, le programme d'installation affiche `Configured Nucleus to deploy aws.greengrass.Cli` component s'il a créé le déploiement avec succès.
 - Si vous le spécifiez `--setup-system-service true`, le programme d'installation affiche `Successfully set up Nucleus as a system service` s'il a configuré et exécuté le logiciel en tant que service.
 - Si vous ne le spécifiez pas `--setup-system-service true`, le programme d'installation affiche `Launched Nucleus successfully` s'il a réussi et a exécuté le logiciel.
2. Ignorez cette étape si vous avez installé la [Noyau de Greengrass](#) version 2.0.4 ou une version ultérieure. Si vous avez téléchargé la dernière version du logiciel, vous avez installé la version 2.0.4 ou ultérieure.

Exécutez la commande suivante pour définir les autorisations de fichier requises pour le dossier racine de votre logiciel AWS IoT Greengrass Core. Remplacez-le `/greengrass/v2` par le dossier racine que vous avez spécifié dans votre commande d'installation et remplacez-le `/greengrass` par le dossier parent de votre dossier racine.

```
sudo chmod 755 /greengrass/v2 && sudo chmod 755 /greengrass
```

Si vous avez installé le logiciel AWS IoT Greengrass Core en tant que service système, le programme d'installation exécute le logiciel pour vous. Dans le cas contraire, vous devez exécuter le logiciel manuellement. Pour de plus amples informations, veuillez consulter [Exécutez le logiciel AWS IoT Greengrass Core](#).

Note

Par défaut, le rôle IAM créé par le programme d'installation n'autorise pas l'accès aux artefacts des composants dans les compartiments S3. Pour déployer des composants personnalisés qui définissent des artefacts dans Amazon S3, vous devez ajouter des autorisations au rôle afin de permettre à votre appareil principal de récupérer les artefacts des composants. Pour de plus amples informations, veuillez consulter [Autoriser l'accès aux compartiments S3 pour les artefacts des composants](#).

Si vous ne possédez pas encore de compartiment S3 pour les artefacts des composants, vous pouvez ajouter ces autorisations ultérieurement après avoir créé un compartiment.

 Note

Lorsque le logiciel AWS IoT Greengrass Core se connecte au cloud, votre appareil est reconnu comme un appareil Core.

Pour plus d'informations sur la configuration et l'utilisation du logiciel AWS IoT Greengrass, consultez les rubriques suivantes :

- [Configuration du logiciel AWS IoT Greengrass de base](#)
- [Développer des AWS IoT Greengrass composants](#)
- [Déployer AWS IoT Greengrass des composants sur des appareils](#)
- [Interface de ligne de commande Greengrass](#)

Installation AWS IoT Greengrass du logiciel Core avec provisionnement manuel des ressources

Le logiciel AWS IoT Greengrass Core inclut un programme d'installation qui configure votre appareil en tant qu'appareil principal Greengrass. Pour configurer un appareil manuellement, vous pouvez créer les ressources requises AWS IoT et les ressources IAM que l'appareil doit utiliser. Si vous créez ces ressources manuellement, il n'est pas nécessaire de fournir des AWS informations d'identification au programme d'installation.

Lorsque vous installez manuellement le logiciel AWS IoT Greengrass Core, vous pouvez également configurer le périphérique pour qu'il utilise un proxy réseau ou qu'il se connecte AWS au port 443. Vous devrez peut-être spécifier ces options de configuration si votre appareil fonctionne derrière un pare-feu ou un proxy réseau, par exemple. Pour de plus amples informations, veuillez consulter [Connexion au port 443 ou via un proxy réseau](#).

Vous pouvez également configurer le logiciel AWS IoT Greengrass Core pour utiliser un module de sécurité matériel (HSM) via l'interface [PKCS #11](#). Cette fonctionnalité vous permet de stocker en toute sécurité les fichiers de clés privées et de certificats afin qu'ils ne soient pas exposés ou dupliqués dans le logiciel. Vous pouvez stocker des clés privées et des certificats sur un module matériel tel qu'un HSM, un module TPM (Trusted Platform Module) ou un autre élément cryptographique. Cette fonctionnalité n'est disponible que sur les appareils Linux. Pour plus d'informations sur la sécurité matérielle et les conditions requises pour l'utiliser, consultez [Intégration de sécurité matérielle](#).

⚠ Important

Avant de télécharger le logiciel AWS IoT Greengrass Core, vérifiez que votre appareil principal répond à la [configuration requise](#) pour installer et exécuter le logiciel AWS IoT Greengrass Core v2.0.

Rubriques

- [Récupérer des points de AWS IoT terminaison](#)
- [Créez n'importe AWS IoT quoi](#)
- [Créez le certificat d'objet](#)
- [Configurer le certificat d'objet](#)
- [Création d'un rôle d'échange de jetons](#)
- [Télécharger les certificats sur l'appareil](#)
- [Configuration de l'environnement de l'appareil](#)
- [Téléchargez le logiciel AWS IoT Greengrass de base](#)
- [Installation du logiciel AWS IoT Greengrass Core](#)

Récupérer des points de AWS IoT terminaison

Obtenez les AWS IoT points de terminaison qui vous Compte AWS conviennent et enregistrez-les pour les utiliser ultérieurement. Votre appareil utilise ces points de terminaison pour se connecter à AWS IoT. Procédez comme suit :

1. Obtenez le point de terminaison de AWS IoT données pour votre Compte AWS.

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

La réponse ressemble à l'exemple suivant, si la demande aboutit.

```
{
  "endpointAddress": "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
}
```

2. Obtenez le point de terminaison des informations d' AWS IoT identification pour votre Compte AWS.

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

La réponse ressemble à l'exemple suivant, si la demande aboutit.

```
{
  "endpointAddress": "device-credentials-prefix.credentials.iot.us-
west-2.amazonaws.com"
}
```

Créez n'importe AWS IoT quoi

AWS IoT les objets représentent les appareils et les entités logiques auxquels ils se connectent AWS IoT. Les appareils Greengrass Core sont AWS IoT des objets. Lorsque vous enregistrez un appareil en tant qu' AWS IoT objet, celui-ci peut utiliser un certificat numérique pour s'authentifier. AWS

Dans cette section, vous allez créer un AWS IoT objet qui représente votre appareil.

Pour créer AWS IoT quelque chose

1. Créez n'importe AWS IoT quel objet pour votre appareil. Sur votre ordinateur de développement, exécutez la commande suivante.
 - Remplacez *MyGreengrassCore* par le nom de l'objet à utiliser. Ce nom est également le nom de votre appareil principal Greengrass.

Note

Le nom de l'objet ne peut pas contenir de caractères deux-points (:).

```
aws iot create-thing --thing-name MyGreengrassCore
```

La réponse ressemble à l'exemple suivant, si la demande aboutit.

```
{
  "thingName": "MyGreengrassCore",
  "thingArn": "arn:aws:iot:us-west-2:123456789012:thing/MyGreengrassCore",
  "thingId": "8cb4b6cd-268e-495d-b5b9-1713d71dbf42"
```

```
}
```

2. (Facultatif) Ajoutez l' AWS IoT objet à un nouveau groupe d'objets ou à un groupe d'objets existant. Vous utilisez des groupes d'objets pour gérer des flottes d'appareils principaux de Greengrass. Lorsque vous déployez des composants logiciels sur vos appareils, vous pouvez cibler des appareils individuels ou des groupes d'appareils. Vous pouvez ajouter un appareil à un groupe d'objets avec un déploiement Greengrass actif pour déployer les composants logiciels de ce groupe d'objets sur l'appareil. Procédez comme suit :

- a. (Facultatif) Créez un AWS IoT groupe d'objets.

- Remplacez *MyGreengrassCoreGroup* par le nom du groupe d'objets à créer.

 Note

Le nom du groupe d'objets ne peut pas contenir de deux-points (:).

```
aws iot create-thing-group --thing-group-name MyGreengrassCoreGroup
```

La réponse ressemble à l'exemple suivant, si la demande aboutit.

```
{
  "thingGroupName": "MyGreengrassCoreGroup",
  "thingGroupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/
MyGreengrassCoreGroup",
  "thingGroupId": "4df721e1-ff9f-4f97-92dd-02db4e3f03aa"
}
```

- b. Ajoutez l' AWS IoT objet à un groupe d'objets.

- Remplacez *MyGreengrassCore* par le nom de votre AWS IoT objet.
- Remplacez *MyGreengrassCoreGroup* par le nom du groupe d'objets.

```
aws iot add-thing-to-thing-group --thing-name MyGreengrassCore --thing-group-
name MyGreengrassCoreGroup
```

La commande n'a aucune sortie si la demande aboutit.

Créez le certificat d'objet

Lorsque vous enregistrez un appareil en tant qu' AWS IoT objet, celui-ci peut utiliser un certificat numérique pour s'authentifier. AWS Ce certificat permet à l'appareil de communiquer avec AWS IoT et AWS IoT Greengrass.

Dans cette section, vous allez créer et télécharger des certificats que votre appareil peut utiliser pour se connecter AWS.

Si vous souhaitez configurer le logiciel AWS IoT Greengrass Core pour utiliser un module de sécurité matériel (HSM) pour stocker en toute sécurité la clé privée et le certificat, suivez les étapes pour créer le certificat à partir d'une clé privée dans un HSM. Sinon, suivez les étapes pour créer le certificat et la clé privée dans le AWS IoT service. La fonctionnalité de sécurité matérielle n'est disponible que sur les appareils Linux. Pour plus d'informations sur la sécurité matérielle et les conditions requises pour l'utiliser, consultez [Intégration de sécurité matérielle](#).

Créez le certificat et la clé privée dans le AWS IoT service

Pour créer le certificat d'objet

1. Créez un dossier dans lequel vous téléchargerez les certificats de l' AWS IoT objet.

```
mkdir greengrass-v2-certs
```

2. Créez et téléchargez les certificats AWS IoT correspondants.

```
aws iot create-keys-and-certificate --set-as-active --certificate-pem-outfile  
greengrass-v2-certs/device.pem.crt --public-key-outfile greengrass-v2-certs/  
public.pem.key --private-key-outfile greengrass-v2-certs/private.pem.key
```

La réponse ressemble à l'exemple suivant, si la demande aboutit.

```
{  
  "certificateArn": "arn:aws:iot:us-west-2:123456789012:cert/  
aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4",  
  "certificateId":  
  "aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4",  
  "certificatePem": "-----BEGIN CERTIFICATE-----  
MIICiTCCAfICCCQD6m7oRw0uX0jANBgkqhkiG9w  
0BAQUFADCBiDELMAkGA1UEBhMCVVMxCzAJBgNVBAgTAldBMRAwDgYDVQQHEwdTZ  
WF0dGx1MQ8wDQYDVQQKEwZBbWF6b24xZDASBgNVBASTC0lBTSBDb25zb2x1MRIw
```

```

EAYDVQQDEw1UZsXN0Q21sYWMxHZAAdBgkqhkiG9w0BCQEWEG5vb251QGFTYXpvi5
jb20wHhcNMTEwNDI1MjA0NTIxWhcNMTEwNDI1MjA0NTIxWjCBiDELMakGA1UEBh
MCVVMxCzAJBgNVBAGTAldBMRAwDgYDVQQHEwdTZWF0dGx1MQ8wDQYDVQQKEwZBb
WF6b24xFDASBgNVBASTC0lBTSBDb25zb2x1MRIwEAYDVQQDEw1UZsXN0Q21sYWMx
HZAAdBgkqhkiG9w0BCQEWEG5vb251QGFTYXpvi5jb20wgZ8wDQYJKoZIhvcNAQEE
BBQADgY0AMIGJAoGBAMaK0dn+a4GmWIWJ21uUSfwfEvySWtC2XADZ4nB+BLYgVI
k60CpiwsZ3G93vUEI03IyNoH/f0wYK8m9TrDHudUZg3qX4waLG5M43q7Wgc/MbQ
ITx0USQv7c7ugFFDzQGBZsWY6786m86gpEIbb30hjZnzcVQAaRHhd1QWIMm2nr
AgMBAAEwDQYJKoZIhvcNAQEFBQADgYEAtCu4nUhVVxYUntneD9+h8Mg9q6q+auN
KyExzyLwax1Aoo7TJHidbtS4J5iNmZgXL0FkbFFBjvSfpJI1J00zbhNYS5f6Guo
EDmFJl0ZxBHjJnyp3780D8uTs7fLvJx79LjSTbNYiytVbZPQUQ5Yaxu2jXnimvw
3rrszlaEXAMPLE=
-----END CERTIFICATE-----",
  "keyPair": {
    "PublicKey": "-----BEGIN PUBLIC KEY-----\
MIIBIjANBgkqhkiG9w0BCQEFAAQCAQ8AMIIBCgKCAQEAEXAMPLE1nnyJwKSMHw4h\
MMEXAMPLEUuuN/dMAS3fyce8DW/4+EXAMPLEYjmoF/YVF/gHr99VEEXAMPLE5VF13\
59VK7cEXAMPLE67GK+y+jikqX0gHh/xJTtwo
+sGpWEXAMPLEDz18x0d2ka4tCzuWEXAMPLEahJbYkCPUBSU8opVkr7qkEXAMPLE1DR6sx2Hocli00Ltu6Fkw91swQWE
\GB3ZPrNh0PzQYvjUSTZeccyNCx2EXAMPLEvp9mQ0UXP6plfgxwKRX2fEXAMPLEda\
hJLXkX3rHU2xbxJSq7D+XEXAMPLEcw+LyFhI5mgFR188eGdsAEXAMPLE1nI9EesG\
FQIDAQAB\
-----END PUBLIC KEY-----\
",
    "PrivateKey": "-----BEGIN RSA PRIVATE KEY-----\
key omitted for security reasons\
-----END RSA PRIVATE KEY-----\
"
  }
}

```

Enregistrez le nom de ressource Amazon (ARN) du certificat afin de l'utiliser pour configurer le certificat ultérieurement.

Créez le certificat à partir d'une clé privée dans un HSM

Note

Cette fonctionnalité est disponible pour les versions 2.5.3 et ultérieures du composant [Greengrass](#) nucleus. AWS IoT Greengrass ne prend actuellement pas en charge cette fonctionnalité sur les appareils Windows principaux.

Pour créer le certificat d'objet

1. Sur le périphérique principal, initialisez un jeton PKCS #11 dans le HSM et générez une clé privée. La clé privée doit être une clé RSA de taille RSA-2048 (ou supérieure) ou une clé ECC.

Note

Pour utiliser un module de sécurité matériel avec des clés ECC, vous devez utiliser [Greengrass](#) nucleus v2.5.6 ou version ultérieure.

Pour utiliser un module de sécurité matériel et un [gestionnaire de secrets](#), vous devez utiliser un module de sécurité matériel avec des clés RSA.

Consultez la documentation de votre HSM pour savoir comment initialiser le jeton et générer la clé privée. Si votre HSM prend en charge l'objet IDs, spécifiez un ID d'objet lorsque vous générez la clé privée. Enregistrez l'ID d'emplacement, le code PIN utilisateur, le libellé de l'objet, l'ID d'objet (si votre HSM en utilise un) que vous spécifiez lorsque vous initialisez le jeton et générez la clé privée. Vous utiliserez ces valeurs ultérieurement lorsque vous importez le certificat d'objet dans le HSM et que vous configurez le logiciel AWS IoT Greengrass principal.

2. Créez une demande de signature de certificat (CSR) à partir de la clé privée. AWS IoT utilise ce CSR pour créer un certificat d'objet pour la clé privée que vous avez générée dans le HSM. Pour plus d'informations sur la création d'un CSR à partir de la clé privée, consultez la documentation de votre HSM. Le CSR est un fichier, tel que `iotdevicekey.csr`.
3. Copiez le CSR de l'appareil vers votre ordinateur de développement. Si SSH et SCP sont activés sur l'ordinateur de développement et le périphérique, vous pouvez utiliser la `scp` commande de votre ordinateur de développement pour transférer le CSR. Remplacez *device-ip-address* par l'adresse IP de votre appareil, puis `~/iotdevicekey.csr` par le chemin d'accès au fichier CSR sur l'appareil.

```
scp device-ip-address:~/iotdevicekey.csr iotdevicekey.csr
```

4. Sur votre ordinateur de développement, créez un dossier dans lequel vous téléchargerez le certificat AWS IoT correspondant.

```
mkdir greengrass-v2-certs
```

5. Utilisez le fichier CSR pour créer et télécharger le certificat correspondant à l'AWS IoT objet sur votre ordinateur de développement.

```
aws iot create-certificate-from-csr --set-as-active --certificate-signing-
request=file:///iotdevicekey.csr --certificate-pem-outfile greengrass-v2-certs/
device.pem.crt
```

La réponse ressemble à l'exemple suivant, si la demande aboutit.

```
{
  "certificateArn": "arn:aws:iot:us-west-2:123456789012:cert/
aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4",
  "certificateId":
"aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4",
  "certificatePem": "-----BEGIN CERTIFICATE-----
MIICiTCCAfICCQD6m7oRw0uX0jANBgkqhkiG9w
0BAQUFADCBiDELMAkGA1UEBhMCVVMxCzAJBgNVBAGTAldBMRAwDgYDVQQHEwdTZ
WF0dGx1MQ8wDQYDVQQKEwZBbWF6b24xFDASBgNVBASTC01BTSBDb25zb2x1MRIw
EAYDVQQDEw1UZXR0Q21sYW1uXzAdBgkqhkiG9w0BCQEWEG5vb25lQGFTYXpvi5
jb20wHhcNMTEwNDI1MjA0NTIxWhcNMTEwNDI1MjA0NTIxWjCBiDELMAkGA1UEBh
MCVVMxCzAJBgNVBAGTAldBMRAwDgYDVQQHEwdTZWF0dGx1MQ8wDQYDVQQKEwZBb
WF6b24xFDASBgNVBASTC01BTSBDb25zb2x1MRIwEAYDVQQDEw1UZXR0Q21sYW1u
HzAdBgkqhkiG9w0BCQEWEG5vb25lQGFTYXpvi5jb20wgZ8wDQYJKoZIhvcNAQEE
BBQADgY0AMIGJAoGBAMaK0dn+a4GmWIWJ21uUSfwfEvySWtC2XADZ4nB+BLyGVI
k60CpiwsZ3G93vUEI03IyNoH/f0wYK8m9TrDHudUZg3qX4waLG5M43q7Wgc/MbQ
ITx0USQv7c7ugFFDzQGBzZswY6786m86gpEIbb30hjZnzcVQAaRHhd1QWIMm2nr
AgMBAAEwDQYJKoZIhvcNAQEFBQADgYEAtCu4nUhVVxYUntneD9+h8Mg9q6q+auN
KyExzyLwax1Aoo7TJHidbtS4J5iNmZgXL0FkbFFBjvSfpJI1J00zbhNYS5f6Guo
EDmFJl0ZxBHjJnyp3780D8uTs7fLvJx79LjSTbNYiytVbZPQUQ5Yaxu2jXnimvw
3rrszlaEXAMPLE=
-----END CERTIFICATE-----"
}
```

Enregistrez l'ARN du certificat afin de l'utiliser pour le configurer ultérieurement.

Configurer le certificat d'objet

Attachez le certificat d' AWS IoT objet à l'objet que vous avez créé précédemment et ajoutez une AWS IoT politique au certificat afin de définir les AWS IoT autorisations pour le périphérique principal.

Pour configurer le certificat de l'objet

1. Joignez le certificat à l' AWS IoT objet.

- Remplacez *MyGreengrassCore* par le nom de votre AWS IoT objet.
- Remplacez le certificat Amazon Resource Name (ARN) par l'ARN du certificat que vous avez créé à l'étape précédente.

```
aws iot attach-thing-principal --thing-name MyGreengrassCore
--principal arn:aws:iot:us-west-2:123456789012:cert/
aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4
```

La commande n'a aucune sortie si la demande aboutit.

2. Créez et joignez une AWS IoT politique qui définit les AWS IoT autorisations pour votre appareil principal Greengrass. La politique suivante permet d'accéder à tous les sujets MQTT et aux opérations Greengrass, afin que votre appareil fonctionne avec les applications personnalisées et les modifications futures qui nécessitent de nouvelles opérations Greengrass. Vous pouvez restreindre cette politique en fonction de votre cas d'utilisation. Pour de plus amples informations, veuillez consulter [AWS IoT Politique minimale pour les appareils AWS IoT Greengrass V2 principaux](#).

Si vous avez déjà configuré un appareil principal Greengrass, vous pouvez joindre sa AWS IoT politique au lieu d'en créer une nouvelle.

Procédez comme suit :

- a. Créez un fichier contenant le document de AWS IoT politique dont les appareils principaux de Greengrass ont besoin.

Par exemple, sur un système basé sur Linux, vous pouvez exécuter la commande suivante pour utiliser GNU nano pour créer le fichier.

```
nano greengrass-v2-iot-policy.json
```

Copiez le code JSON suivant dans le fichier.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
```



```

    "Action": [
      "iot:Publish",
      "iot:Subscribe",
      "iot:Receive",
      "iot:Connect",
      "greengrass:*"
    ],
    "Resource": [
      "*"
    ]
  }
]
}

```

b. Créez une AWS IoT politique à partir du document de stratégie.

- Remplacez *GreengrassV2IoTThingPolicy* par le nom de la politique à créer.

```

aws iot create-policy --policy-name GreengrassV2IoTThingPolicy --policy-
document file://greengrass-v2-iot-policy.json

```

La réponse ressemble à l'exemple suivant, si la demande aboutit.

```

{
  "policyName": "GreengrassV2IoTThingPolicy",
  "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassV2IoTThingPolicy",
  "policyDocument": "{
    \\\"Version\\\": \\\"2012-10-17          \\\",
    \\\"Statement\\\": [
      {
        \\\"Effect\\\": \\\"Allow\\\",
        \\\"Action\\\": [
          \\\"iot:Publish\\\",
          \\\"iot:Subscribe\\\",
          \\\"iot:Receive\\\",
          \\\"iot:Connect\\\",
          \\\"greengrass:*\\\"
        ],
        \\\"Resource\\\": [
          \\\"*\\\"
        ]
      }
    ]
  }"
}

```

```
    }  
  ]  
  },  
  "policyVersionId": "1"  
}
```

c. Joignez la AWS IoT politique au certificat de l' AWS IoT objet.

- Remplacez *GreengrassV2IoTThingPolicy* par le nom de la politique à joindre.
- Remplacez l'ARN cible par l'ARN du certificat associé à votre AWS IoT objet.

```
aws iot attach-policy --policy-name GreengrassV2IoTThingPolicy  
--target arn:aws:iot:us-west-2:123456789012:cert/  
aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4
```

La commande n'a aucune sortie si la demande aboutit.

Création d'un rôle d'échange de jetons

Les appareils principaux de Greengrass utilisent un rôle de service IAM, appelé rôle d'échange de jetons, pour autoriser les appels aux services. AWS L'appareil utilise le fournisseur AWS IoT d'informations d'identification pour obtenir des AWS informations d'identification temporaires pour ce rôle, ce qui lui permet d'interagir avec Amazon Logs AWS IoT, d'envoyer des journaux à Amazon CloudWatch Logs et de télécharger des artefacts de composants personnalisés depuis Amazon S3. Pour de plus amples informations, veuillez consulter [Autoriser les appareils principaux à interagir avec les AWS services](#).

Vous utilisez un alias de AWS IoT rôle pour configurer le rôle d'échange de jetons pour les appareils principaux de Greengrass. Les alias de rôle vous permettent de modifier le rôle d'échange de jetons d'un appareil tout en conservant la même configuration de l'appareil. Pour plus d'informations, consultez la section [Autorisation des appels directs vers AWS des services](#) dans le Guide du AWS IoT Core développeur.

Dans cette section, vous allez créer un rôle IAM d'échange de jetons et un alias de AWS IoT rôle pointant vers le rôle. Si vous avez déjà configuré un appareil principal Greengrass, vous pouvez utiliser son rôle d'échange de jetons et son alias de rôle au lieu d'en créer de nouveaux. Ensuite, vous configurez l'appareil pour AWS IoT qu'il utilise ce rôle et cet alias.

Pour créer un rôle IAM d'échange de jetons

1. Créez un rôle IAM que votre appareil peut utiliser comme rôle d'échange de jetons. Procédez comme suit :
 - a. Créez un fichier contenant le document de politique de confiance requis par le rôle d'échange de jetons.

Par exemple, sur un système basé sur Linux, vous pouvez exécuter la commande suivante pour utiliser GNU nano pour créer le fichier.

```
nano device-role-trust-policy.json
```

Copiez le code JSON suivant dans le fichier.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "credentials.iot.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

- b. Créez le rôle d'échange de jetons avec le document de politique de confiance.
 - Remplacez *GreengrassV2TokenExchangeRole* par le nom du rôle IAM à créer.

```
aws iam create-role --role-name GreengrassV2TokenExchangeRole --assume-role-policy-document file://device-role-trust-policy.json
```

La réponse ressemble à l'exemple suivant, si la demande aboutit.

```
{
  "Role": {
    "Path": "/",
```

```
"RoleName": "GreengrassV2TokenExchangeRole",
"RoleId": "AR0AZ2YMUHYHK50KM77FB",
"Arn": "arn:aws:iam::123456789012:role/GreengrassV2TokenExchangeRole",
"CreateDate": "2021-02-06T00:13:29+00:00",
"AssumeRolePolicyDocument": {
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "credentials.iot.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

- c. Créez un fichier contenant le document de politique d'accès requis par le rôle d'échange de jetons.

Par exemple, sur un système basé sur Linux, vous pouvez exécuter la commande suivante pour utiliser GNU nano pour créer le fichier.

```
nano device-role-access-policy.json
```

Copiez le code JSON suivant dans le fichier.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams",
        "s3:GetBucketLocation"
      ],
      "Resource": "*"
    }
  ]
}
```

```
]
}
```

 Note

Cette politique d'accès n'autorise pas l'accès aux artefacts des composants dans les compartiments S3. Pour déployer des composants personnalisés qui définissent des artefacts dans Amazon S3, vous devez ajouter des autorisations au rôle afin de permettre à votre appareil principal de récupérer les artefacts des composants. Pour de plus amples informations, veuillez consulter [Autoriser l'accès aux compartiments S3 pour les artefacts des composants](#).

Si vous ne possédez pas encore de compartiment S3 pour les artefacts des composants, vous pouvez ajouter ces autorisations ultérieurement après avoir créé un compartiment.

- d. Créez la politique IAM à partir du document de stratégie.
- Remplacez *GreengrassV2TokenExchangeRoleAccess* par le nom de la politique IAM à créer.

```
aws iam create-policy --policy-name GreengrassV2TokenExchangeRoleAccess --  
policy-document file://device-role-access-policy.json
```

La réponse ressemble à l'exemple suivant, si la demande aboutit.

```
{  
  "Policy": {  
    "PolicyName": "GreengrassV2TokenExchangeRoleAccess",  
    "PolicyId": "ANPAZ2YMUHYHACI7C5Z66",  
    "Arn": "arn:aws:iam::123456789012:policy/  
GreengrassV2TokenExchangeRoleAccess",  
    "Path": "/",  
    "DefaultVersionId": "v1",  
    "AttachmentCount": 0,  
    "PermissionsBoundaryUsageCount": 0,  
    "IsAttachable": true,  
    "CreateDate": "2021-02-06T00:37:17+00:00",  
    "UpdateDate": "2021-02-06T00:37:17+00:00"  
  }  
}
```

```
}
```

e. Associez la politique IAM au rôle d'échange de jetons.

- Remplacez *GreengrassV2TokenExchangeRole* par le nom du rôle IAM.
- Remplacez l'ARN de la stratégie par l'ARN de la stratégie IAM que vous avez créée à l'étape précédente.

```
aws iam attach-role-policy --role-name GreengrassV2TokenExchangeRole --policy-arn arn:aws:iam::123456789012:policy/GreengrassV2TokenExchangeRoleAccess
```

La commande n'a aucune sortie si la demande aboutit.

2. Créez un alias de AWS IoT rôle qui pointe vers le rôle d'échange de jetons.

- Remplacez *GreengrassCoreTokenExchangeRoleAlias* par le nom de l'alias de rôle à créer.
- Remplacez l'ARN du rôle par l'ARN du rôle IAM que vous avez créé à l'étape précédente.

```
aws iot create-role-alias --role-alias GreengrassCoreTokenExchangeRoleAlias --role-arn arn:aws:iam::123456789012:role/GreengrassV2TokenExchangeRole
```

La réponse ressemble à l'exemple suivant, si la demande aboutit.

```
{
  "roleAlias": "GreengrassCoreTokenExchangeRoleAlias",
  "roleAliasArn": "arn:aws:iot:us-west-2:123456789012:rolealias/GreengrassCoreTokenExchangeRoleAlias"
}
```

Note

Pour créer un alias de rôle, vous devez être autorisé à transmettre le rôle IAM d'échange de jetons à AWS IoT. Si vous recevez un message d'erreur lorsque vous essayez de créer un alias de rôle, vérifiez que votre AWS utilisateur dispose de cette autorisation. Pour plus d'informations, consultez la section [Octroi à un utilisateur des autorisations](#)

[lui permettant de transférer un rôle à un AWS service](#) dans le Guide de Gestion des identités et des accès AWS l'utilisateur.

3. Créez et attachez une AWS IoT politique qui permet à votre appareil principal Greengrass d'utiliser l'alias de rôle pour assumer le rôle d'échange de jetons. Si vous avez déjà configuré un appareil principal Greengrass, vous pouvez associer sa AWS IoT politique d'alias de rôle au lieu d'en créer une nouvelle. Procédez comme suit :
 - a. (Facultatif) Créez un fichier contenant le document AWS IoT de politique requis par l'alias de rôle.

Par exemple, sur un système basé sur Linux, vous pouvez exécuter la commande suivante pour utiliser GNU nano pour créer le fichier.

```
nano greengrass-v2-iot-role-alias-policy.json
```

Copiez le code JSON suivant dans le fichier.

- Remplacez l'ARN de la ressource par l'ARN de votre alias de rôle.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iot:AssumeRoleWithCertificate",
      "Resource": "arn:aws:iot:us-west-2:123456789012:rolealias/  
GreengrassCoreTokenExchangeRoleAlias"
    }
  ]
}
```

- b. Créez une AWS IoT politique à partir du document de stratégie.
 - Remplacez *GreengrassCoreTokenExchangeRoleAliasPolicy* par le nom de la AWS IoT politique à créer.

```
aws iot create-policy --policy-name GreengrassCoreTokenExchangeRoleAliasPolicy
--policy-document file://greengrass-v2-iot-role-alias-policy.json
```

La réponse ressemble à l'exemple suivant, si la demande aboutit.

```
{
  "policyName": "GreengrassCoreTokenExchangeRoleAliasPolicy",
  "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassCoreTokenExchangeRoleAliasPolicy",
  "policyDocument": "{
    \\\"Version\\\": \\\"2012-10-17          \\\",
    \\\"Statement\\\": [
      {
        \\\"Effect\\\": \\\"Allow\\\",
        \\\"Action\\\": \\\"iot:AssumeRoleWithCertificate\\\",
        \\\"Resource\\\": \\\"arn:aws:iot:us-west-2:123456789012:rolealias/
GreengrassCoreTokenExchangeRoleAlias\\\"
      }
    ]
  }",
  "policyVersionId": "1"
}
```

c. Joignez la AWS IoT politique au certificat de l' AWS IoT objet.

- Remplacez *GreengrassCoreTokenExchangeRoleAliasPolicy* par le nom de la AWS IoT politique d'alias de rôle.
- Remplacez l'ARN cible par l'ARN du certificat associé à votre AWS IoT objet.

```
aws iot attach-policy --policy-name GreengrassCoreTokenExchangeRoleAliasPolicy
--target arn:aws:iot:us-west-2:123456789012:cert/
aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4
```

La commande n'a aucune sortie si la demande aboutit.

Télécharger les certificats sur l'appareil

Auparavant, vous avez téléchargé le certificat de votre appareil sur votre ordinateur de développement. Dans cette section, vous allez copier le certificat sur votre appareil principal pour configurer celui-ci avec les certificats auxquels il se connecte AWS IoT. Vous téléchargez également le certificat de l'autorité de certification racine (CA) Amazon. Si vous utilisez un HSM, vous importez également le fichier de certificat dans le HSM dans cette section.

- Si vous avez créé le certificat d'objet et la clé privée dans le AWS IoT service précédemment, suivez les étapes pour télécharger les certificats avec la clé privée et les fichiers de certificat.
- Si vous avez créé le certificat d'objet à partir d'une clé privée dans un module de sécurité matériel (HSM) plus tôt, suivez les étapes pour télécharger les certificats avec la clé privée et le certificat dans un HSM.

Télécharger des certificats avec clé privée et fichiers de certificats

Pour télécharger des certificats sur l'appareil

1. Copiez le certificat d' AWS IoT objet de votre ordinateur de développement sur l'appareil. Si SSH et SCP sont activés sur l'ordinateur de développement et le périphérique, vous pouvez utiliser la scp commande de votre ordinateur de développement pour transférer le certificat. Remplacez *device-ip-address* par l'adresse IP de votre appareil.

```
scp -r greengrass-v2-certs/ device-ip-address:~
```

2. Créez le dossier racine Greengrass sur l'appareil. Vous installerez ultérieurement le logiciel AWS IoT Greengrass Core dans ce dossier.

Note

Pour Windows, la limitation de la longueur du chemin est de 260 caractères. Si vous utilisez Windows, utilisez un dossier racine tel que C:\greengrass\v2 ou D:\greengrass\v2 pour conserver les chemins des composants Greengrass en dessous de la limite de 260 caractères.

Linux or Unix

- `/greengrass/v2` Remplacez-le par le dossier à utiliser.

```
sudo mkdir -p /greengrass/v2
```

Windows Command Prompt

- `C:\greengrass\v2` Remplacez-le par le dossier à utiliser.

```
mkdir C:\greengrass\v2
```

PowerShell

- `C:\greengrass\v2` Remplacez-le par le dossier à utiliser.

```
mkdir C:\greengrass\v2
```

3. (Linux uniquement) Définissez les autorisations du parent du dossier racine de Greengrass.

- Remplacez `/greengrass` par le parent du dossier racine.

```
sudo chmod 755 /greengrass
```

4. Copiez les AWS IoT certificats d'objets dans le dossier racine de Greengrass.

Linux or Unix

- `/greengrass/v2` Remplacez-le par le dossier racine de Greengrass.

```
sudo cp -R ~/greengrass-v2-certs/* /greengrass/v2
```

Windows Command Prompt

- `C:\greengrass\v2` Remplacez-le par le dossier à utiliser.

```
robocopy %USERPROFILE%\greengrass-v2-certs C:\greengrass\v2 /E
```

PowerShell

- `C:\greengrass\v2` Remplacez-le par le dossier à utiliser.

```
cp -Path ~\greengrass-v2-certs\* -Destination C:\greengrass\v2
```

5. Téléchargez le certificat de l'autorité de certification racine (CA) Amazon. AWS IoT les certificats sont associés par défaut au certificat de l'autorité de certification racine d'Amazon.

Linux or Unix

```
sudo curl -o /greengrass/v2/AmazonRootCA1.pem https://www.amazontrust.com/repository/AmazonRootCA1.pem
```

Windows Command Prompt (CMD)

```
curl -o C:\greengrass\v2\AmazonRootCA1.pem https://www.amazontrust.com/repository/AmazonRootCA1.pem
```

PowerShell

```
iwr -Uri https://www.amazontrust.com/repository/AmazonRootCA1.pem -OutFile C:\greengrass\v2\AmazonRootCA1.pem
```

Téléchargez les certificats avec la clé privée et le certificat dans un HSM

Note

Cette fonctionnalité est disponible pour les versions 2.5.3 et ultérieures du composant [Greengrass](#) nucleus. AWS IoT Greengrass ne prend actuellement pas en charge cette fonctionnalité sur les appareils Windows principaux.

Pour télécharger des certificats sur l'appareil

1. Copiez le certificat d' AWS IoT objet de votre ordinateur de développement sur l'appareil. Si SSH et SCP sont activés sur l'ordinateur de développement et le périphérique, vous pouvez utiliser la scp commande de votre ordinateur de développement pour transférer le certificat. Remplacez *device-ip-address* par l'adresse IP de votre appareil.

```
scp -r greengrass-v2-certs/ device-ip-address:~
```

2. Créez le dossier racine Greengrass sur l'appareil. Vous installerez ultérieurement le logiciel AWS IoT Greengrass Core dans ce dossier.

Note

Pour Windows, la limitation de la longueur du chemin est de 260 caractères. Si vous utilisez Windows, utilisez un dossier racine tel que C:\greengrass\v2 ou D:\greengrass\v2 pour conserver les chemins des composants Greengrass en dessous de la limite de 260 caractères.

Linux or Unix

- */greengrass/v2* Remplacez-le par le dossier à utiliser.

```
sudo mkdir -p /greengrass/v2
```

Windows Command Prompt

- *C:\greengrass\v2* Remplacez-le par le dossier à utiliser.

```
mkdir C:\greengrass\v2
```

PowerShell

- *C:\greengrass\v2* Remplacez-le par le dossier à utiliser.

```
mkdir C:\greengrass\v2
```

3. (Linux uniquement) Définissez les autorisations du parent du dossier racine de Greengrass.

- Remplacez */greengrass* par le parent du dossier racine.

```
sudo chmod 755 /greengrass
```

4. Importez le fichier de certificat d'~/greengrass-v2-certs/device.pem.crt objet dans le HSM. Consultez la documentation de votre HSM pour savoir comment y importer des certificats. Importez le certificat en utilisant le jeton, l'ID de slot, le code PIN utilisateur, le même libellé d'objet et le même ID d'objet (si votre HSM en utilise un) où vous avez généré la clé privée dans le HSM précédemment.

 Note

Si vous avez généré la clé privée plus tôt sans ID d'objet et que le certificat possède un ID d'objet, définissez l'ID d'objet de la clé privée sur la même valeur que le certificat. Consultez la documentation de votre HSM pour savoir comment définir l'ID d'objet pour l'objet clé privée.

5. (Facultatif) Supprimez le fichier de certificat d'objet afin qu'il n'existe que dans le HSM.

```
rm ~/greengrass-v2-certs/device.pem.crt
```

6. Téléchargez le certificat de l'autorité de certification racine (CA) Amazon. AWS IoT les certificats sont associés par défaut au certificat de l'autorité de certification racine d'Amazon.

Linux or Unix

```
sudo curl -o /greengrass/v2/AmazonRootCA1.pem https://www.amazontrust.com/repository/AmazonRootCA1.pem
```

Windows Command Prompt (CMD)

```
curl -o C:\greengrass\v2\AmazonRootCA1.pem https://www.amazontrust.com/repository/AmazonRootCA1.pem
```

PowerShell

```
iwr -Uri https://www.amazontrust.com/repository/AmazonRootCA1.pem -OutFile C:  
:\greengrass\v2\AmazonRootCA1.pem
```

Configuration de l'environnement de l'appareil

Suivez les étapes décrites dans cette section pour configurer un appareil Linux ou Windows à utiliser comme périphérique AWS IoT Greengrass principal.

Configuration d'un appareil Linux

Pour configurer un appareil Linux pour AWS IoT Greengrass V2

1. Installez le moteur d'exécution Java, dont le logiciel AWS IoT Greengrass Core a besoin pour fonctionner. Nous vous recommandons d'utiliser les versions de support à long terme d'[Amazon Corretto](#) ou d'OpenJDK. La version 8 ou supérieure est requise. Les commandes suivantes vous montrent comment installer OpenJDK sur votre appareil.

- Pour les distributions basées sur Debian ou Ubuntu :

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

- Pour les distributions basées sur Red Hat :

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

- Dans Amazon Linux 2 :

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

- Pour Amazon Linux 2023 :

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

Lorsque l'installation est terminée, exécutez la commande suivante pour vérifier que Java s'exécute sur votre appareil Linux.

```
java -version
```

La commande affiche la version de Java exécutée sur le périphérique. Par exemple, sur une distribution basée sur Debian, le résultat peut ressembler à celui de l'exemple suivant.

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

2. (Facultatif) Créez l'utilisateur système et le groupe par défaut qui exécutent les composants sur le périphérique. Vous pouvez également choisir de laisser le programme d'installation du logiciel AWS IoT Greengrass Core créer cet utilisateur et ce groupe lors de l'installation avec l'argument `--component-default-user installer`. Pour de plus amples informations, veuillez consulter [Arguments d'installation](#).

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

3. Vérifiez que l'utilisateur qui exécute le logiciel AWS IoT Greengrass Core (généralement `root`) est autorisé à exécuter `sudo` avec n'importe quel utilisateur et n'importe quel groupe.
 - a. Exécutez la commande suivante pour ouvrir le `/etc/sudoers` fichier.

```
sudo visudo
```

- b. Vérifiez que l'autorisation accordée à l'utilisateur ressemble à l'exemple suivant.

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

4. (Facultatif) Pour [exécuter des fonctions Lambda conteneurisées](#), vous devez activer [cgroups](#) v1, et vous devez activer et monter les cgroups de mémoire et de périphériques. Si vous ne prévoyez pas d'exécuter des fonctions Lambda conteneurisées, vous pouvez ignorer cette étape.

Pour activer ces options cgroups, démarrez le périphérique avec les paramètres du noyau Linux suivants.

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

Pour plus d'informations sur l'affichage et la définition des paramètres du noyau de votre appareil, consultez la documentation de votre système d'exploitation et de votre chargeur de démarrage. Suivez les instructions pour définir définitivement les paramètres du noyau.

5. Installez toutes les autres dépendances requises sur votre appareil, comme indiqué dans la liste des exigences figurant dans [Exigences relatives aux dispositifs](#).

Configuration d'un appareil Windows

Note

Cette fonctionnalité est disponible pour les versions 2.5.0 et ultérieures du composant [Greengrass](#) nucleus.

Pour configurer un appareil Windows pour AWS IoT Greengrass V2

1. Installez le moteur d'exécution Java, dont le logiciel AWS IoT Greengrass Core a besoin pour fonctionner. Nous vous recommandons d'utiliser les versions de support à long terme d'[Amazon Corretto](#) ou d'OpenJDK. La version 8 ou supérieure est requise.
2. Vérifiez si Java est disponible sur la variable système [PATH](#), et ajoutez-le dans le cas contraire. Le LocalSystem compte exécute le logiciel AWS IoT Greengrass Core. Vous devez donc ajouter Java à la variable système PATH au lieu de la variable utilisateur PATH pour votre utilisateur. Procédez comme suit :
 - a. Appuyez sur la touche Windows pour ouvrir le menu de démarrage.
 - b. Tapez **environment variables** pour rechercher les options du système dans le menu Démarrer.
 - c. Dans les résultats de recherche du menu Démarrer, choisissez Modifier les variables d'environnement du système pour ouvrir la fenêtre des propriétés du système.
 - d. Choisissez les variables d'environnement... pour ouvrir la fenêtre des variables d'environnement.
 - e. Sous Variables système, sélectionnez Chemin, puis Modifier. Dans la fenêtre Modifier la variable d'environnement, vous pouvez afficher chaque chemin sur une ligne distincte.
 - f. Vérifiez si le chemin d'accès au bin dossier d'installation de Java est présent. Le chemin peut ressembler à celui de l'exemple suivant.


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

- g. Si le bin dossier de l'installation Java est absent de Path, choisissez Nouveau pour l'ajouter, puis OK.
3. Ouvrez l'invite de commande Windows (cmd.exe) en tant qu'administrateur.
4. Créez l'utilisateur par défaut dans le LocalSystem compte sur l'appareil Windows.
password Remplacez-le par un mot de passe sécurisé.

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

Tip

En fonction de votre configuration Windows, le mot de passe de l'utilisateur peut être configuré pour expirer à une date ultérieure. Pour vous assurer que vos applications Greengrass continuent de fonctionner, suivez la date d'expiration du mot de passe et mettez-le à jour avant son expiration. Vous pouvez également définir le mot de passe de l'utilisateur pour qu'il n'expire jamais.

- Pour vérifier la date d'expiration d'un utilisateur et de son mot de passe, exécutez la commande suivante.

```
net user ggc_user | findstr /C:expires
```

- Pour définir le mot de passe d'un utilisateur afin qu'il n'expire jamais, exécutez la commande suivante.

```
wmic UserAccount where "Name='ggc_user'" set PasswordExpires=False
```

- Si vous utilisez Windows 10 ou une version ultérieure où la [wmi commande est obsolète](#), exécutez la commande suivante PowerShell .

```
Get-CimInstance -Query "SELECT * from Win32_UserAccount WHERE name = 'ggc_user'" | Set-CimInstance -Property @{PasswordExpires="False"}
```

5. Téléchargez et installez l'[PsExecutilitaire](#) de Microsoft sur l'appareil.

6. Utilisez l' PsExec utilitaire pour stocker le nom d'utilisateur et le mot de passe de l'utilisateur par défaut dans l'instance Credential Manager du LocalSystem compte. *password* Remplacez-le par le mot de passe utilisateur que vous avez défini précédemment.

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

S'il PsExec License Agreements'ouvre, choisissez Acceptd'accepter la licence et exécutez la commande.

Note

Sur les appareils Windows, le LocalSystem compte exécute le noyau Greengrass, et vous devez utiliser l' PsExec utilitaire pour stocker les informations utilisateur par défaut dans le LocalSystem compte. L'application Credential Manager stocke ces informations dans le compte Windows de l'utilisateur actuellement connecté, plutôt que dans le LocalSystem compte.

Téléchargez le logiciel AWS IoT Greengrass de base

Vous pouvez télécharger la dernière version du logiciel AWS IoT Greengrass Core à l'adresse suivante :

- <https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest.zip>

Note

Vous pouvez télécharger une version spécifique du logiciel AWS IoT Greengrass Core à l'emplacement suivant. Remplacez *version* par la version à télécharger.

```
https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-version.zip
```

Pour télécharger le logiciel AWS IoT Greengrass Core

1. Sur votre appareil principal, téléchargez le logiciel AWS IoT Greengrass Core dans un fichier nommé `greengrass-nucleus-latest.zip`.

Linux or Unix

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

Windows Command Prompt (CMD)

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

PowerShell

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

Si vous téléchargez ce logiciel, vous acceptez le [contrat de licence du logiciel Greengrass Core](#).

2. (Facultatif) Pour vérifier la signature du logiciel Greengrass Nucleus

Note

Cette fonctionnalité est disponible avec Greengrass nucleus version 2.9.5 et versions ultérieures.

- a. Utilisez la commande suivante pour vérifier la signature de votre artefact Greengrass nucleus :

Linux or Unix

```
jarsigner -verify -certs -verbose greengrass-nucleus-latest.zip
```

Windows Command Prompt (CMD)

Le nom du fichier peut être différent selon la version du JDK que vous installez.
jdk17.0.6_10 Remplacez-le par la version du JDK que vous avez installée.

```
"C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe" -  
verify -certs -verbose greengrass-nucleus-latest.zip
```

PowerShell

Le nom du fichier peut être différent selon la version du JDK que vous installez.

jdk17.0.6_10 Remplacez-le par la version du JDK que vous avez installée.

```
'C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe' -  
verify -certs -verbose greengrass-nucleus-latest.zip
```

- b. L'`jarsigner` invocation produit une sortie qui indique les résultats de la vérification.
 - i. Si le fichier zip Greengrass nucleus est signé, le résultat contient l'instruction suivante :

```
jar verified.
```

- ii. Si le fichier zip Greengrass nucleus n'est pas signé, le résultat contient l'instruction suivante :

```
jar is unsigned.
```

- c. Si vous avez fourni l'`-certs` option Jarsigner en même temps que les `-verbose` options `-verify` et, le résultat inclut également des informations détaillées sur le certificat du signataire.
3. Décompressez le logiciel AWS IoT Greengrass Core dans un dossier de votre appareil.
GreengrassInstaller Remplacez-le par le dossier que vous souhaitez utiliser.

Linux or Unix

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

Windows Command Prompt (CMD)

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

PowerShell

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

4. (Facultatif) Exécutez la commande suivante pour voir la version du logiciel AWS IoT Greengrass Core.

```
java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
```

Important

Si vous installez une version du noyau Greengrass antérieure à la version 2.4.0, ne supprimez pas ce dossier après avoir installé le logiciel Core. AWS IoT Greengrass Le logiciel AWS IoT Greengrass Core utilise les fichiers de ce dossier pour s'exécuter.

Si vous avez téléchargé la dernière version du logiciel, vous devez installer la version 2.4.0 ou ultérieure, et vous pouvez supprimer ce dossier après avoir installé le logiciel AWS IoT Greengrass Core.

Installation du logiciel AWS IoT Greengrass Core

Exécutez le programme d'installation avec des arguments qui spécifient les actions suivantes :

- Effectuez l'installation à partir d'un fichier de configuration partiel qui indique d'utiliser les AWS ressources et les certificats que vous avez créés précédemment. Le logiciel AWS IoT Greengrass Core utilise un fichier de configuration qui spécifie la configuration de chaque composant Greengrass de l'appareil. Le programme d'installation crée un fichier de configuration complet à partir du fichier de configuration partiel que vous fournissez.
- Spécifiez d'utiliser l'utilisateur `ggc_user` du système pour exécuter les composants logiciels sur le périphérique principal. Sur les appareils Linux, cette commande indique également d'utiliser le groupe `ggc_group` système, et le programme d'installation crée l'utilisateur et le groupe système pour vous.
- Configurez le logiciel AWS IoT Greengrass Core en tant que service système qui s'exécute au démarrage. Sur les appareils Linux, cela nécessite le [système d'initialisation Systemd](#).

 Important

Sur les appareils Windows Core, vous devez configurer le logiciel AWS IoT Greengrass Core en tant que service système.

Pour plus d'informations sur les arguments que vous pouvez spécifier, consultez [Arguments d'installation](#).

 Note

Si vous utilisez AWS IoT Greengrass un appareil dont la mémoire est limitée, vous pouvez contrôler la quantité de mémoire utilisée par le logiciel AWS IoT Greengrass Core. Pour contrôler l'allocation de mémoire, vous pouvez définir les options de taille de segment de mémoire JVM dans le paramètre de `jvmOptions` configuration de votre composant Nucleus. Pour de plus amples informations, veuillez consulter [Contrôlez l'allocation de mémoire grâce aux options JVM](#).

- Si vous avez créé le certificat d'objet et la clé privée dans le AWS IoT service précédemment, suivez les étapes pour installer le logiciel AWS IoT Greengrass Core avec les fichiers de clé privée et de certificat.
- Si vous avez créé le certificat d'objet à partir d'une clé privée dans un module de sécurité matériel (HSM) plus tôt, suivez les étapes pour installer le logiciel AWS IoT Greengrass Core avec la clé privée et le certificat dans un HSM.

Installez le logiciel AWS IoT Greengrass Core avec les fichiers de clé privée et de certificat

Pour installer le logiciel AWS IoT Greengrass Core

1. Vérifiez la version du logiciel AWS IoT Greengrass Core.
 - Remplacez *GreengrassInstaller* par le chemin d'accès au dossier contenant le logiciel.

```
java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
```

2. Utilisez un éditeur de texte pour créer un fichier de configuration nommé `config.yaml` à fournir au programme d'installation.

Par exemple, sur un système basé sur Linux, vous pouvez exécuter la commande suivante pour utiliser GNU nano pour créer le fichier.

```
nano GreengrassInstaller/config.yaml
```

Copiez le contenu YAML suivant dans le fichier. Ce fichier de configuration partiel spécifie les paramètres du système et les paramètres du noyau de Greengrass.

```
---
system:
  certificateFilePath: "/greengrass/v2/device.pem.crt"
  privateKeyPath: "/greengrass/v2/private.pem.key"
  rootCaPath: "/greengrass/v2/AmazonRootCA1.pem"
  rootpath: "/greengrass/v2"
  thingName: "MyGreengrassCore"
services:
  aws.greengrass.Nucleus:
    componentType: "NUCLEUS"
    version: "2.16.1"
    configuration:
      awsRegion: "us-west-2"
      iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
      iotDataEndpoint: "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
      iotCredEndpoint: "device-credentials-prefix.credentials.iot.us-west-2.amazonaws.com"
```

Ensuite, procédez comme suit :

- Remplacez chaque instance de */greengrass/v2* par le dossier racine de Greengrass.
- Remplacez *MyGreengrassCore* par le nom de l' AWS IoT objet.
- *2.16.1* Remplacez-le par la version du logiciel AWS IoT Greengrass Core.
- Remplacez *us-west-2* par l' Région AWS endroit où vous avez créé les ressources.
- Remplacez *GreengrassCoreTokenExchangeRoleAlias* par le nom de l'alias du rôle d'échange de jetons.
- Remplacez le `iotDataEndpoint` par votre point de terminaison de AWS IoT données.

- Remplacez le point de terminaison `iotCredEndpoint` par vos AWS IoT informations d'identification.

Note

Dans ce fichier de configuration, vous pouvez personnaliser d'autres options de configuration du noyau, telles que les ports et le proxy réseau à utiliser, comme indiqué dans l'exemple suivant. Pour plus d'informations, consultez la section [Configuration du noyau de Greengrass](#).

```
---
system:
  certificateFilePath: "/greengrass/v2/device.pem.crt"
  privateKeyPath: "/greengrass/v2/private.pem.key"
  rootCaPath: "/greengrass/v2/AmazonRootCA1.pem"
  rootpath: "/greengrass/v2"
  thingName: "MyGreengrassCore"
services:
  aws.greengrass.Nucleus:
    componentType: "NUCLEUS"
    version: "2.16.1"
    configuration:
      awsRegion: "us-west-2"
      iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
      iotCredEndpoint: "device-credentials-prefix.credentials.iot.us-
west-2.amazonaws.com"
      iotDataEndpoint: "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
    mqtt:
      port: 443
    greengrassDataPlanePort: 443
    networkProxy:
      noProxyAddresses: "http://192.168.0.1,www.example.com"
      proxy:
        url: "https://my-proxy-server:1100"
        username: "Mary_Major"
        password: "pass@word1357"
```

3. Exécutez le programme d'installation et spécifiez `--init-config` de fournir le fichier de configuration.

- Remplacez `/greengrass/v2` ou `C:\greengrass\v2` par le dossier racine de Greengrass.
- Remplacez chaque instance de `GreengrassInstaller` par le dossier dans lequel vous avez décompressé le programme d'installation.

Linux or Unix

```
sudo -E java -Droot="/greengrass/v2" -Dlog.store=FILE \  
-jar ./GreengrassInstaller/lib/Greengrass.jar \  
--init-config ./GreengrassInstaller/config.yaml \  
--component-default-user ggc_user:ggc_group \  
--setup-system-service true
```

Windows Command Prompt (CMD)

```
java -Droot="C:\greengrass\v2" "-Dlog.store=FILE" ^  
-jar ./GreengrassInstaller/lib/Greengrass.jar ^  
--init-config ./GreengrassInstaller/config.yaml ^  
--component-default-user ggc_user ^  
--setup-system-service true
```

PowerShell

```
java -Droot="C:\greengrass\v2" "-Dlog.store=FILE" `   
-jar ./GreengrassInstaller/lib/Greengrass.jar `   
--init-config ./GreengrassInstaller/config.yaml `   
--component-default-user ggc_user `   
--setup-system-service true
```

Important

Sur les appareils Windows Core, vous `--setup-system-service true` devez spécifier de configurer le logiciel AWS IoT Greengrass Core en tant que service système.

Si vous le spécifiez `--setup-system-service true`, le programme d'installation affiche `Successfully set up Nucleus as a system service` s'il a configuré et exécuté le

logiciel en tant que service système. Dans le cas contraire, le programme d'installation n'affiche aucun message s'il installe le logiciel avec succès.

 Note

Vous ne pouvez pas utiliser l'`deploy-dev-tool` argument pour déployer des outils de développement locaux lorsque vous exécutez le programme d'installation sans l'`--provision true` argument. Pour plus d'informations sur le déploiement de la CLI Greengrass directement sur votre appareil, consultez. [Interface de ligne de commande Greengrass](#)

4. Vérifiez l'installation en consultant les fichiers du dossier racine.

Linux or Unix

```
ls /greengrass/v2
```

Windows Command Prompt (CMD)

```
dir C:\greengrass\v2
```

PowerShell

```
ls C:\greengrass\v2
```

Si l'installation a réussi, le dossier racine contient plusieurs dossiers, tels que `configpackages`, `etlogs`.

Installez le logiciel AWS IoT Greengrass principal avec la clé privée et le certificat dans un HSM

 Note

Cette fonctionnalité est disponible pour les versions 2.5.3 et ultérieures du composant [Greengrass](#) nucleus. AWS IoT Greengrass ne prend actuellement pas en charge cette fonctionnalité sur les appareils Windows principaux.

Pour installer le logiciel AWS IoT Greengrass Core

1. Vérifiez la version du logiciel AWS IoT Greengrass Core.

- Remplacez *GreengrassInstaller* par le chemin d'accès au dossier contenant le logiciel.

```
java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
```

2. Pour permettre au logiciel AWS IoT Greengrass Core d'utiliser la clé privée et le certificat du HSM, installez le [composant fournisseur PKCS #11](#) lors de l'installation du logiciel AWS IoT Greengrass Core. Le composant fournisseur PKCS #11 est un plugin que vous pouvez configurer lors de l'installation. Vous pouvez télécharger la dernière version du composant fournisseur PKCS #11 depuis l'emplacement suivant :

- <https://d2s8p88vqu9w66.cloudfront.net/releases/Pkcs11Provider/aws.greengrass.crypto.pkcs11Provider-Latest.jar>

Téléchargez le plugin du fournisseur PKCS #11 dans un fichier nommé `aws.greengrass.crypto.Pkcs11Provider.jar`.

GreengrassInstaller Remplacez-le par le dossier que vous souhaitez utiliser.

```
curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/Pkcs11Provider/
aws.greengrass.crypto.Pkcs11Provider-latest.jar > GreengrassInstaller/
aws.greengrass.crypto.Pkcs11Provider.jar
```

Si vous téléchargez ce logiciel, vous acceptez le [contrat de licence du logiciel Greengrass Core](#).

3. Utilisez un éditeur de texte pour créer un fichier de configuration nommé `config.yaml` à fournir au programme d'installation.

Par exemple, sur un système basé sur Linux, vous pouvez exécuter la commande suivante pour utiliser GNU nano pour créer le fichier.

```
nano GreengrassInstaller/config.yaml
```

Copiez le contenu YAML suivant dans le fichier. Ce fichier de configuration partiel spécifie les paramètres système, les paramètres du noyau Greengrass et les paramètres du fournisseur PKCS #11.

```

---
system:
  certificateFilePath: "pkcs11:object=iotdevicekey;type=cert"
  privateKeyPath: "pkcs11:object=iotdevicekey;type=private"
  rootCaPath: "/greengrass/v2/AmazonRootCA1.pem"
  rootpath: "/greengrass/v2"
  thingName: "MyGreengrassCore"
services:
  aws.greengrass.Nucleus:
    componentType: "NUCLEUS"
    version: "2.16.1"
    configuration:
      awsRegion: "us-west-2"
      iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
      iotDataEndpoint: "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
      iotCredEndpoint: "device-credentials-prefix.credentials.iot.us-west-2.amazonaws.com"
  aws.greengrass.crypto.Pkcs11Provider:
    configuration:
      name: "softhsm_pkcs11"
      library: "/usr/local/Cellar/softhsm/2.6.1/lib/softhsm/libsofthsm2.so"
      slot: 1
      userPin: "1234"

```

Ensuite, procédez comme suit :

- Remplacez chaque instance du PKCS #11 par le libellé URIs de l'objet *iotdevicekey* dans lequel vous avez créé la clé privée et importé le certificat.
- Remplacez chaque instance de */greengrass/v2* par le dossier racine de Greengrass.
- Remplacez *MyGreengrassCore* par le nom de l' AWS IoT objet.
- *2.16.1* Remplacez-le par la version du logiciel AWS IoT Greengrass Core.
- Remplacez *us-west-2* par l' Région AWS endroit où vous avez créé les ressources.
- Remplacez *GreengrassCoreTokenExchangeRoleAlias* par le nom de l'alias du rôle d'échange de jetons.
- Remplacez le *iotDataEndpoint* par votre point de terminaison de AWS IoT données.
- Remplacez le point de terminaison *iotCredEndpoint* par vos AWS IoT informations d'identification.

- Remplacez les paramètres de configuration du `aws.greengrass.crypto.Pkcs11Provider` composant par les valeurs de la configuration HSM sur le périphérique principal.

Note

Dans ce fichier de configuration, vous pouvez personnaliser d'autres options de configuration du noyau, telles que les ports et le proxy réseau à utiliser, comme indiqué dans l'exemple suivant. Pour plus d'informations, consultez la section [Configuration du noyau de Greengrass](#).

```
---
system:
  certificateFilePath: "pkcs11:object=iotdevicekey;type=cert"
  privateKeyPath: "pkcs11:object=iotdevicekey;type=private"
  rootCaPath: "/greengrass/v2/AmazonRootCA1.pem"
  rootpath: "/greengrass/v2"
  thingName: "MyGreengrassCore"
services:
  aws.greengrass.Nucleus:
    componentType: "NUCLEUS"
    version: "2.16.1"
    configuration:
      awsRegion: "us-west-2"
      iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
      iotDataEndpoint: "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
      iotCredEndpoint: "device-credentials-prefix.credentials.iot.us-west-2.amazonaws.com"
      mqtt:
        port: 443
      greengrassDataPlanePort: 443
      networkProxy:
        noProxyAddresses: "http://192.168.0.1,www.example.com"
        proxy:
          url: "https://my-proxy-server:1100"
          username: "Mary_Major"
          password: "pass@word1357"
  aws.greengrass.crypto.Pkcs11Provider:
    configuration:
      name: "softhsm_pkcs11"
      library: "/usr/local/Cellar/softhsm/2.6.1/lib/softhsm/libsofthsm2.so"
```

```
slot: 1
userPin: "1234"
```

4. Exécutez le programme d'installation et spécifiez `--init-config` de fournir le fichier de configuration.

- `/greengrass/v2` Remplacez-le par le dossier racine de Greengrass.
- Remplacez chaque instance de `GreengrassInstaller` par le dossier dans lequel vous avez décompressé le programme d'installation.

```
sudo -E java -Droot="/greengrass/v2" -Dlog.store=FILE \  
-jar ./GreengrassInstaller/lib/Greengrass.jar \  
--trusted-plugin ./GreengrassInstaller/aws.greengrass.crypto.Pkcs11Provider.jar \  
--init-config ./GreengrassInstaller/config.yaml \  
--component-default-user ggc_user:ggc_group \  
--setup-system-service true
```

Important

Sur les appareils Windows Core, vous `--setup-system-service true` devez spécifier de configurer le logiciel AWS IoT Greengrass Core en tant que service système.

Si vous le spécifiez `--setup-system-service true`, le programme d'installation affiche `Successfully set up Nucleus as a system service` s'il a configuré et exécuté le logiciel en tant que service système. Dans le cas contraire, le programme d'installation n'affiche aucun message s'il installe le logiciel avec succès.

Note

Vous ne pouvez pas utiliser l'`deploy-dev-tool` argument pour déployer des outils de développement locaux lorsque vous exécutez le programme d'installation sans l'`--provision true` argument. Pour plus d'informations sur le déploiement de la CLI Greengrass directement sur votre appareil, consultez. [Interface de ligne de commande Greengrass](#)

5. Vérifiez l'installation en consultant les fichiers du dossier racine.

Linux or Unix

```
ls /greengrass/v2
```

Windows Command Prompt (CMD)

```
dir C:\greengrass\v2
```

PowerShell

```
ls C:\greengrass\v2
```

Si l'installation a réussi, le dossier racine contient plusieurs dossiers, tels que configpackages, etlogs.

Si vous avez installé le logiciel AWS IoT Greengrass Core en tant que service système, le programme d'installation exécute le logiciel pour vous. Dans le cas contraire, vous devez exécuter le logiciel manuellement. Pour de plus amples informations, veuillez consulter [Exécutez le logiciel AWS IoT Greengrass Core](#).

Note

Lorsque le logiciel AWS IoT Greengrass Core se connecte au cloud, votre appareil est reconnu comme un appareil Core.

Pour plus d'informations sur la configuration et l'utilisation du logiciel AWS IoT Greengrass, consultez les rubriques suivantes :

- [Configuration du logiciel AWS IoT Greengrass de base](#)
- [Développer des AWS IoT Greengrass composants](#)
- [Déployer AWS IoT Greengrass des composants sur des appareils](#)
- [Interface de ligne de commande Greengrass](#)

Installation AWS IoT Greengrass du logiciel de base avec provisionnement du AWS IoT parc

Cette fonctionnalité est disponible pour les versions 2.4.0 et ultérieures du composant [Greengrass](#) nucleus.

Avec le provisionnement du AWS IoT parc, vous pouvez configurer AWS IoT pour générer et fournir en toute sécurité des certificats d'appareils X.509 et des clés privées à vos appareils lorsqu'ils se connectent AWS IoT pour la première fois. AWS IoT fournit des certificats clients signés par l'autorité de certification Amazon Root (CA). Vous pouvez également configurer AWS IoT pour spécifier des groupes d'objets, des types d'objets et des autorisations pour les appareils principaux de Greengrass que vous approvisionnez dans le cadre du provisionnement de flotte. Vous définissez un modèle de provisionnement pour définir le mode AWS IoT d'approvisionnement de chaque appareil. Le modèle de provisionnement spécifie les ressources d'objet, de politique et de certificat à créer pour un appareil lors du provisionnement. Pour plus d'informations, consultez la section [Modèles de provisionnement](#) dans le Guide du AWS IoT Core développeur.

AWS IoT Greengrass fournit un plugin de provisionnement de AWS IoT flotte que vous pouvez utiliser pour installer le logiciel AWS IoT Greengrass Core à l'aide AWS des ressources créées par le provisionnement de AWS IoT flotte. Le plugin de provisionnement de flotte utilise le provisionnement par réclamation. Les appareils utilisent un certificat de demande d'approvisionnement et une clé privée pour obtenir un certificat de périphérique X.509 et une clé privée uniques qu'ils peuvent utiliser pour leurs opérations régulières. Vous pouvez intégrer le certificat de réclamation et la clé privée dans chaque appareil pendant la fabrication, afin que vos clients puissent activer les appareils ultérieurement lorsque chaque appareil sera en ligne. Vous pouvez utiliser le même certificat de réclamation et la même clé privée pour plusieurs appareils. Pour plus d'informations, consultez la section [Provisionnement par réclamation](#) dans le Guide du AWS IoT Core développeur.

Note

Le plug-in de provisionnement de flotte ne prend actuellement pas en charge le stockage de clés privées et de fichiers de certificats dans un module de sécurité matérielle (HSM). Pour utiliser un HSM, [installez le logiciel AWS IoT Greengrass Core avec un provisionnement manuel](#).

Pour installer le logiciel AWS IoT Greengrass Core avec provisionnement de AWS IoT flotte, vous devez configurer les ressources utilisées pour approvisionner Compte AWS les AWS IoT appareils

principaux de Greengrass. Ces ressources incluent un modèle de provisionnement, des certificats de réclamation et un rôle [IAM d'échange de jetons](#). Après avoir créé ces ressources, vous pouvez les réutiliser pour approvisionner plusieurs appareils principaux dans un parc. Pour de plus amples informations, veuillez consulter [Configurer le provisionnement du AWS IoT parc pour les appareils principaux de Greengrass](#).

 Important

Avant de télécharger le logiciel AWS IoT Greengrass Core, vérifiez que votre appareil principal répond à la [configuration requise](#) pour installer et exécuter le logiciel AWS IoT Greengrass Core v2.0.

Rubriques

- [Conditions préalables](#)
- [Récupérer des points de AWS IoT terminaison](#)
- [Télécharger les certificats sur l'appareil](#)
- [Configuration de l'environnement de l'appareil](#)
- [Téléchargez le logiciel AWS IoT Greengrass de base](#)
- [Téléchargez le plugin de provisionnement de AWS IoT flotte](#)
- [Installation du logiciel AWS IoT Greengrass de base](#)
- [Configurer le provisionnement du AWS IoT parc pour les appareils principaux de Greengrass](#)
- [Configuration du plugin de provisionnement de AWS IoT flotte](#)
- [AWS IoT journal des modifications du plugin de provisionnement de flotte](#)

Conditions préalables

Pour installer le logiciel AWS IoT Greengrass Core avec le provisionnement du AWS IoT parc, vous devez d'abord [configurer le provisionnement du AWS IoT parc pour les appareils principaux de Greengrass](#). Après avoir effectué ces étapes une fois, vous pouvez utiliser le provisionnement du parc pour installer le logiciel AWS IoT Greengrass Core sur autant d'appareils que vous le souhaitez.

Récupérer des points de AWS IoT terminaison

Obtenez les AWS IoT points de terminaison qui vous Compte AWS conviennent et enregistrez-les pour les utiliser ultérieurement. Votre appareil utilise ces points de terminaison pour se connecter à AWS IoT. Procédez comme suit :

1. Obtenez le point de terminaison de AWS IoT données pour votre Compte AWS.

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

La réponse ressemble à l'exemple suivant, si la demande aboutit.

```
{
  "endpointAddress": "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
}
```

2. Obtenez le point de terminaison des informations d' AWS IoT identification pour votre Compte AWS.

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

La réponse ressemble à l'exemple suivant, si la demande aboutit.

```
{
  "endpointAddress": "device-credentials-prefix.credentials.iot.us-  
west-2.amazonaws.com"
}
```

Télécharger les certificats sur l'appareil

L'appareil utilise un certificat de réclamation et une clé privée pour authentifier sa demande de fourniture de AWS ressources et acquérir un certificat de périphérique X.509. Vous pouvez intégrer le certificat de réclamation et la clé privée dans l'appareil pendant la fabrication, ou copier le certificat et la clé sur l'appareil lors de l'installation. Dans cette section, vous devez copier le certificat de réclamation et la clé privée sur l'appareil. Vous téléchargez également le certificat Amazon Root Certificate Authority (CA) sur l'appareil.

 Important

L'approvisionnement des clés privées des réclamations doit être sécurisé à tout moment, y compris sur les appareils principaux de Greengrass. Nous vous recommandons d'utiliser CloudWatch les statistiques et les journaux Amazon pour détecter les signes d'utilisation abusive, tels que l'utilisation non autorisée du certificat de réclamation pour approvisionner des appareils. Si vous détectez une utilisation abusive, désactivez le certificat de demande d'approvisionnement afin qu'il ne puisse pas être utilisé pour le provisionnement des appareils. Pour plus d'informations, consultez la section [Surveillance AWS IoT](#) dans le guide du AWS IoT Core développeur.

Pour vous aider à mieux gérer le nombre d'appareils et les appareils qui s'enregistrent dans le vôtre Compte AWS, vous pouvez spécifier un crochet de préapprovisionnement lorsque vous créez un modèle de provisionnement de flotte. Un hook de pré-provisionnement est une AWS Lambda fonction qui valide les paramètres du modèle fournis par les appareils lors de l'enregistrement. Par exemple, vous pouvez créer un hook de pré-provisionnement qui compare l'identifiant d'un appareil à une base de données afin de vérifier que l'appareil est autorisé à effectuer le provisionnement. Pour plus d'informations, consultez la section [Pre-provisioning hooks](#) dans le Guide du AWS IoT Core développeur.

Pour télécharger les certificats de réclamation sur l'appareil

1. Copiez le certificat de réclamation et la clé privée sur l'appareil. Si SSH et SCP sont activés sur l'ordinateur de développement et le périphérique, vous pouvez utiliser la scp commande sur votre ordinateur de développement pour transférer le certificat de réclamation et la clé privée. L'exemple de commande suivant transfère ces fichiers vers le périphérique dans un dossier nommé `claim-certs` sur votre ordinateur de développement. Remplacez *device-ip-address* par l'adresse IP de votre appareil.

```
scp -r claim-certs/ device-ip-address:~
```

2. Créez le dossier racine Greengrass sur l'appareil. Vous installerez ultérieurement le logiciel AWS IoT Greengrass Core dans ce dossier.

 Note

Pour Windows, la limitation de la longueur du chemin est de 260 caractères. Si vous utilisez Windows, utilisez un dossier racine tel que `C:\greengrass\v2` ou `D:`

`\greengrass\v2` pour conserver les chemins des composants Greengrass en dessous de la limite de 260 caractères.

Linux or Unix

- `/greengrass/v2` Remplacez-le par le dossier à utiliser.

```
sudo mkdir -p /greengrass/v2
```

Windows Command Prompt

- `C:\greengrass\v2` Remplacez-le par le dossier à utiliser.

```
mkdir C:\greengrass\v2
```

PowerShell

- `C:\greengrass\v2` Remplacez-le par le dossier à utiliser.

```
mkdir C:\greengrass\v2
```

3. (Linux uniquement) Définissez les autorisations du parent du dossier racine de Greengrass.

- Remplacez `/greengrass` par le parent du dossier racine.

```
sudo chmod 755 /greengrass
```

4. Déplacez les certificats de réclamation vers le dossier racine de Greengrass.

- Remplacez `/greengrass/v2` ou `C:\greengrass\v2` par le dossier racine de Greengrass.

Linux or Unix

```
sudo mv ~/claim-certs /greengrass/v2
```

Windows Command Prompt (CMD)

```
move %USERPROFILE%\claim-certs C:\greengrass\v2
```

PowerShell

```
mv -Path ~\claim-certs -Destination C:\greengrass\v2
```

5. Téléchargez le certificat de l'autorité de certification racine (CA) Amazon. AWS IoT les certificats sont associés par défaut au certificat de l'autorité de certification racine d'Amazon.

Linux or Unix

```
sudo curl -o /greengrass/v2/AmazonRootCA1.pem https://www.amazontrust.com/repository/AmazonRootCA1.pem
```

Windows Command Prompt (CMD)

```
curl -o C:\greengrass\v2\AmazonRootCA1.pem https://www.amazontrust.com/repository/AmazonRootCA1.pem
```

PowerShell

```
iwr -Uri https://www.amazontrust.com/repository/AmazonRootCA1.pem -OutFile C:\greengrass\v2\AmazonRootCA1.pem
```

Configuration de l'environnement de l'appareil

Suivez les étapes décrites dans cette section pour configurer un appareil Linux ou Windows à utiliser comme périphérique AWS IoT Greengrass principal.

Configuration d'un appareil Linux

Pour configurer un appareil Linux pour AWS IoT Greengrass V2

1. Installez le moteur d'exécution Java, dont le logiciel AWS IoT Greengrass Core a besoin pour fonctionner. Nous vous recommandons d'utiliser les versions de support à long terme d'[Amazon](#)

[Corretto](#) ou d'OpenJDK. La version 8 ou supérieure est requise. Les commandes suivantes vous montrent comment installer OpenJDK sur votre appareil.

- Pour les distributions basées sur Debian ou Ubuntu :

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

- Pour les distributions basées sur Red Hat :

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

- Dans Amazon Linux 2 :

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

- Pour Amazon Linux 2023 :

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

Lorsque l'installation est terminée, exécutez la commande suivante pour vérifier que Java s'exécute sur votre appareil Linux.

```
java -version
```

La commande affiche la version de Java exécutée sur le périphérique. Par exemple, sur une distribution basée sur Debian, le résultat peut ressembler à celui de l'exemple suivant.

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

2. (Facultatif) Créez l'utilisateur système et le groupe par défaut qui exécutent les composants sur le périphérique. Vous pouvez également choisir de laisser le programme d'installation du logiciel AWS IoT Greengrass Core créer cet utilisateur et ce groupe lors de l'installation avec l'argument `--component-default-user` installer. Pour de plus amples informations, veuillez consulter [Arguments d'installation](#).

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

```
sudo groupadd --system ggc_group
```

3. Vérifiez que l'utilisateur qui exécute le logiciel AWS IoT Greengrass Core (généralement `root`) est autorisé à exécuter `sudo` avec n'importe quel utilisateur et n'importe quel groupe.

- a. Exécutez la commande suivante pour ouvrir le `/etc/sudoers` fichier.

```
sudo visudo
```

- b. Vérifiez que l'autorisation accordée à l'utilisateur ressemble à l'exemple suivant.

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

4. (Facultatif) Pour [exécuter des fonctions Lambda conteneurisées](#), vous devez activer [cgroups](#) v1, et vous devez activer et monter les cgroups de mémoire et de périphériques. Si vous ne prévoyez pas d'exécuter des fonctions Lambda conteneurisées, vous pouvez ignorer cette étape.

Pour activer ces options cgroups, démarrez le périphérique avec les paramètres du noyau Linux suivants.

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

Pour plus d'informations sur l'affichage et la définition des paramètres du noyau de votre appareil, consultez la documentation de votre système d'exploitation et de votre chargeur de démarrage. Suivez les instructions pour définir définitivement les paramètres du noyau.

5. Installez toutes les autres dépendances requises sur votre appareil, comme indiqué dans la liste des exigences figurant dans [Exigences relatives aux dispositifs](#).

Configuration d'un appareil Windows

Note

Cette fonctionnalité est disponible pour les versions 2.5.0 et ultérieures du composant [Greengrass](#) nucleus.

Pour configurer un appareil Windows pour AWS IoT Greengrass V2

1. Installez le moteur d'exécution Java, dont le logiciel AWS IoT Greengrass Core a besoin pour fonctionner. Nous vous recommandons d'utiliser les versions de support à long terme d'[Amazon Corretto](#) ou d'OpenJDK. La version 8 ou supérieure est requise.
2. Vérifiez si Java est disponible sur la variable système [PATH](#), et ajoutez-le dans le cas contraire. Le LocalSystem compte exécute le logiciel AWS IoT Greengrass Core. Vous devez donc ajouter Java à la variable système PATH au lieu de la variable utilisateur PATH pour votre utilisateur. Procédez comme suit :
 - a. Appuyez sur la touche Windows pour ouvrir le menu de démarrage.
 - b. Tapez **environment variables** pour rechercher les options du système dans le menu Démarrer.
 - c. Dans les résultats de recherche du menu Démarrer, choisissez Modifier les variables d'environnement du système pour ouvrir la fenêtre des propriétés du système.
 - d. Choisissez les variables d'environnement... pour ouvrir la fenêtre des variables d'environnement.
 - e. Sous Variables système, sélectionnez Chemin, puis Modifier. Dans la fenêtre Modifier la variable d'environnement, vous pouvez afficher chaque chemin sur une ligne distincte.
 - f. Vérifiez si le chemin d'accès au bin dossier d'installation de Java est présent. Le chemin peut ressembler à celui de l'exemple suivant.

```
C:\\Program Files\\Amazon Corretto\\jdk11.0.13_8\\bin
```
 - g. Si le bin dossier de l'installation Java est absent de Path, choisissez Nouveau pour l'ajouter, puis OK.
3. Ouvrez l'invite de commande Windows (cmd.exe) en tant qu'administrateur.
4. Créez l'utilisateur par défaut dans le LocalSystem compte sur l'appareil Windows. *password* Remplacez-le par un mot de passe sécurisé.

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

Tip

En fonction de votre configuration Windows, le mot de passe de l'utilisateur peut être configuré pour expirer à une date ultérieure. Pour vous assurer que vos applications

Greengrass continue de fonctionner, suivez la date d'expiration du mot de passe et mettez-le à jour avant son expiration. Vous pouvez également définir le mot de passe de l'utilisateur pour qu'il n'expire jamais.

- Pour vérifier la date d'expiration d'un utilisateur et de son mot de passe, exécutez la commande suivante.

```
net user ggc_user | findstr /C:expires
```

- Pour définir le mot de passe d'un utilisateur afin qu'il n'expire jamais, exécutez la commande suivante.

```
wmic UserAccount where "Name='ggc_user'" set PasswordExpires=False
```

- Si vous utilisez Windows 10 ou une version ultérieure où la [wmic commande est obsolète](#), exécutez la commande suivante PowerShell .

```
Get-CimInstance -Query "SELECT * from Win32_UserAccount WHERE name = 'ggc_user'" | Set-CimInstance -Property @{PasswordExpires="False"}
```

5. Téléchargez et installez l'[PsExec utilitaire](#) de Microsoft sur l'appareil.
6. Utilisez l' PsExec utilitaire pour stocker le nom d'utilisateur et le mot de passe de l'utilisateur par défaut dans l'instance Credential Manager du LocalSystem compte. *password* Remplacez-le par le mot de passe utilisateur que vous avez défini précédemment.

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

S'il PsExec License Agreements'ouvre, choisissez Acceptd'accepter la licence et exécutez la commande.

Note

Sur les appareils Windows, le LocalSystem compte exécute le noyau Greengrass, et vous devez utiliser l' PsExec utilitaire pour stocker les informations utilisateur par défaut dans le LocalSystem compte. L'application Credential Manager stocke ces informations dans le compte Windows de l'utilisateur actuellement connecté, plutôt que dans le LocalSystem compte.

Téléchargez le logiciel AWS IoT Greengrass de base

Vous pouvez télécharger la dernière version du logiciel AWS IoT Greengrass Core à l'adresse suivante :

- <https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest.zip>

Note

Vous pouvez télécharger une version spécifique du logiciel AWS IoT Greengrass Core à l'emplacement suivant. Remplacez *version* par la version à télécharger.

```
https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-version.zip
```

Pour télécharger le logiciel AWS IoT Greengrass Core

1. Sur votre appareil principal, téléchargez le logiciel AWS IoT Greengrass Core dans un fichier nommé `greengrass-nucleus-latest.zip`.

Linux or Unix

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

Windows Command Prompt (CMD)

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

PowerShell

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

Si vous téléchargez ce logiciel, vous acceptez le [contrat de licence du logiciel Greengrass Core](#).

2. (Facultatif) Pour vérifier la signature du logiciel Greengrass Nucleus

 Note

Cette fonctionnalité est disponible avec Greengrass nucleus version 2.9.5 et versions ultérieures.

- a. Utilisez la commande suivante pour vérifier la signature de votre artefact Greengrass nucleus :

Linux or Unix

```
jarsigner -verify -certs -verbose greengrass-nucleus-latest.zip
```

Windows Command Prompt (CMD)

Le nom du fichier peut être différent selon la version du JDK que vous installez.
jdk17.0.6_10 Remplacez-le par la version du JDK que vous avez installée.

```
"C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe" -  
verify -certs -verbose greengrass-nucleus-latest.zip
```

PowerShell

Le nom du fichier peut être différent selon la version du JDK que vous installez.
jdk17.0.6_10 Remplacez-le par la version du JDK que vous avez installée.

```
'C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe' -  
verify -certs -verbose greengrass-nucleus-latest.zip
```

- b. L'`jarsigner` invocation produit une sortie qui indique les résultats de la vérification.
 - i. Si le fichier zip Greengrass nucleus est signé, le résultat contient l'instruction suivante :

```
jar verified.
```
 - ii. Si le fichier zip Greengrass nucleus n'est pas signé, le résultat contient l'instruction suivante :

```
jar is unsigned.
```

- c. Si vous avez fourni l'-certsoption Jarsigner en même temps que les -verbose options -verify et, le résultat inclut également des informations détaillées sur le certificat du signataire.
3. Décompressez le logiciel AWS IoT Greengrass Core dans un dossier de votre appareil. *GreengrassInstaller* Remplacez-le par le dossier que vous souhaitez utiliser.

Linux or Unix

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

Windows Command Prompt (CMD)

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

PowerShell

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

4. (Facultatif) Exécutez la commande suivante pour voir la version du logiciel AWS IoT Greengrass Core.

```
java -jar ./ GreengrassInstaller/lib/Greengrass.jar --version
```

Important

Si vous installez une version du noyau Greengrass antérieure à la version 2.4.0, ne supprimez pas ce dossier après avoir installé le logiciel Core. AWS IoT Greengrass Le logiciel AWS IoT Greengrass Core utilise les fichiers de ce dossier pour s'exécuter.

Si vous avez téléchargé la dernière version du logiciel, vous devez installer la version 2.4.0 ou ultérieure, et vous pouvez supprimer ce dossier après avoir installé le logiciel AWS IoT Greengrass Core.

Téléchargez le plugin de provisionnement de AWS IoT flotte

Vous pouvez télécharger la dernière version du plugin de provisionnement de AWS IoT flotte à l'adresse suivante :

- <https://d2s8p88vqu9w66.cloudfront.net/releases/aws-greengrass-FleetProvisioningByClaim/fleetprovisioningbyclaim-latest.jar>

Note

Vous pouvez télécharger une version spécifique du plugin de provisionnement de AWS IoT flotte à l'emplacement suivant. Remplacez *version* par la version à télécharger. Pour plus d'informations sur chaque version du plugin de provisionnement de flotte, consultez [AWS IoT journal des modifications du plugin de provisionnement de flotte](#).

```
https://d2s8p88vqu9w66.cloudfront.net/releases/aws-greengrass-  
FleetProvisioningByClaim/fleetprovisioningbyclaim-version.jar
```

Le plugin de provisionnement de flotte est open source. Pour consulter son code source, consultez le [plugin de provisionnement de AWS IoT flotte](#) sur GitHub.

Pour télécharger le plugin de provisionnement de AWS IoT flotte

- Sur votre appareil, téléchargez le plug-in de provisionnement de AWS IoT flotte dans un fichier nommé `aws.greengrass.FleetProvisioningByClaim.jar`. *GreengrassInstaller* Remplacez-le par le dossier que vous souhaitez utiliser.

Linux or Unix

```
curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/aws-greengrass-  
FleetProvisioningByClaim/fleetprovisioningbyclaim-latest.jar  
> GreengrassInstaller/aws.greengrass.FleetProvisioningByClaim.jar
```

Windows Command Prompt (CMD)

```
curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/aws-greengrass-  
FleetProvisioningByClaim/fleetprovisioningbyclaim-latest.jar  
> GreengrassInstaller/aws.greengrass.FleetProvisioningByClaim.jar
```

PowerShell

```
iwr -Uri https://d2s8p88vqu9w66.cloudfront.net/releases/aws-greengrass-  
FleetProvisioningByClaim/fleetprovisioningbyclaim-latest.jar -  
OutFile GreengrassInstaller/aws.greengrass.FleetProvisioningByClaim.jar
```

Si vous téléchargez ce logiciel, vous acceptez le [contrat de licence du logiciel Greengrass Core](#).

Installation du logiciel AWS IoT Greengrass de base

Exécutez le programme d'installation avec des arguments qui spécifient les actions suivantes :

- Effectuez l'installation à partir d'un fichier de configuration partiel qui indique d'utiliser le plug-in de provisionnement de flotte pour provisionner les AWS ressources. Le logiciel AWS IoT Greengrass Core utilise un fichier de configuration qui spécifie la configuration de chaque composant Greengrass de l'appareil. Le programme d'installation crée un fichier de configuration complet à partir du fichier de configuration partiel que vous fournissez et AWS des ressources créées par le plug-in de provisionnement de flotte.
- Spécifiez d'utiliser l'utilisateur `ggc_user` du système pour exécuter les composants logiciels sur le périphérique principal. Sur les appareils Linux, cette commande indique également d'utiliser le groupe `ggc_group` système, et le programme d'installation crée l'utilisateur et le groupe système pour vous.
- Configurez le logiciel AWS IoT Greengrass Core en tant que service système qui s'exécute au démarrage. Sur les appareils Linux, cela nécessite le [système d'initialisation Systemd](#).

Important

Sur les appareils Windows Core, vous devez configurer le logiciel AWS IoT Greengrass Core en tant que service système.

Pour plus d'informations sur les arguments que vous pouvez spécifier, consultez [Arguments d'installation](#).

Note

Si vous utilisez AWS IoT Greengrass un appareil dont la mémoire est limitée, vous pouvez contrôler la quantité de mémoire utilisée par le logiciel AWS IoT Greengrass Core. Pour contrôler l'allocation de mémoire, vous pouvez définir les options de taille de segment de mémoire JVM dans le paramètre de `jvmOptions` configuration de votre composant Nucleus. Pour de plus amples informations, veuillez consulter [Contrôlez l'allocation de mémoire grâce aux options JVM](#).

Pour installer le logiciel AWS IoT Greengrass Core

1. Vérifiez la version du logiciel AWS IoT Greengrass Core.
 - Remplacez *GreengrassInstaller* par le chemin d'accès au dossier contenant le logiciel.

```
java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
```

2. Utilisez un éditeur de texte pour créer un fichier de configuration nommé `config.yaml` à fournir au programme d'installation.

Par exemple, sur un système basé sur Linux, vous pouvez exécuter la commande suivante pour utiliser GNU nano pour créer le fichier.

```
nano GreengrassInstaller/config.yaml
```

Copiez le contenu YAML suivant dans le fichier. Ce fichier de configuration partiel spécifie les paramètres du plugin de provisionnement de flotte. Pour plus d'informations sur les options que vous pouvez spécifier, consultez [Configuration du plugin de provisionnement de AWS IoT flotte](#).

Linux or Unix

```
---
services:
  aws.greengrass.Nucleus:
    version: "2.16.1"
```

```
aws.greengrass.FleetProvisioningByClaim:
  configuration:
    rootPath: "/greengrass/v2"
    awsRegion: "us-west-2"
    iotDataEndpoint: "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
    iotCredentialEndpoint: "device-credentials-prefix.credentials.iot.us-west-2.amazonaws.com"
    iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
    provisioningTemplate: "GreengrassFleetProvisioningTemplate"
    claimCertificatePath: "/greengrass/v2/claim-certs/claim.pem.crt"
    claimCertificatePrivateKeyPath: "/greengrass/v2/claim-certs/claim.private.pem.key"
    rootCaPath: "/greengrass/v2/AmazonRootCA1.pem"
    templateParameters:
      ThingName: "MyGreengrassCore"
      ThingGroupName: "MyGreengrassCoreGroup"
```

Windows

```
---
services:
  aws.greengrass.Nucleus:
    version: "2.16.1"
  aws.greengrass.FleetProvisioningByClaim:
    configuration:
      rootPath: "C:\\greengrass\\v2"
      awsRegion: "us-west-2"
      iotDataEndpoint: "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
      iotCredentialEndpoint: "device-credentials-prefix.credentials.iot.us-west-2.amazonaws.com"
      iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
      provisioningTemplate: "GreengrassFleetProvisioningTemplate"
      claimCertificatePath: "C:\\greengrass\\v2\\claim-certs\\claim.pem.crt"
      claimCertificatePrivateKeyPath: "C:\\greengrass\\v2\\claim-certs\\claim.private.pem.key"
      rootCaPath: "C:\\greengrass\\v2\\AmazonRootCA1.pem"
      templateParameters:
        ThingName: "MyGreengrassCore"
        ThingGroupName: "MyGreengrassCoreGroup"
```

Ensuite, procédez comme suit :

- Remplacez **2.16.1** par la version du logiciel AWS IoT Greengrass Core.
- Remplacez chaque instance de **/greengrass/v2** ou **C:\greengrass\v2** par le dossier racine de Greengrass.

 Note

Sur les appareils Windows, vous devez spécifier les séparateurs de chemin sous forme de barres obliques inverses (\\) doubles, tels que. C:\\greengrass\\v2

- **us-west-2** Remplacez-le par la AWS région dans laquelle vous avez créé le modèle de provisionnement et les autres ressources.
- Remplacez le `iotDataEndpoint` par votre point de terminaison de AWS IoT données.
- Remplacez le point de terminaison `iotCredentialEndpoint` par vos AWS IoT informations d'identification.
- Remplacez **GreengrassCoreTokenExchangeRoleAlias** par le nom de l'alias du rôle d'échange de jetons.
- Remplacez **GreengrassFleetProvisioningTemplate** par le nom du modèle de provisionnement de flotte.
- Remplacez le `claimCertificatePath` par le chemin d'accès au certificat de réclamation sur l'appareil.
- Remplacez le `claimCertificatePrivateKeyPath` par le chemin d'accès à la clé privée du certificat de réclamation sur l'appareil.
- Remplacez les paramètres du modèle (`templateParameters`) par les valeurs à utiliser pour approvisionner le périphérique. Cet exemple fait référence à l'[exemple de modèle](#) qui définit `ThingName` et définit `ThingGroupName` les paramètres.

 Note

Dans ce fichier de configuration, vous pouvez personnaliser d'autres options de configuration telles que les ports et le proxy réseau à utiliser, comme indiqué dans l'exemple suivant. Pour plus d'informations, consultez la section Configuration du [noyau de Greengrass](#).

Linux or Unix

```

---
services:
  aws.greengrass.Nucleus:
    version: "2.16.1"
    configuration:
      mqtt:
        port: 443
      greengrassDataPlanePort: 443
      networkProxy:
        noProxyAddresses: "http://192.168.0.1,www.example.com"
        proxy:
          url: "http://my-proxy-server:1100"
          username: "Mary_Major"
          password: "pass@word1357"
  aws.greengrass.FleetProvisioningByClaim:
    configuration:
      rootPath: "/greengrass/v2"
      awsRegion: "us-west-2"
      iotDataEndpoint: "device-data-prefix-ats.iot.us-
west-2.amazonaws.com"
      iotCredentialEndpoint: "device-credentials-
prefix.credentials.iot.us-west-2.amazonaws.com"
      iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
      provisioningTemplate: "GreengrassFleetProvisioningTemplate"
      claimCertificatePath: "/greengrass/v2/claim-certs/claim.pem.crt"
      claimCertificatePrivateKeyPath: "/greengrass/v2/claim-certs/
claim.private.pem.key"
      rootCaPath: "/greengrass/v2/AmazonRootCA1.pem"
      templateParameters:
        ThingName: "MyGreengrassCore"
        ThingGroupName: "MyGreengrassCoreGroup"
      mqttPort: 443
      proxyUrl: "http://my-proxy-server:1100"
      proxyUserName: "Mary_Major"
      proxyPassword: "pass@word1357"

```

Windows

```

---
services:

```

```

aws.greengrass.Nucleus:
  version: "2.16.1"
  configuration:
    mqtt:
      port: 443
    greengrassDataPlanePort: 443
    networkProxy:
      noProxyAddresses: "http://192.168.0.1,www.example.com"
      proxy:
        url: "http://my-proxy-server:1100"
        username: "Mary_Major"
        password: "pass@word1357"
aws.greengrass.FleetProvisioningByClaim:
  configuration:
    rootPath: "C:\\greengrass\\v2"
    awsRegion: "us-west-2"
    iotDataEndpoint: "device-data-prefix-ats.iot.us-
west-2.amazonaws.com"
    iotCredentialEndpoint: "device-credentials-
prefix.credentials.iot.us-west-2.amazonaws.com"
    iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
    provisioningTemplate: "GreengrassFleetProvisioningTemplate"
    claimCertificatePath: "C:\\greengrass\\v2\\claim-certs\\
\\claim.pem.crt"
    claimCertificatePrivateKeyPath: "C:\\greengrass\\v2\\claim-certs\\
\\claim.private.pem.key"
    rootCaPath: "C:\\greengrass\\v2\\AmazonRootCA1.pem"
    templateParameters:
      ThingName: "MyGreengrassCore"
      ThingGroupName: "MyGreengrassCoreGroup"
    mqttPort: 443
    proxyUrl: "http://my-proxy-server:1100"
    proxyUserName: "Mary_Major"
    proxyPassword: "pass@word1357"

```

Pour utiliser un proxy HTTPS, vous devez utiliser la version 1.1.0 ou ultérieure du plugin de provisionnement de flotte. Vous devez également spécifier le `rootCaPath` soussystem, comme indiqué dans l'exemple suivant.

Linux or Unix

```
system:
  rootCaPath: "/greengrass/v2/AmazonRootCA1.pem"
services:
  ...
```

Windows

```
---
system:
  rootCaPath: "C:\\greengrass\\v2\\AmazonRootCA1.pem"
services:
  ...
```

3. Exécutez le programme d'installation. Spécifiez `--trusted-plugin` pour fournir le plug-in de provisionnement de flotte et spécifiez `--init-config` pour fournir le fichier de configuration.
 - `/greengrass/v2` Remplacez-le par le dossier racine de Greengrass.
 - Remplacez chaque instance de `GreengrassInstaller` par le dossier dans lequel vous avez décompressé le programme d'installation.

Linux or Unix

```
sudo -E java -Droot="/greengrass/v2" -Dlog.store=FILE \
-jar ./GreengrassInstaller/lib/Greengrass.jar \
--trusted-plugin ./GreengrassInstaller/
aws.greengrass.FleetProvisioningByClaim.jar \
--init-config ./GreengrassInstaller/config.yaml \
--component-default-user ggc_user:ggc_group \
--setup-system-service true
```

Windows Command Prompt (CMD)

```
java -Droot="C:\\greengrass\\v2" "-Dlog.store=FILE" ^
-jar ./GreengrassInstaller/lib/Greengrass.jar ^
--trusted-plugin ./GreengrassInstaller/
aws.greengrass.FleetProvisioningByClaim.jar ^
--init-config ./GreengrassInstaller/config.yaml ^
--component-default-user ggc_user ^
--setup-system-service true
```

PowerShell

```
java -Droot="C:\greengrass\v2" "-Dlog.store=FILE" `
  -jar ./GreengrassInstaller/lib/Greengrass.jar `
  --trusted-plugin ./GreengrassInstaller/
aws.greengrass.FleetProvisioningByClaim.jar `
  --init-config ./GreengrassInstaller/config.yaml `
  --component-default-user ggc_user `
  --setup-system-service true
```

Important

Sur les appareils Windows Core, vous `--setup-system-service true` devez spécifier de configurer le logiciel AWS IoT Greengrass Core en tant que service système.

Si vous le spécifiez `--setup-system-service true`, le programme d'installation affiche `Successfully set up Nucleus as a system service` s'il a configuré et exécuté le logiciel en tant que service système. Dans le cas contraire, le programme d'installation n'affiche aucun message s'il installe le logiciel avec succès.

Note

Vous ne pouvez pas utiliser l'`deploy-dev-tools` argument pour déployer des outils de développement locaux lorsque vous exécutez le programme d'installation sans l'`--provision true` argument. Pour plus d'informations sur le déploiement de la CLI Greengrass directement sur votre appareil, consultez. [Interface de ligne de commande Greengrass](#)

4. Vérifiez l'installation en consultant les fichiers du dossier racine.

Linux or Unix

```
ls /greengrass/v2
```

Windows Command Prompt (CMD)

```
dir C:\greengrass\v2
```

PowerShell

```
ls C:\greengrass\v2
```

Si l'installation a réussi, le dossier racine contient plusieurs dossiers, tels que configpackages, etlogs.

Si vous avez installé le logiciel AWS IoT Greengrass Core en tant que service système, le programme d'installation exécute le logiciel pour vous. Dans le cas contraire, vous devez exécuter le logiciel manuellement. Pour de plus amples informations, veuillez consulter [Exécutez le logiciel AWS IoT Greengrass Core](#).

Pour plus d'informations sur la configuration et l'utilisation du logiciel AWS IoT Greengrass, consultez les rubriques suivantes :

- [Configuration du logiciel AWS IoT Greengrass de base](#)
- [Développer des AWS IoT Greengrass composants](#)
- [Déployer AWS IoT Greengrass des composants sur des appareils](#)
- [Interface de ligne de commande Greengrass](#)

Configurer le provisionnement du AWS IoT parc pour les appareils principaux de Greengrass

Pour [installer le logiciel AWS IoT Greengrass Core avec le provisionnement de flotte](#), vous devez d'abord configurer les ressources suivantes dans votre Compte AWS. Ces ressources permettent aux appareils de s'enregistrer AWS IoT et de fonctionner en tant qu'appareils principaux de Greengrass. Suivez les étapes décrites dans cette section une seule fois pour créer et configurer ces ressources dans votre Compte AWS.

- Rôle IAM d'échange de jetons, utilisé par les principaux appareils pour autoriser les appels aux AWS services.

- Alias de AWS IoT rôle pointant vers le rôle d'échange de jetons.
- (Facultatif) Une AWS IoT politique utilisée par les appareils principaux pour autoriser les appels vers les AWS IoT Greengrass services AWS IoT et. Cette AWS IoT politique doit `iot:AssumeRoleWithCertificate` autoriser l'alias de AWS IoT rôle pointant vers le rôle d'échange de jetons.

Vous pouvez utiliser une AWS IoT politique unique pour tous les appareils principaux de votre parc, ou vous pouvez configurer le modèle de provisionnement de votre flotte afin de créer une AWS IoT politique pour chaque appareil principal.

- Un modèle AWS IoT de provisionnement de flotte. Ce modèle doit spécifier les éléments suivants :
 - N'importe AWS IoT quel objet, ressource. Vous pouvez spécifier une liste de groupes d'objets existants pour déployer des composants sur chaque appareil lors de sa mise en ligne.
 - Une ressource AWS IoT en matière de politiques. Cette ressource peut définir l'une des propriétés suivantes :
 - Le nom d'une AWS IoT politique existante. Si vous choisissez cette option, les principaux appareils que vous créez à partir de ce modèle utilisent la même AWS IoT politique et vous pouvez gérer leurs autorisations en tant que flotte.
 - Un document AWS IoT de politique. Si vous choisissez cette option, chaque périphérique principal que vous créez à partir de ce modèle utilise une AWS IoT politique unique, et vous pouvez gérer les autorisations pour chaque périphérique principal individuel.
 - Une ressource AWS IoT de certificat. Cette ressource de certificat doit utiliser le `AWS::IoT::Certificate::Id` paramètre pour associer le certificat au périphérique principal. Pour plus d'informations, consultez la section [Just-in-time Provisioning](#) dans le Guide du AWS IoT développeur.
- Un certificat de demande d' AWS IoT approvisionnement et une clé privée pour le modèle de provisionnement de la flotte. Vous pouvez intégrer ce certificat et cette clé privée dans les appareils pendant la fabrication, afin que les appareils puissent s'enregistrer et s'approvisionner eux-mêmes lorsqu'ils sont mis en ligne.

Important

L'approvisionnement des clés privées des réclamations doit être sécurisé à tout moment, y compris sur les appareils principaux de Greengrass. Nous vous recommandons d'utiliser CloudWatch les statistiques et les journaux Amazon pour détecter les signes d'utilisation abusive, tels que l'utilisation non autorisée du certificat de réclamation pour approvisionner des appareils. Si vous détectez une utilisation abusive, désactivez le certificat de demande

d'approvisionnement afin qu'il ne puisse pas être utilisé pour le provisionnement des appareils. Pour plus d'informations, consultez la section [Surveillance AWS IoT](#) dans le guide du AWS IoT Core développeur.

Pour vous aider à mieux gérer le nombre d'appareils et les appareils qui s'enregistrent dans le vôtre Compte AWS, vous pouvez spécifier un crochet de préapprovisionnement lorsque vous créez un modèle de provisionnement de flotte. Un hook de pré-provisionnement est une AWS Lambda fonction qui valide les paramètres du modèle fournis par les appareils lors de l'enregistrement. Par exemple, vous pouvez créer un hook de pré-provisionnement qui compare l'identifiant d'un appareil à une base de données afin de vérifier que l'appareil est autorisé à effectuer le provisionnement. Pour plus d'informations, consultez la section [Pre-provisioning hooks](#) dans le Guide du AWS IoT Core développeur.

- AWS IoT Politique que vous associez au certificat de demande d'approvisionnement pour autoriser les appareils à s'enregistrer et à utiliser le modèle de provisionnement de flotte.

Rubriques

- [Création d'un rôle d'échange de jetons](#)
- [Création d'une AWS IoT politique](#)
- [Création d'un modèle de provisionnement de flotte](#)
- [Création d'un certificat de demande d'approvisionnement et d'une clé privée](#)

Création d'un rôle d'échange de jetons

Les appareils principaux de Greengrass utilisent un rôle de service IAM, appelé rôle d'échange de jetons, pour autoriser les appels aux services. AWS L'appareil utilise le fournisseur AWS IoT d'informations d'identification pour obtenir des AWS informations d'identification temporaires pour ce rôle, ce qui lui permet d'interagir avec Amazon Logs AWS IoT, d'envoyer des journaux à Amazon CloudWatch Logs et de télécharger des artefacts de composants personnalisés depuis Amazon S3. Pour de plus amples informations, veuillez consulter [Autoriser les appareils principaux à interagir avec les AWS services](#).

Vous utilisez un alias de AWS IoT rôle pour configurer le rôle d'échange de jetons pour les appareils principaux de Greengrass. Les alias de rôle vous permettent de modifier le rôle d'échange de jetons d'un appareil tout en conservant la même configuration. Pour plus d'informations, consultez la section [Autorisation des appels directs vers AWS des services](#) dans le Guide du AWS IoT Core développeur.

Dans cette section, vous allez créer un rôle IAM d'échange de jetons et un alias de AWS IoT rôle pointant vers le rôle. Si vous avez déjà configuré un appareil principal Greengrass, vous pouvez utiliser son rôle d'échange de jetons et son alias de rôle au lieu d'en créer de nouveaux.

Pour créer un rôle IAM d'échange de jetons

1. Créez un rôle IAM que votre appareil peut utiliser comme rôle d'échange de jetons. Procédez comme suit :
 - a. Créez un fichier contenant le document de politique de confiance requis par le rôle d'échange de jetons.

Par exemple, sur un système basé sur Linux, vous pouvez exécuter la commande suivante pour utiliser GNU nano pour créer le fichier.

```
nano device-role-trust-policy.json
```

Copiez le code JSON suivant dans le fichier.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "credentials.iot.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

- b. Créez le rôle d'échange de jetons avec le document de politique de confiance.
 - Remplacez *GreengrassV2TokenExchangeRole* par le nom du rôle IAM à créer.

```
aws iam create-role --role-name GreengrassV2TokenExchangeRole --assume-role-policy-document file:///device-role-trust-policy.json
```

La réponse ressemble à l'exemple suivant, si la demande aboutit.

```
{
  "Role": {
    "Path": "/",
    "RoleName": "GreengrassV2TokenExchangeRole",
    "RoleId": "AR0AZ2YMUHYHK50KM77FB",
    "Arn": "arn:aws:iam::123456789012:role/GreengrassV2TokenExchangeRole",
    "CreateDate": "2021-02-06T00:13:29+00:00",
    "AssumeRolePolicyDocument": {
      "Version": "2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Principal": {
            "Service": "credentials.iot.amazonaws.com"
          },
          "Action": "sts:AssumeRole"
        }
      ]
    }
  }
}
```

- c. Créez un fichier contenant le document de politique d'accès requis par le rôle d'échange de jetons.

Par exemple, sur un système basé sur Linux, vous pouvez exécuter la commande suivante pour utiliser GNU nano pour créer le fichier.

```
nano device-role-access-policy.json
```

Copiez le code JSON suivant dans le fichier.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams",

```

```

        "s3:GetBucketLocation"
    ],
    "Resource": "*"
}
]
}

```

Note

Cette politique d'accès n'autorise pas l'accès aux artefacts des composants dans les compartiments S3. Pour déployer des composants personnalisés qui définissent des artefacts dans Amazon S3, vous devez ajouter des autorisations au rôle afin de permettre à votre appareil principal de récupérer les artefacts des composants. Pour de plus amples informations, veuillez consulter [Autoriser l'accès aux compartiments S3 pour les artefacts des composants](#).

Si vous ne possédez pas encore de compartiment S3 pour les artefacts des composants, vous pouvez ajouter ces autorisations ultérieurement après avoir créé un compartiment.

d. Créez la politique IAM à partir du document de stratégie.

- Remplacez *GreengrassV2TokenExchangeRoleAccess* par le nom de la politique IAM à créer.

```
aws iam create-policy --policy-name GreengrassV2TokenExchangeRoleAccess --
policy-document file://device-role-access-policy.json
```

La réponse ressemble à l'exemple suivant, si la demande aboutit.

```

{
  "Policy": {
    "PolicyName": "GreengrassV2TokenExchangeRoleAccess",
    "PolicyId": "ANPAZ2YMUHYHACI7C5Z66",
    "Arn": "arn:aws:iam::123456789012:policy/GreengrassV2TokenExchangeRoleAccess",
    "Path": "/",
    "DefaultVersionId": "v1",
    "AttachmentCount": 0,
    "PermissionsBoundaryUsageCount": 0,

```

```
"IsAttachable": true,  
"CreateDate": "2021-02-06T00:37:17+00:00",  
"UpdateDate": "2021-02-06T00:37:17+00:00"  
}  
}
```

e. Associez la politique IAM au rôle d'échange de jetons.

- Remplacez *GreengrassV2TokenExchangeRole* par le nom du rôle IAM.
- Remplacez l'ARN de la stratégie par l'ARN de la stratégie IAM que vous avez créée à l'étape précédente.

```
aws iam attach-role-policy --role-name GreengrassV2TokenExchangeRole --policy-  
arn arn:aws:iam::123456789012:policy/GreengrassV2TokenExchangeRoleAccess
```

La commande n'a aucune sortie si la demande aboutit.

2. Créez un alias de AWS IoT rôle qui pointe vers le rôle d'échange de jetons.

- Remplacez *GreengrassCoreTokenExchangeRoleAlias* par le nom de l'alias de rôle à créer.
- Remplacez l'ARN du rôle par l'ARN du rôle IAM que vous avez créé à l'étape précédente.

```
aws iot create-role-alias --role-alias GreengrassCoreTokenExchangeRoleAlias --role-  
arn arn:aws:iam::123456789012:role/GreengrassV2TokenExchangeRole
```

La réponse ressemble à l'exemple suivant, si la demande aboutit.

```
{  
  "roleAlias": "GreengrassCoreTokenExchangeRoleAlias",  
  "roleAliasArn": "arn:aws:iot:us-west-2:123456789012:rolealias/  
GreengrassCoreTokenExchangeRoleAlias"  
}
```

Note

Pour créer un alias de rôle, vous devez être autorisé à transmettre le rôle IAM d'échange de jetons à AWS IoT. Si vous recevez un message d'erreur lorsque vous essayez de

créer un alias de rôle, vérifiez que votre AWS utilisateur dispose de cette autorisation. Pour plus d'informations, consultez la section [Octroi à un utilisateur des autorisations lui permettant de transférer un rôle à un AWS service](#) dans le Guide de Gestion des identités et des accès AWS l'utilisateur.

Création d'une AWS IoT politique

Une fois que vous avez enregistré un appareil en tant qu' AWS IoT objet, celui-ci peut utiliser un certificat numérique pour s'authentifier. AWS Ce certificat inclut une ou plusieurs AWS IoT politiques qui définissent les autorisations qu'un appareil peut utiliser avec le certificat. Ces politiques permettent à l'appareil de communiquer avec AWS IoT et AWS IoT Greengrass.

Dans le AWS IoT cadre du provisionnement du parc, les appareils se connectent AWS IoT pour créer et télécharger un certificat d'appareil. Dans le modèle de provisionnement de flotte que vous créez dans la section suivante, vous pouvez spécifier s'il AWS IoT attache la même AWS IoT politique aux certificats de tous les appareils ou s'il en crée une nouvelle pour chaque appareil.

Dans cette section, vous allez créer une AWS IoT politique qui AWS IoT s'attache aux certificats de tous les appareils. Grâce à cette approche, vous pouvez gérer les autorisations pour tous les appareils en tant que flotte. Si vous préférez créer une nouvelle AWS IoT politique pour chaque appareil, vous pouvez ignorer cette section et vous référer à la politique qu'elle contient lorsque vous définissez votre modèle de flotte.

Pour créer une AWS IoT politique

- Créez une AWS IoT politique qui définit les AWS IoT autorisations pour votre parc d'appareils principaux Greengrass. La politique suivante permet d'accéder à tous les sujets MQTT et aux opérations Greengrass, afin que votre appareil fonctionne avec les applications personnalisées et les modifications futures qui nécessitent de nouvelles opérations Greengrass. Cette politique autorise également `iot:AssumeRoleWithCertificate` autorisation, qui permet à vos appareils d'utiliser le rôle d'échange de jetons que vous avez créé dans la section précédente. Vous pouvez restreindre cette politique en fonction de votre cas d'utilisation. Pour de plus amples informations, veuillez consulter [AWS IoT Politique minimale pour les appareils AWS IoT Greengrass V2 principaux](#).

Procédez comme suit :

- a. Créez un fichier contenant le document de AWS IoT politique dont les appareils principaux de Greengrass ont besoin.

Par exemple, sur un système basé sur Linux, vous pouvez exécuter la commande suivante pour utiliser GNU nano pour créer le fichier.

```
nano greengrass-v2-iot-policy.json
```

Copiez le code JSON suivant dans le fichier.

- Remplacez la `iot:AssumeRoleWithCertificate` ressource par l'ARN de l'alias de AWS IoT rôle que vous avez créé dans la section précédente.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Publish",
        "iot:Subscribe",
        "iot:Receive",
        "iot:Connect",
        "greengrass:*"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": "iot:AssumeRoleWithCertificate",
      "Resource": "arn:aws:iot:us-east-1:123456789012:rolealias/  
GreengrassCoreTokenExchangeRoleAlias"
    }
  ]
}
```

- b. Créez une AWS IoT politique à partir du document de stratégie.
- Remplacez *GreengrassV2IoTThingPolicy* par le nom de la politique à créer.

```
aws iot create-policy --policy-name GreengrassV2IoTThingPolicy --policy-document file://greengrass-v2-iot-policy.json
```

La réponse ressemble à l'exemple suivant, si la demande aboutit.

JSON

```
{
  "policyName": "GreengrassV2IoTThingPolicy",
  "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/GreengrassV2IoTThingPolicy",
  "policyDocument": "{
    \"Version\": \"2012-10-17\",
    \"Statement\": [
      {
        \"Effect\": \"Allow\",
        \"Action\": [
          \"iot:Publish\",
          \"iot:Subscribe\",
          \"iot:Receive\",
          \"iot:Connect\",
          \"greengrass:*\"
        ],
        \"Resource\": [
          \"*\"
        ]
      },
      {
        \"Effect\": \"Allow\",
        \"Action\": \"iot:AssumeRoleWithCertificate\",
        \"Resource\": \"arn:aws:iot:us-west-2:123456789012:rolealias/GreengrassCoreTokenExchangeRoleAlias\"
      }
    ]
  }",
  "policyVersionId": "1"
```

```
}
```

Création d'un modèle de provisionnement de flotte

AWS IoT les modèles de provisionnement de flotte définissent comment provisionner AWS IoT les objets, les politiques et les certificats. Pour approvisionner les appareils principaux de Greengrass avec le plug-in de provisionnement de flotte, vous devez créer un modèle qui spécifie les éléments suivants :

- N'importe AWS IoT quel objet, ressource. Vous pouvez spécifier une liste de groupes d'objets existants pour déployer des composants sur chaque appareil lors de sa mise en ligne.
- Une ressource AWS IoT en matière de politiques. Cette ressource peut définir l'une des propriétés suivantes :
 - Le nom d'une AWS IoT politique existante. Si vous choisissez cette option, les principaux appareils que vous créez à partir de ce modèle utilisent la même AWS IoT politique et vous pouvez gérer leurs autorisations en tant que flotte.
 - Un document AWS IoT de politique. Si vous choisissez cette option, chaque périphérique principal que vous créez à partir de ce modèle utilise une AWS IoT politique unique, et vous pouvez gérer les autorisations pour chaque périphérique principal individuel.
- Une ressource AWS IoT de certificat. Cette ressource de certificat doit utiliser le `AWS::IoT::Certificate::Id` paramètre pour associer le certificat au périphérique principal. Pour plus d'informations, consultez la section [Just-in-time Provisioning](#) dans le Guide du AWS IoT développeur.

Dans le modèle, vous pouvez spécifier d'ajouter l' AWS IoT objet à une liste de groupes d'objets existants. Lorsque l'appareil principal se connecte AWS IoT Greengrass pour la première fois, il reçoit des déploiements Greengrass pour chaque groupe d'objets dont il est membre. Vous pouvez utiliser des groupes d'objets pour déployer les derniers logiciels sur chaque appareil dès sa mise en ligne. Pour de plus amples informations, veuillez consulter [Déployer AWS IoT Greengrass des composants sur des appareils](#).

Le AWS IoT service nécessite des autorisations pour créer et mettre à jour AWS IoT des ressources dans vos appareils Compte AWS lorsque vous approvisionnez des appareils. Pour donner accès au AWS IoT service, vous devez créer un rôle IAM et le fournir lors de la création du modèle. AWS IoT fournit une politique gérée, l'[AWSIoTThingsenregistration](#), qui permet d'accéder à toutes les autorisations AWS IoT susceptibles d'être utilisées lors du provisionnement des appareils. Vous

pouvez utiliser cette politique gérée ou créer une politique personnalisée qui limite les autorisations de la politique gérée pour votre cas d'utilisation.

Dans cette section, vous créez un rôle IAM qui permet AWS IoT de fournir des ressources pour les appareils, et vous créez un modèle de provisionnement de flotte qui utilise ce rôle IAM.

Pour créer un modèle de provisionnement de flotte

1. Créez un rôle IAM AWS IoT capable de fournir des ressources dans votre Compte AWS.

Procédez comme suit :

- a. Créez un fichier contenant le document de politique de confiance qui permet AWS IoT d'assumer le rôle.

Par exemple, sur un système basé sur Linux, vous pouvez exécuter la commande suivante pour utiliser GNU nano pour créer le fichier.

```
nano aws-iot-trust-policy.json
```

Copiez le code JSON suivant dans le fichier.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "iot.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

- b. Créez un rôle IAM avec le document de politique de confiance.

- Remplacez *GreengrassFleetProvisioningRole* par le nom du rôle IAM à créer.

```
aws iam create-role --role-name GreengrassFleetProvisioningRole --assume-role-policy-document file://aws-iot-trust-policy.json
```

La réponse ressemble à l'exemple suivant, si la demande aboutit.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Connect",
        "iot:Publish",
        "iot:Subscribe",
        "iot:Receive"
      ],
      "Resource": "*"
    }
  ]
}
```

- c. Consultez la politique [AWSIoTThingsd'enregistrement](#), qui permet d'accéder à toutes les autorisations AWS IoT susceptibles d'être utilisées lors de l'approvisionnement des appareils. Vous pouvez utiliser cette politique gérée ou créer une politique personnalisée qui définit des autorisations limitées pour votre cas d'utilisation. Si vous choisissez de créer une politique personnalisée, faites-le maintenant.
- d. Associez la politique IAM au rôle de provisionnement de la flotte.
 - Remplacez *GreengrassFleetProvisioningRole* par le nom du rôle IAM.
 - Si vous avez créé une stratégie personnalisée à l'étape précédente, remplacez l'ARN de la stratégie par l'ARN de la stratégie IAM à utiliser.

```
aws iam attach-role-policy --role-name GreengrassFleetProvisioningRole --  
policy-arn arn:aws:iam::aws:policy/service-role/AWSIoTThingsRegistration
```

La commande n'a aucune sortie si la demande aboutit.

2. (Facultatif) Créez un hook de pré-approvisionnement, qui est une AWS Lambda fonction qui valide les paramètres du modèle fournis par les appareils lors de l'enregistrement. Vous pouvez utiliser un hook de préprovisionnement pour mieux contrôler quels appareils et combien d'appareils sont intégrés dans votre. Compte AWS Pour plus d'informations, consultez la section [Pre-provisioning hooks](#) dans le Guide du AWS IoT Core développeur.
3. Créez un modèle de provisionnement de flotte. Procédez comme suit :
 - a. Créez un fichier contenant le modèle de document de provisionnement.

Par exemple, sur un système basé sur Linux, vous pouvez exécuter la commande suivante pour utiliser GNU nano pour créer le fichier.

```
nano greengrass-fleet-provisioning-template.json
```

Rédigez le modèle de document de provisionnement. Vous pouvez commencer par l'exemple de modèle de provisionnement suivant, qui indique de créer un AWS IoT objet doté des propriétés suivantes :

- Le nom de l'objet est la valeur que vous spécifiez dans le paramètre du ThingName modèle.
- L'objet est membre du groupe d'objets que vous spécifiez dans le paramètre du ThingGroupName modèle. Le groupe d'objets doit exister dans votre Compte AWS.
- Le certificat de l'objet est GreengrassV2IoTThingPolicy associé à la AWS IoT politique nommée.

Pour plus d'informations, consultez la section [Modèles de provisionnement](#) dans le Guide du AWS IoT Core développeur.

```
{  
  "Parameters": {  
    "ThingName": {
```

```
    "Type": "String"
  },
  "ThingGroupName": {
    "Type": "String"
  },
  "AWS::IoT::Certificate::Id": {
    "Type": "String"
  }
},
"Resources": {
  "MyThing": {
    "OverrideSettings": {
      "AttributePayload": "REPLACE",
      "ThingGroups": "REPLACE",
      "ThingTypeName": "REPLACE"
    },
    "Properties": {
      "AttributePayload": {},
      "ThingGroups": [
        {
          "Ref": "ThingGroupName"
        }
      ],
      "ThingName": {
        "Ref": "ThingName"
      }
    },
    "Type": "AWS::IoT::Thing"
  },
  "MyPolicy": {
    "Properties": {
      "PolicyName": "GreengrassV2IoTThingPolicy"
    },
    "Type": "AWS::IoT::Policy"
  },
  "MyCertificate": {
    "Properties": {
      "CertificateId": {
        "Ref": "AWS::IoT::Certificate::Id"
      },
      "Status": "Active"
    },
    "Type": "AWS::IoT::Certificate"
  }
}
```

```
}
}
```

Note

MyThingMyPolicy, et *MyCertificate* sont des noms arbitraires qui identifient chaque spécification de ressource dans le modèle de provisionnement du parc. AWS IoT n'utilise pas ces noms dans les ressources qu'il crée à partir du modèle. Vous pouvez utiliser ces noms ou les remplacer par des valeurs qui vous aident à identifier chaque ressource du modèle.

- b. Créez le modèle de provisionnement de flotte à partir du document modèle de provisionnement.
 - Remplacez *GreengrassFleetProvisioningTemplate* par le nom du modèle à créer.
 - Remplacez la description du modèle par une description de votre modèle.
 - Remplacez l'ARN du rôle de provisionnement par l'ARN du rôle que vous avez créé précédemment.

Linux or Unix

```
aws iot create-provisioning-template \
  --template-name GreengrassFleetProvisioningTemplate \
  --description "A provisioning template for Greengrass core devices." \
  --provisioning-role-arn "arn:aws:iam::123456789012:role/  
GreengrassFleetProvisioningRole" \
  --template-body file://greengrass-fleet-provisioning-template.json \
  --enabled
```

Windows Command Prompt (CMD)

```
aws iot create-provisioning-template ^
  --template-name GreengrassFleetProvisioningTemplate ^
  --description "A provisioning template for Greengrass core devices." ^
  --provisioning-role-arn "arn:aws:iam::123456789012:role/  
GreengrassFleetProvisioningRole" ^
  --template-body file://greengrass-fleet-provisioning-template.json ^
  --enabled
```

PowerShell

```
aws iot create-provisioning-template `
  --template-name GreengrassFleetProvisioningTemplate `
  --description "A provisioning template for Greengrass core devices." `
  --provisioning-role-arn "arn:aws:iam::123456789012:role/  
GreengrassFleetProvisioningRole" `
  --template-body file://greengrass-fleet-provisioning-template.json `
  --enabled
```

Note

Si vous avez créé un hook de pré-provisionnement, spécifiez l'ARN de la fonction Lambda du hook de pré-provisionnement avec l'argument. `--pre-provisioning-hook`

```
--pre-provisioning-hook targetArn=arn:aws:lambda:us-  
west-2:123456789012:function:GreengrassPreProvisioningHook
```

La réponse ressemble à l'exemple suivant, si la demande aboutit.

```
{
  "templateArn": "arn:aws:iot:us-west-2:123456789012:provisioningtemplate/
GreengrassFleetProvisioningTemplate",
  "templateName": "GreengrassFleetProvisioningTemplate",
  "defaultVersionId": 1
}
```

Création d'un certificat de demande d'approvisionnement et d'une clé privée

Les certificats de réclamation sont des certificats X.509 qui permettent aux appareils de s'enregistrer en tant qu' AWS IoT objets et de récupérer un certificat d'appareil X.509 unique à utiliser pour les opérations régulières. Après avoir créé un certificat de réclamation, vous joignez une AWS IoT politique qui permet aux appareils de l'utiliser pour créer des certificats d'appareils uniques et approvisionner à l'aide d'un modèle de provisionnement de flotte. Les appareils dotés du

certificat de réclamation peuvent effectuer le provisionnement en utilisant uniquement le modèle de provisionnement que vous autorisez dans la AWS IoT politique.

Dans cette section, vous créez le certificat de réclamation et vous le configurez pour les appareils à utiliser avec le modèle de provisionnement de flotte que vous avez créé dans la section précédente.

Important

L'approvisionnement des clés privées des réclamations doit être sécurisé à tout moment, y compris sur les appareils principaux de Greengrass. Nous vous recommandons d'utiliser CloudWatch les statistiques et les journaux Amazon pour détecter les signes d'utilisation abusive, tels que l'utilisation non autorisée du certificat de réclamation pour approvisionner des appareils. Si vous détectez une utilisation abusive, désactivez le certificat de demande d'approvisionnement afin qu'il ne puisse pas être utilisé pour le provisionnement des appareils. Pour plus d'informations, consultez la section [Surveillance AWS IoT](#) dans le guide du AWS IoT Core développeur.

Pour vous aider à mieux gérer le nombre d'appareils et les appareils qui s'enregistrent dans le vôtre Compte AWS, vous pouvez spécifier un crochet de préapprovisionnement lorsque vous créez un modèle de provisionnement de flotte. Un hook de pré-provisionnement est une AWS Lambda fonction qui valide les paramètres du modèle fournis par les appareils lors de l'enregistrement. Par exemple, vous pouvez créer un hook de pré-provisionnement qui compare l'identifiant d'un appareil à une base de données afin de vérifier que l'appareil est autorisé à effectuer le provisionnement. Pour plus d'informations, consultez la section [Pre-provisioning hooks](#) dans le Guide du AWS IoT Core développeur.

Pour créer un certificat de demande d'approvisionnement et une clé privée

1. Créez un dossier dans lequel vous téléchargerez le certificat de réclamation et la clé privée.

```
mkdir claim-certs
```

2. Créez et enregistrez un certificat et une clé privée à utiliser pour le provisionnement. AWS IoT fournit des certificats clients signés par l'autorité de certification Amazon Root (CA).

Linux or Unix

```
aws iot create-keys-and-certificate \  
  --certificate-pem-outfile "claim-certs/claim.pem.crt" \  
  --private-key-pem-outfile "claim-certs/claim.pem.key" \  
  --certificate-authority "arn:aws:iam::aws:root" \  
  --role-arn "arn:aws:iam::aws:role/aws-logs-role-aws-logs" \  
  --key-usage "keyEncipherment, keyAgreement, keyWrap, keySignature, digitalSignature" \  
  --certificate-validity 3650
```

```
--public-key-outfile "claim-certs/claim.public.pem.key" \  
--private-key-outfile "claim-certs/claim.private.pem.key" \  
--set-as-active
```

Windows Command Prompt (CMD)

```
aws iot create-keys-and-certificate ^  
--certificate-pem-outfile "claim-certs/claim.pem.crt" ^  
--public-key-outfile "claim-certs/claim.public.pem.key" ^  
--private-key-outfile "claim-certs/claim.private.pem.key" ^  
--set-as-active
```

PowerShell

```
aws iot create-keys-and-certificate `   
--certificate-pem-outfile "claim-certs/claim.pem.crt" `   
--public-key-outfile "claim-certs/claim.public.pem.key" `   
--private-key-outfile "claim-certs/claim.private.pem.key" `   
--set-as-active
```

La réponse contient des informations sur le certificat, si la demande aboutit. Enregistrez l'ARN du certificat pour l'utiliser ultérieurement.

3. Créez et attachez une AWS IoT politique qui permet aux appareils d'utiliser le certificat pour créer des certificats d'appareils uniques et de les approvisionner à l'aide du modèle de provisionnement du parc. La politique suivante autorise l'accès à l'API MQTT de provisionnement des appareils. Pour plus d'informations, consultez la section [Device Provisioning MQTT API](#) dans le Guide du AWS IoT Core développeur.

Procédez comme suit :

- a. Créez un fichier contenant le document de AWS IoT politique dont les appareils principaux de Greengrass ont besoin.

Par exemple, sur un système basé sur Linux, vous pouvez exécuter la commande suivante pour utiliser GNU nano pour créer le fichier.

```
nano greengrass-provisioning-claim-iot-policy.json
```

Copiez le code JSON suivant dans le fichier.

- Remplacez chaque instance de *region* par celle Région AWS où vous avez configuré le provisionnement de la flotte.
- Remplacez chaque instance de *account-id* par votre Compte AWS identifiant.
- Remplacez chaque instance de *GreengrassFleetProvisioningTemplate* par le nom du modèle de provisionnement de flotte que vous avez créé dans la section précédente.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iot:Connect",
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:Publish",
        "iot:Receive"
      ],
      "Resource": [
        "arn:aws:iot:us-east-1:123456789012:topic/$aws/certificates/create/*",
        "arn:aws:iot:us-east-1:123456789012:topic/$aws/provisioning-templates/GreengrassFleetProvisioningTemplate/provision/*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": "iot:Subscribe",
      "Resource": [
        "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/certificates/create/*",
        "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/provisioning-templates/GreengrassFleetProvisioningTemplate/provision/*"
      ]
    }
  ]
}
```

```
]
}
```

b. Créez une AWS IoT politique à partir du document de stratégie.

- Remplacez *GreengrassProvisioningClaimPolicy* par le nom de la politique à créer.

```
aws iot create-policy --policy-name GreengrassProvisioningClaimPolicy --policy-
document file://greengrass-provisioning-claim-iot-policy.json
```

La réponse ressemble à l'exemple suivant, si la demande aboutit.

JSON

```
{
  "policyName": "GreengrassProvisioningClaimPolicy",
  "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassProvisioningClaimPolicy",
  "policyDocument": "{
    \"Version\": \"2012-10-17\",
    \"Statement\": [
      {
        \"Effect\": \"Allow\",
        \"Action\": \"iot:Connect\",
        \"Resource\": \"*\"
      },
      {
        \"Effect\": \"Allow\",
        \"Action\": [
          \"iot:Publish\",
          \"iot:Receive\"
        ],
        \"Resource\": [
          \"arn:aws:iot:us-east-1:123456789012:topic/$aws/certificates/
create/*\",
          \"arn:aws:iot:us-east-1:123456789012:topic/$aws/provisioning-
templates/GreengrassFleetProvisioningTemplate/provision/*\"
        ]
      }
    ]
  }
```

```

    \"Effect\": \"Allow\",
    \"Action\": \"iot:Subscribe\",
    \"Resource\": [
      \"arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/
certificates/create/*\",
      \"arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/provisioning-
templates/GreengrassFleetProvisioningTemplate/provision/*\"
    ]
  }
}
}],
\"policyVersionId\": \"1\"
}

```

4. Joignez la AWS IoT politique au certificat de réclamation d'approvisionnement.

- Remplacez *GreengrassProvisioningClaimPolicy* par le nom de la politique à joindre.
- Remplacez l'ARN cible par l'ARN du certificat de demande d'approvisionnement.

```

aws iot attach-policy --policy-name GreengrassProvisioningClaimPolicy --
target arn:aws:iot:us-west-2:123456789012:cert/
aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4

```

La commande n'a aucune sortie si la demande aboutit.

Vous disposez désormais d'un certificat de demande d'approvisionnement et d'une clé privée que les appareils peuvent utiliser pour s'enregistrer AWS IoT et s'approvisionner en tant qu'appareils principaux de Greengrass. Vous pouvez intégrer le certificat de réclamation et la clé privée dans les appareils pendant la fabrication, ou copier le certificat et la clé sur les appareils avant d'installer le logiciel AWS IoT Greengrass Core. Pour de plus amples informations, veuillez consulter [Installation AWS IoT Greengrass du logiciel de base avec provisionnement du AWS IoT parc](#).

Configuration du plugin de provisionnement de AWS IoT flotte

Le plugin de provisionnement de AWS IoT flotte fournit les paramètres de configuration suivants que vous pouvez personnaliser lorsque vous [installez le logiciel AWS IoT Greengrass Core avec le provisionnement de flotte](#).

rootPath

Le chemin d'accès au dossier à utiliser comme racine pour le logiciel AWS IoT Greengrass Core.

awsRegion

Celui Région AWS que le plugin de provisionnement de flotte utilise pour provisionner les AWS ressources.

iotDataEndpoint

Le point AWS IoT de terminaison de données pour votre Compte AWS.

iotCredentialEndpoint

Le point de terminaison des informations d' AWS IoT identification de votre Compte AWS.

iotRoleAlias

Alias de AWS IoT rôle qui pointe vers un rôle IAM d'échange de jetons. Le fournisseur AWS IoT d'informations d'identification assume ce rôle pour permettre au dispositif principal de Greengrass d'interagir avec AWS les services. Pour de plus amples informations, veuillez consulter [Autoriser les appareils principaux à interagir avec les AWS services](#).

provisioningTemplate

Modèle de provisionnement de AWS IoT flotte à utiliser pour provisionner les AWS ressources. Ce modèle doit spécifier les éléments suivants :

- N'importe AWS IoT quel objet, ressource. Vous pouvez spécifier une liste de groupes d'objets existants pour déployer des composants sur chaque appareil lors de sa mise en ligne.
- Une ressource AWS IoT en matière de politiques. Cette ressource peut définir l'une des propriétés suivantes :
 - Le nom d'une AWS IoT politique existante. Si vous choisissez cette option, les principaux appareils que vous créez à partir de ce modèle utilisent la même AWS IoT politique, et vous pouvez gérer leurs autorisations en tant que flotte.
 - Un document AWS IoT de politique. Si vous choisissez cette option, chaque périphérique principal que vous créez à partir de ce modèle utilise une AWS IoT politique unique, et vous pouvez gérer les autorisations pour chaque périphérique principal individuel.
- Une ressource AWS IoT de certificat. Cette ressource de certificat doit utiliser le `AWS::IoT::Certificate::Id` paramètre pour associer le certificat au périphérique principal. Pour plus d'informations, consultez la section [Just-in-time Provisioning](#) dans le Guide du AWS IoT développeur.

Pour plus d'informations, consultez la section [Modèles de provisionnement](#) dans le Guide du AWS IoT Core développeur.

`claimCertificatePath`

Le chemin d'accès au certificat de demande de provisionnement pour le modèle de provisionnement que vous spécifiez dans `provisioningTemplate`. Pour plus d'informations, consultez [CreateProvisioningClaim](#) dans la Référence d'API AWS IoT Core .

`claimCertificatePrivateKeyPath`

Le chemin d'accès à la clé privée du certificat de demande d'approvisionnement pour le modèle de provisionnement que vous spécifiez dans `provisioningTemplate`. Pour plus d'informations, consultez [CreateProvisioningClaim](#) dans la Référence d'API AWS IoT Core .

Important

L'approvisionnement des clés privées des réclamations doit être sécurisé à tout moment, y compris sur les appareils principaux de Greengrass. Nous vous recommandons d'utiliser CloudWatch les statistiques et les journaux Amazon pour détecter les signes d'utilisation abusive, tels que l'utilisation non autorisée du certificat de réclamation pour approvisionner des appareils. Si vous détectez une utilisation abusive, désactivez le certificat de demande d'approvisionnement afin qu'il ne puisse pas être utilisé pour le provisionnement des appareils. Pour plus d'informations, consultez la section [Surveillance AWS IoT](#) dans le guide du AWS IoT Core développeur.

Pour vous aider à mieux gérer le nombre d'appareils et les appareils qui s'enregistrent dans le vôtre Compte AWS, vous pouvez spécifier un crochet de préapprovisionnement lorsque vous créez un modèle de provisionnement de flotte. Un hook de pré-provisionnement est une AWS Lambda fonction qui valide les paramètres du modèle fournis par les appareils lors de l'enregistrement. Par exemple, vous pouvez créer un hook de pré-provisionnement qui compare l'identifiant d'un appareil à une base de données afin de vérifier que l'appareil est autorisé à effectuer le provisionnement. Pour plus d'informations, consultez la section [Pre-provisioning hooks](#) dans le Guide du AWS IoT Core développeur.

`rootCaPath`

Le chemin d'accès au certificat de l'autorité de certification racine (CA) Amazon.

templateParameters

(Facultatif) Carte des paramètres à fournir au modèle de provisionnement de flotte. Pour plus d'informations, consultez la [section sur les paramètres des modèles de provisionnement](#) dans le Guide du AWS IoT Core développeur.

deviceId

(Facultatif) L'identifiant de l'appareil à utiliser comme identifiant client lorsque le plugin de provisionnement de flotte crée une connexion MQTT avec AWS IoT.

Par défaut : un UUID aléatoire.

mqttPort

(Facultatif) Le port à utiliser pour les connexions MQTT.

Par défaut : 8883

proxyUrl

(Facultatif) L'URL du serveur proxy au format `scheme://userinfo@host:port`. Pour utiliser un proxy HTTPS, vous devez utiliser la version 1.1.0 ou ultérieure du plugin de provisionnement de flotte.

- `scheme`— Le schéma, qui doit être `http` ou `https`.

Important

Les appareils principaux de Greengrass doivent exécuter [Greengrass nucleus](#) v2.5.0 ou version ultérieure pour utiliser les proxys HTTPS.

Si vous configurez un proxy HTTPS, vous devez ajouter le certificat CA du serveur proxy au certificat CA racine Amazon de l'appareil principal. Pour de plus amples informations, veuillez consulter [Permettre au périphérique principal de faire confiance à un proxy HTTPS](#).

- `userinfo`— (Facultatif) Les informations relatives au nom d'utilisateur et au mot de passe. Si vous spécifiez ces informations dans `url`, le périphérique principal de Greengrass ignore les `username` et `password` champs.
- `host`— Le nom d'hôte ou l'adresse IP du serveur proxy.
- `port`— (Facultatif) Le numéro de port. Si vous ne spécifiez pas le port, le périphérique principal de Greengrass utilise les valeurs par défaut suivantes :

- http— 80
- https— 443

proxyUserName

(Facultatif) Le nom d'utilisateur qui authentifie le serveur proxy.

proxyPassword

(Facultatif) Le mot de passe qui authentifie le serveur proxy.

Parcours CSR

(Facultatif) Le chemin d'accès au fichier de demande de signature de certificat (CSR) à utiliser pour créer le certificat de l'appareil à partir d'un CSR. Pour plus d'informations, consultez la section [Provisioning by claim](#) dans le guide du AWS IoT Core développeur.

csrPrivateKeyParcours

(Facultatif, obligatoire si `csrPath` est déclaré) Le chemin d'accès à la clé privée utilisée pour générer le CSR. La clé privée doit avoir été utilisée pour générer le CSR. Pour plus d'informations, consultez la section [Provisioning by claim](#) dans le guide du AWS IoT Core développeur.

Chemin du certificat

(Facultatif) Le chemin à utiliser pour enregistrer le certificat de l'appareil téléchargé.

privateKeyPath

(Facultatif) Le chemin à utiliser pour enregistrer la clé privée de l'appareil téléchargé.

AWS IoT journal des modifications du plugin de provisionnement de flotte

Le tableau suivant décrit les modifications apportées à chaque version du provisionnement de la AWS IoT flotte par claim plugin (`aws.greengrass.FleetProvisioningByClaim`).

Version	Modifications
1.2.2	Corrections de bugs et améliorations • Ajoute la prise en charge des chemins personnalisés de certificat de périphérique principal (<code>certificatePath</code>) et de clé privée (<code>privateKeyPath</code>).

Version	Modifications
1.2.1	Corrections de bugs et améliorations • Résout un problème en raison duquel le plug-in de provisionnement de flotte est hors ligne lors du démarrage du noyau Greengrass. Le plugin de provisionnement de flotte réessaie désormais indéfiniment les appels MQTT connect.
1.2.0	Corrections de bugs et améliorations • Prend en charge le provisionnement des appareils via une demande de signature de certificat avec un chemin de clé privée configurable. • Corrections et améliorations mineures.
1.1.0	Corrections de bugs et améliorations • Ajoute la prise en charge de formats de chemin de fichier supplémentaires lorsque vous configurez le plug-in sur des appareils Windows. • Ajoute la prise en charge des configurations de proxy réseau HTTPS. Pour plus d'informations, consultez Connexion au port 443 ou via un proxy réseau et Permettre au périphérique principal de faire confiance à un proxy HTTPS.
1.0.0	Première version.

Installez le logiciel AWS IoT Greengrass Core avec un provisionnement personnalisé des ressources

Cette fonctionnalité est disponible pour les versions 2.4.0 et ultérieures du composant [Greengrass](#) nucleus.

Le programme d'installation du logiciel AWS IoT Greengrass Core fournit une interface Java que vous pouvez implémenter dans un plugin personnalisé qui fournit les AWS ressources nécessaires. Vous pouvez développer un plug-in de provisionnement pour utiliser des certificats clients X.509 personnalisés ou pour exécuter des étapes de provisionnement complexes non prises en charge par d'autres processus d'installation. Pour plus d'informations, consultez la section [Création de vos propres certificats clients](#) dans le Guide du AWS IoT Core développeur.

Pour exécuter un plug-in de provisionnement personnalisé lorsque vous installez le logiciel AWS IoT Greengrass Core, vous devez créer un fichier JAR que vous fournissez au programme d'installation. Le programme d'installation exécute le plug-in, qui renvoie une configuration d'approvisionnement qui définit les AWS ressources pour le périphérique principal de Greengrass. Le programme d'installation utilise ces informations pour configurer le logiciel AWS IoT Greengrass Core sur l'appareil. Pour de plus amples informations, veuillez consulter [Développez des plugins de provisionnement personnalisés](#).

Important

Avant de télécharger le logiciel AWS IoT Greengrass Core, vérifiez que votre appareil principal répond à la [configuration requise](#) pour installer et exécuter le logiciel AWS IoT Greengrass Core v2.0.

Rubriques

- [Conditions préalables](#)
- [Configuration de l'environnement de l'appareil](#)
- [Téléchargez le logiciel AWS IoT Greengrass Core](#)
- [Installation du logiciel AWS IoT Greengrass Core](#)
- [Développez des plugins de provisionnement personnalisés](#)

Conditions préalables

Pour installer le logiciel AWS IoT Greengrass Core avec un provisionnement personnalisé, vous devez disposer des éléments suivants :

- Un fichier JAR pour un plugin de provisionnement personnalisé qui implémente `leDeviceIdentityInterface`. Le plug-in de provisionnement personnalisé doit renvoyer des valeurs pour chaque paramètre de configuration du système et du noyau. Dans le cas contraire, vous devez fournir ces valeurs dans le fichier de configuration lors de l'installation. Pour de plus amples informations, veuillez consulter [Développez des plugins de provisionnement personnalisés](#).

Configuration de l'environnement de l'appareil

Suivez les étapes décrites dans cette section pour configurer un appareil Linux ou Windows à utiliser comme périphérique AWS IoT Greengrass principal.

Configuration d'un appareil Linux

Pour configurer un appareil Linux pour AWS IoT Greengrass V2

1. Installez le moteur d'exécution Java, dont le logiciel AWS IoT Greengrass Core a besoin pour fonctionner. Nous vous recommandons d'utiliser les versions de support à long terme d'[Amazon Corretto](#) ou d'OpenJDK. La version 8 ou supérieure est requise. Les commandes suivantes vous montrent comment installer OpenJDK sur votre appareil.

- Pour les distributions basées sur Debian ou Ubuntu :

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

- Pour les distributions basées sur Red Hat :

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

- Dans Amazon Linux 2 :

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

- Pour Amazon Linux 2023 :

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

Lorsque l'installation est terminée, exécutez la commande suivante pour vérifier que Java s'exécute sur votre appareil Linux.

```
java -version
```

La commande affiche la version de Java exécutée sur le périphérique. Par exemple, sur une distribution basée sur Debian, le résultat peut ressembler à celui de l'exemple suivant.

```
openjdk version "11.0.9.1" 2020-11-04
```

```
OpenJDK Runtime Environment (build 11.0.9.1+1-post-Debian-1deb10u2)
OpenJDK 64-Bit Server VM (build 11.0.9.1+1-post-Debian-1deb10u2, mixed mode)
```

2. (Facultatif) Créez l'utilisateur système et le groupe par défaut qui exécutent les composants sur le périphérique. Vous pouvez également choisir de laisser le programme d'installation du logiciel AWS IoT Greengrass Core créer cet utilisateur et ce groupe lors de l'installation avec l'argument `--component-default-user` installer. Pour de plus amples informations, veuillez consulter [Arguments d'installation](#).

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

3. Vérifiez que l'utilisateur qui exécute le logiciel AWS IoT Greengrass Core (généralement `root`) est autorisé à exécuter `sudo` avec n'importe quel utilisateur et n'importe quel groupe.

- a. Exécutez la commande suivante pour ouvrir le `/etc/sudoers` fichier.

```
sudo visudo
```

- b. Vérifiez que l'autorisation accordée à l'utilisateur ressemble à l'exemple suivant.

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

4. (Facultatif) Pour [exécuter des fonctions Lambda conteneurisées](#), vous devez activer [cgroups](#) v1, et vous devez activer et monter les cgroups de mémoire et de périphériques. Si vous ne prévoyez pas d'exécuter des fonctions Lambda conteneurisées, vous pouvez ignorer cette étape.

Pour activer ces options cgroups, démarrez le périphérique avec les paramètres du noyau Linux suivants.

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

Pour plus d'informations sur l'affichage et la définition des paramètres du noyau de votre appareil, consultez la documentation de votre système d'exploitation et de votre chargeur de démarrage. Suivez les instructions pour définir définitivement les paramètres du noyau.

5. Installez toutes les autres dépendances requises sur votre appareil, comme indiqué dans la liste des exigences figurant dans [Exigences relatives aux dispositifs](#).

Configuration d'un appareil Windows

Note

Cette fonctionnalité est disponible pour les versions 2.5.0 et ultérieures du composant [Greengrass](#) nucleus.

Pour configurer un appareil Windows pour AWS IoT Greengrass V2

1. Installez le moteur d'exécution Java, dont le logiciel AWS IoT Greengrass Core a besoin pour fonctionner. Nous vous recommandons d'utiliser les versions de support à long terme d'[Amazon Corretto](#) ou d'OpenJDK. La version 8 ou supérieure est requise.
2. Vérifiez si Java est disponible sur la variable système [PATH](#), et ajoutez-le dans le cas contraire. Le LocalSystem compte exécute le logiciel AWS IoT Greengrass Core. Vous devez donc ajouter Java à la variable système PATH au lieu de la variable utilisateur PATH pour votre utilisateur. Procédez comme suit :
 - a. Appuyez sur la touche Windows pour ouvrir le menu de démarrage.
 - b. Tapez **environment variables** pour rechercher les options du système dans le menu Démarrer.
 - c. Dans les résultats de recherche du menu Démarrer, choisissez Modifier les variables d'environnement du système pour ouvrir la fenêtre des propriétés du système.
 - d. Choisissez les variables d'environnement... pour ouvrir la fenêtre des variables d'environnement.
 - e. Sous Variables système, sélectionnez Chemin, puis Modifier. Dans la fenêtre Modifier la variable d'environnement, vous pouvez afficher chaque chemin sur une ligne distincte.
 - f. Vérifiez si le chemin d'accès au bin dossier d'installation de Java est présent. Le chemin peut ressembler à celui de l'exemple suivant.

```
C:\\Program Files\\Amazon Corretto\\jdk11.0.13_8\\bin
```
 - g. Si le bin dossier de l'installation Java est absent de Path, choisissez Nouveau pour l'ajouter, puis OK.
3. Ouvrez l'invite de commande Windows (cmd.exe) en tant qu'administrateur.
4. Créez l'utilisateur par défaut dans le LocalSystem compte sur l'appareil Windows.
password Remplacez-le par un mot de passe sécurisé.

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

Tip

En fonction de votre configuration Windows, le mot de passe de l'utilisateur peut être configuré pour expirer à une date ultérieure. Pour vous assurer que vos applications Greengrass continuent de fonctionner, suivez la date d'expiration du mot de passe et mettez-le à jour avant son expiration. Vous pouvez également définir le mot de passe de l'utilisateur pour qu'il n'expire jamais.

- Pour vérifier la date d'expiration d'un utilisateur et de son mot de passe, exécutez la commande suivante.

```
net user ggc_user | findstr /C:expires
```

- Pour définir le mot de passe d'un utilisateur afin qu'il n'expire jamais, exécutez la commande suivante.

```
wmic UserAccount where "Name='ggc_user'" set PasswordExpires=False
```

- Si vous utilisez Windows 10 ou une version ultérieure où la [wmiccommande est obsolète](#), exécutez la commande suivante PowerShell .

```
Get-CimInstance -Query "SELECT * from Win32_UserAccount WHERE name = 'ggc_user'" | Set-CimInstance -Property @{PasswordExpires="False"}
```

5. Téléchargez et installez l'[PsExecutilitaire](#) de Microsoft sur l'appareil.
6. Utilisez l' PsExec utilitaire pour stocker le nom d'utilisateur et le mot de passe de l'utilisateur par défaut dans l'instance Credential Manager du LocalSystem compte. *password* Remplacez-le par le mot de passe utilisateur que vous avez défini précédemment.

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

S'il PsExec License Agreements'ouvre, choisissez Acceptd'accepter la licence et exécutez la commande.

 Note

Sur les appareils Windows, le LocalSystem compte exécute le noyau Greengrass, et vous devez utiliser l' PsExec utilitaire pour stocker les informations utilisateur par défaut dans le LocalSystem compte. L'application Credential Manager stocke ces informations dans le compte Windows de l'utilisateur actuellement connecté, plutôt que dans le LocalSystem compte.

Téléchargez le logiciel AWS IoT Greengrass Core

Vous pouvez télécharger la dernière version du logiciel AWS IoT Greengrass Core à l'adresse suivante :

- <https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest.zip>

 Note

Vous pouvez télécharger une version spécifique du logiciel AWS IoT Greengrass Core à l'emplacement suivant. Remplacez *version* par la version à télécharger.

```
https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-version.zip
```

Pour télécharger le logiciel AWS IoT Greengrass Core

1. Sur votre appareil principal, téléchargez le logiciel AWS IoT Greengrass Core dans un fichier nommé `greengrass-nucleus-latest.zip`.

Linux or Unix

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

Windows Command Prompt (CMD)

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

PowerShell

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

Si vous téléchargez ce logiciel, vous acceptez le [contrat de licence du logiciel Greengrass Core](#).

2. (Facultatif) Pour vérifier la signature du logiciel Greengrass Nucleus

Note

Cette fonctionnalité est disponible avec Greengrass nucleus version 2.9.5 et versions ultérieures.

- a. Utilisez la commande suivante pour vérifier la signature de votre artefact Greengrass nucleus :

Linux or Unix

```
jarsigner -verify -certs -verbose greengrass-nucleus-latest.zip
```

Windows Command Prompt (CMD)

Le nom du fichier peut être différent selon la version du JDK que vous installez.
jdk17.0.6_10 Remplacez-le par la version du JDK que vous avez installée.

```
"C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe" -verify -certs -verbose greengrass-nucleus-latest.zip
```

PowerShell

Le nom du fichier peut être différent selon la version du JDK que vous installez.
jdk17.0.6_10 Remplacez-le par la version du JDK que vous avez installée.

```
'C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe' -  
verify -certs -verbose greengrass-nucleus-latest.zip
```

- b. L'`jarsigner` invocation produit une sortie qui indique les résultats de la vérification.
 - i. Si le fichier zip Greengrass nucleus est signé, le résultat contient l'instruction suivante :

```
jar verified.
```

- ii. Si le fichier zip Greengrass nucleus n'est pas signé, le résultat contient l'instruction suivante :

```
jar is unsigned.
```

- c. Si vous avez fourni l'`-certs` option Jarsigner en même temps que les `-verbose` options `-verify` et, le résultat inclut également des informations détaillées sur le certificat du signataire.
- 3. Décompressez le logiciel AWS IoT Greengrass Core dans un dossier de votre appareil.
GreengrassInstaller Remplacez-le par le dossier que vous souhaitez utiliser.

Linux or Unix

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

Windows Command Prompt (CMD)

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

PowerShell

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



```
rm greengrass-nucleus-latest.zip
```

4. (Facultatif) Exécutez la commande suivante pour voir la version du logiciel AWS IoT Greengrass Core.

```
java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
```

Important

Si vous installez une version du noyau Greengrass antérieure à la version 2.4.0, ne supprimez pas ce dossier après avoir installé le logiciel Core. AWS IoT Greengrass Le logiciel AWS IoT Greengrass Core utilise les fichiers de ce dossier pour s'exécuter.

Si vous avez téléchargé la dernière version du logiciel, vous devez installer la version 2.4.0 ou ultérieure, et vous pouvez supprimer ce dossier après avoir installé le logiciel AWS IoT Greengrass Core.

Installation du logiciel AWS IoT Greengrass Core

Exécutez le programme d'installation avec des arguments qui spécifient les actions suivantes :

- Effectuez l'installation à partir d'un fichier de configuration partiel qui indique d'utiliser votre plug-in de provisionnement personnalisé pour provisionner AWS des ressources. Le logiciel AWS IoT Greengrass Core utilise un fichier de configuration qui spécifie la configuration de chaque composant Greengrass de l'appareil. Le programme d'installation crée un fichier de configuration complet à partir du fichier de configuration partiel que vous fournissez et des AWS ressources créées par le plug-in de provisionnement personnalisé.
- Spécifiez d'utiliser l'utilisateur `ggc_user` du système pour exécuter les composants logiciels sur le périphérique principal. Sur les appareils Linux, cette commande indique également d'utiliser le groupe `ggc_group` système, et le programme d'installation crée l'utilisateur et le groupe système pour vous.
- Configurez le logiciel AWS IoT Greengrass Core en tant que service système qui s'exécute au démarrage. Sur les appareils Linux, cela nécessite le [système d'initialisation Systemd](#).

 Important

Sur les appareils Windows Core, vous devez configurer le logiciel AWS IoT Greengrass Core en tant que service système.

Pour plus d'informations sur les arguments que vous pouvez spécifier, consultez [Arguments d'installation](#).

 Note

Si vous utilisez AWS IoT Greengrass un appareil dont la mémoire est limitée, vous pouvez contrôler la quantité de mémoire utilisée par le logiciel AWS IoT Greengrass Core. Pour contrôler l'allocation de mémoire, vous pouvez définir les options de taille de segment de mémoire JVM dans le paramètre de `jvmOptions` configuration de votre composant Nucleus. Pour de plus amples informations, veuillez consulter [Contrôlez l'allocation de mémoire grâce aux options JVM](#).

Pour installer le logiciel AWS IoT Greengrass Core (Linux)

1. Vérifiez la version du logiciel AWS IoT Greengrass Core.
 - Remplacez *GreengrassInstaller* par le chemin d'accès au dossier contenant le logiciel.

```
java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
```

2. Utilisez un éditeur de texte pour créer un fichier de configuration nommé `config.yaml` à fournir au programme d'installation.

Par exemple, sur un système basé sur Linux, vous pouvez exécuter la commande suivante pour utiliser GNU nano pour créer le fichier.

```
nano GreengrassInstaller/config.yaml
```

Copiez le contenu YAML suivant dans le fichier.

```
---
system:
  rootpath: "/greengrass/v2"
  # The following values are optional. Return them from the provisioning plugin or
  # set them here.
  # certificateFilePath: ""
  # privateKeyPath: ""
  # rootCaPath: ""
  # thingName: ""
services:
  aws.greengrass.Nucleus:
    version: "2.16.1"
    configuration:
      # The following values are optional. Return them from the provisioning plugin
      # or set them here.
      # awsRegion: ""
      # iotRoleAlias: ""
      # iotDataEndpoint: ""
      # iotCredEndpoint: ""
  com.example.CustomProvisioning:
    configuration:
      # You can specify configuration parameters to provide to your plugin.
      # pluginParameter: ""
```

Ensuite, procédez comme suit :

- Remplacez *2.16.1* par la version du logiciel AWS IoT Greengrass Core.
- Remplacez chaque instance de */greengrass/v2* par le dossier racine de Greengrass.
- (Facultatif) Spécifiez les valeurs de configuration du système et du noyau. Vous devez définir ces valeurs si votre plugin d'approvisionnement ne les fournit pas.
- (Facultatif) Spécifiez les paramètres de configuration à fournir à votre plug-in de provisionnement.

Note

Dans ce fichier de configuration, vous pouvez personnaliser d'autres options de configuration, telles que les ports et le proxy réseau à utiliser, comme indiqué dans l'exemple suivant. Pour plus d'informations, consultez la section Configuration du [noyau de Greengrass](#).

```
---
system:
  rootpath: "/greengrass/v2"
  # The following values are optional. Return them from the provisioning
  plugin or set them here.
  # certificateFilePath: ""
  # privateKeyPath: ""
  # rootCaPath: ""
  # thingName: ""
services:
  aws.greengrass.Nucleus:
    version: "2.16.1"
    configuration:
      mqtt:
        port: 443
      greengrassDataPlanePort: 443
      networkProxy:
        noProxyAddresses: "http://192.168.0.1,www.example.com"
        proxy:
          url: "http://my-proxy-server:1100"
          username: "Mary_Major"
          password: "pass@word1357"
      # The following values are optional. Return them from the provisioning
      plugin or set them here.
      # awsRegion: ""
      # iotRoleAlias: ""
      # iotDataEndpoint: ""
      # iotCredEndpoint: ""
  com.example.CustomProvisioning:
    configuration:
      # You can specify configuration parameters to provide to your plugin.
      # pluginParameter: ""
```

3. Exécutez le programme d'installation. Spécifiez `--trusted-plugin` pour fournir votre plugin de provisionnement personnalisé et spécifiez `--init-config` pour fournir le fichier de configuration.

Note

Pour Windows, la limitation de la longueur du chemin est de 260 caractères. Si vous utilisez Windows, utilisez un dossier racine tel que `C:\greengrass\v2` ou `D:`

\greengrass\v2 pour conserver les chemins des composants Greengrass en dessous de la limite de 260 caractères.

- Remplacez */greengrass/v2* ou *C:\greengrass\v2* par le dossier racine de Greengrass.
- Remplacez chaque instance de *GreengrassInstaller* par le dossier dans lequel vous avez décompressé le programme d'installation.
- Remplacez le chemin du fichier JAR du plugin de provisionnement personnalisé par le chemin du fichier JAR de votre plugin.

Linux or Unix

```
sudo -E java -Droot="/greengrass/v2" -Dlog.store=FILE \  
-jar ./GreengrassInstaller/lib/Greengrass.jar \  
--trusted-plugin /path/to/com.example.CustomProvisioning.jar \  
--init-config ./GreengrassInstaller/config.yaml \  
--component-default-user ggc_user:ggc_group \  
--setup-system-service true
```

Windows Command Prompt (CMD)

```
java -Droot="C:\greengrass\v2" "-Dlog.store=FILE" ^  
-jar ./GreengrassInstaller/lib/Greengrass.jar ^  
--trusted-plugin /path/to/com.example.CustomProvisioning.jar ^  
--init-config ./GreengrassInstaller/config.yaml ^  
--component-default-user ggc_user ^  
--setup-system-service true
```

PowerShell

```
java -Droot="C:\greengrass\v2" "-Dlog.store=FILE" `\  
-jar ./GreengrassInstaller/lib/Greengrass.jar `\  
--trusted-plugin /path/to/com.example.CustomProvisioning.jar `\  
--init-config ./GreengrassInstaller/config.yaml `\  
--component-default-user ggc_user `\  
--setup-system-service true
```

 Important

Sur les appareils Windows Core, vous devez spécifier de configurer le logiciel AWS IoT Greengrass Core en tant que service système.

Si vous le spécifiez `--setup-system-service true`, le programme d'installation affiche `Successfully set up Nucleus as a system service` s'il a configuré et exécuté le logiciel en tant que service système. Dans le cas contraire, le programme d'installation n'affiche aucun message s'il installe le logiciel avec succès.

 Note

Vous ne pouvez pas utiliser l'`deploy-dev-tools` argument pour déployer des outils de développement locaux lorsque vous exécutez le programme d'installation sans l'`--provision true` argument. Pour plus d'informations sur le déploiement de la CLI Greengrass directement sur votre appareil, consultez [Interface de ligne de commande Greengrass](#)

4. Vérifiez l'installation en consultant les fichiers du dossier racine.

Linux or Unix

```
ls /greengrass/v2
```

Windows Command Prompt (CMD)

```
dir C:\greengrass\v2
```

PowerShell

```
ls C:\greengrass\v2
```

Si l'installation a réussi, le dossier racine contient plusieurs dossiers, tels que `configpackages`, `etlogs`.

Si vous avez installé le logiciel AWS IoT Greengrass Core en tant que service système, le programme d'installation exécute le logiciel pour vous. Dans le cas contraire, vous devez exécuter le logiciel manuellement. Pour de plus amples informations, veuillez consulter [Exécutez le logiciel AWS IoT Greengrass Core](#).

Pour plus d'informations sur la configuration et l'utilisation du logiciel AWS IoT Greengrass, consultez les rubriques suivantes :

- [Configuration du logiciel AWS IoT Greengrass de base](#)
- [Développer des AWS IoT Greengrass composants](#)
- [Déployer AWS IoT Greengrass des composants sur des appareils](#)
- [Interface de ligne de commande Greengrass](#)

Développez des plugins de provisionnement personnalisés

Pour développer un plugin de provisionnement personnalisé, créez une classe Java qui implémente l'`com.aws.greengrass.provisioning.DeviceIdentityInterface`. Vous pouvez inclure le fichier JAR Greengrass nucleus dans votre projet pour accéder à cette interface et à ses classes. Cette interface définit une méthode qui saisit une configuration de plugin et produit une configuration de provisionnement. La configuration de provisionnement définit les configurations du système et du composant du noyau [Greengrass](#). Le programme d'installation du logiciel AWS IoT Greengrass Core utilise cette configuration de provisionnement pour configurer le logiciel AWS IoT Greengrass Core sur un appareil.

Après avoir développé un plug-in de provisionnement personnalisé, créez-le sous forme de fichier JAR que vous pouvez fournir au programme d'installation du logiciel AWS IoT Greengrass Core pour qu'il exécute votre plug-in lors de l'installation. Le programme d'installation exécute votre plug-in de provisionnement personnalisé dans la même JVM qu'il utilise. Vous pouvez donc créer un fichier JAR contenant uniquement le code de votre plug-in.

Note

Le [plugin de provisionnement de AWS IoT flotte](#) implémente le provisionnement de flotte `DeviceIdentityInterface` à utiliser lors de l'installation. Le plugin de provisionnement de flotte est open source, vous pouvez donc explorer son code source pour voir un exemple d'utilisation de l'interface du plug-in de provisionnement. Pour plus d'informations, consultez le [plugin de provisionnement de AWS IoT flotte](#) sur GitHub.

Rubriques

- [Prérequis](#)
- [Implémenter l' DeviceIdentityInterface interface](#)

Prérequis

Pour développer un plugin de provisionnement personnalisé, vous devez créer une classe Java répondant aux exigences suivantes :

- Utilise le `com.aws.greengrass` package ou un package inclus dans le `com.aws.greengrass` package.
- Possède un constructeur sans aucun argument.
- Implémente l'`DeviceIdentityInterface` interface. Pour de plus amples informations, veuillez consulter [Implémenter l' DeviceIdentityInterface interface](#).

Implémenter l' DeviceIdentityInterface interface

Pour utiliser l'`com.aws.greengrass.provisioning.DeviceIdentityInterface` de votre plugin personnalisé, ajoutez le noyau Greengrass en tant que dépendance à votre projet.

Pour utiliser le `DeviceIdentityInterface` dans un projet de plugin de provisionnement personnalisé

- Vous pouvez ajouter le fichier JAR du noyau Greengrass en tant que bibliothèque ou ajouter le noyau Greengrass en tant que dépendance Maven. Effectuez l'une des actions suivantes :
 - Pour ajouter le fichier Greengrass nucleus JAR en tant que bibliothèque, téléchargez le logiciel AWS IoT Greengrass Core, qui contient le fichier Greengrass nucleus JAR. Vous pouvez télécharger la dernière version du logiciel AWS IoT Greengrass Core à l'adresse suivante :
 - <https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest.zip>

Vous pouvez trouver le fichier JAR Greengrass nucleus (`Greengrass.jar`) dans le `lib` dossier du fichier ZIP. Ajoutez ce fichier JAR à votre projet.

- Pour utiliser le noyau Greengrass dans un projet Maven, ajoutez une dépendance à l'`nucleusartifact` du groupe `com.aws.greengrass` Vous devez également ajouter

le greengrass-common dépôt, car le noyau Greengrass n'est pas disponible dans le référentiel central de Maven.

```
<project ...>
  ...
  <repositories>
    <repository>
      <id>greengrass-common</id>
      <name>greengrass common</name>
      <url>https://d2jrmugq4sldf.cloudfront.net/snapshots</url>
    </repository>
  </repositories>
  ...
  <dependencies>
    <dependency>
      <groupId>com.aws.greengrass</groupId>
      <artifactId>nucleus</artifactId>
      <version>2.5.0-SNAPSHOT</version>
      <scope>provided</scope>
    </dependency>
  </dependencies>
</project>
```

L' DeviceIdentityInterfaceinterface

L'`com.aws.greengrass.provisioning.DeviceIdentityInterfaceinterface` a la forme suivante.

Note

[Vous pouvez également explorer ces classes dans le package com.aws.greengrass.provisioning du code source du noyau de Greengrass sur GitHub](#)

```
public interface com.aws.greengrass.provisioning.DeviceIdentityInterface {
    ProvisionConfiguration updateIdentityConfiguration(ProvisionContext context)
        throws RetryableProvisioningException, InterruptedException;

    // Return the name of the plugin.
    String name();
}
```

```
com.aws.greengrass.provisioning.ProvisionConfiguration {
    SystemConfiguration systemConfiguration;
    NucleusConfiguration nucleusConfiguration
}

com.aws.greengrass.provisioning.ProvisionConfiguration.SystemConfiguration {
    String certificateFilePath;
    String privateKeyPath;
    String rootCAPath;
    String thingName;
}

com.aws.greengrass.provisioning.ProvisionConfiguration.NucleusConfiguration {
    String awsRegion;
    String iotCredentialsEndpoint;
    String iotDataEndpoint;
    String iotRoleAlias;
}

com.aws.greengrass.provisioning.ProvisioningContext {
    Map<String, Object> parameterMap;
    String provisioningPolicy; // The policy is always "PROVISION_IF_NOT_PROVISIONED".
}

com.aws.greengrass.provisioning.exceptions.RetryableProvisioningException {}
```

Chaque valeur de configuration dans le `SystemConfiguration` et `NucleusConfiguration` est requise pour installer le logiciel AWS IoT Greengrass Core, mais vous pouvez revenir à `null`. Si votre plug-in de provisionnement personnalisé renvoie une valeur `null` de configuration, vous devez fournir cette valeur dans la configuration du système ou du noyau lorsque vous créez le `config.yaml` fichier à fournir au programme d'installation du logiciel AWS IoT Greengrass Core. Si votre plugin de provisionnement personnalisé renvoie une valeur non nulle pour une option que vous définissez également dans `config.yaml`, le programme d'installation remplace la valeur in `config.yaml` par la valeur renvoyée par le plugin.

Arguments d'installation

Le logiciel AWS IoT Greengrass principal inclut un programme d'installation qui configure le logiciel et fournit les AWS ressources nécessaires au fonctionnement du périphérique principal Greengrass.

Le programme d'installation inclut les arguments suivants que vous pouvez spécifier pour configurer l'installation :

`-h, --help`

(Facultatif) Affichez les informations d'aide du programme d'installation.

`--version`

(Facultatif) Afficher la version du logiciel AWS IoT Greengrass principal.

`-Droot`

(Facultatif) Le chemin d'accès au dossier à utiliser comme racine pour le logiciel AWS IoT Greengrass Core.

 Note

Cet argument définit une propriété JVM, vous devez donc la spécifier avant `-jar` d'exécuter le programme d'installation. Par exemple, spécifiez `java -Droot="/greengrass/v2" -jar /path/to/Greengrass.jar`.

Par défaut :

- Linux : `~/greengrass`
- Windows: `%USERPROFILE%/greengrass`

`-ar, --aws-region`

Le Région AWS logiciel AWS IoT Greengrass Core utilise pour récupérer ou créer les AWS ressources requises.

`-p, --provision`

(Facultatif) Vous pouvez enregistrer cet appareil en tant qu' AWS IoT objet et fournir les AWS ressources dont le périphérique principal a besoin. Si vous le spécifiez `true`, le logiciel AWS IoT Greengrass Core fournit n' AWS IoT importe quel objet, (facultatif) un AWS IoT groupe d'objets, un rôle IAM et un alias de AWS IoT rôle.

Par défaut : `false`

`-tn, --thing-name`

(Facultatif) Le nom de l' AWS IoT objet que vous enregistrez en tant que périphérique principal. Si l'objet portant le nom n'existe pas dans le vôtre Compte AWS, c'est le logiciel AWS IoT Greengrass Core qui le crée.

Note

Le nom de l'objet ne peut pas contenir de caractères deux-points (:).

Vous devez spécifier `--provision true` pour appliquer cet argument.

Par défaut : `GreengrassV2IotThing_` plus un UUID aléatoire.

`-tgn, --thing-group-name`

(Facultatif) Le nom du AWS IoT groupe d'objets dans lequel vous ajoutez l' AWS IoT objet de cet appareil principal. Si un déploiement cible ce groupe d'objets, ce périphérique principal reçoit ce déploiement lorsqu'il se connecte à AWS IoT Greengrass. Si le groupe d'objets portant ce nom n'existe pas dans votre Compte AWS répertoire, le logiciel AWS IoT Greengrass Core le crée.

Note

Le nom du groupe d'objets ne peut pas contenir de deux-points (:).

Vous devez spécifier `--provision true` pour appliquer cet argument.

`-tpn, --thing-policy-name`

Cette fonctionnalité est disponible pour les versions 2.4.0 et ultérieures du composant [Greengrass](#) nucleus.

(Facultatif) Nom de la AWS IoT politique à associer au certificat d' AWS IoT objet de ce périphérique principal. Si la AWS IoT politique portant ce nom n'existe pas dans votre répertoire Compte AWS, le logiciel AWS IoT Greengrass Core la crée.

Le logiciel AWS IoT Greengrass Core crée une AWS IoT politique permissive par défaut. Vous pouvez limiter cette politique ou créer une politique personnalisée dans laquelle vous limitez les

autorisations pour votre cas d'utilisation. Pour de plus amples informations, veuillez consulter [AWS IoT Politique minimale pour les appareils AWS IoT Greengrass V2 principaux](#).

Vous devez spécifier `--provision true` pour appliquer cet argument.

Par défaut : `GreengrassV2IoTThingPolicy`

`-trn, --tes-role-name`

(Facultatif) Nom du rôle IAM à utiliser pour acquérir des AWS informations d'identification permettant à l'appareil principal d'interagir avec les AWS services. Si le rôle portant ce nom n'existe pas dans votre répertoire Compte AWS, le logiciel AWS IoT Greengrass Core le crée avec la `GreengrassV2TokenExchangeRoleAccess` politique. Ce rôle n'a pas accès à vos compartiments S3 dans lesquels vous hébergez des artefacts de composants. Vous devez donc ajouter des autorisations aux compartiments et objets S3 de vos artefacts lorsque vous créez un composant. Pour de plus amples informations, veuillez consulter [Autoriser les appareils principaux à interagir avec les AWS services](#).

Vous devez spécifier `--provision true` pour appliquer cet argument.

Par défaut : `GreengrassV2TokenExchangeRole`

`-tra, --tes-role-alias-name`

(Facultatif) Le nom de l'alias de AWS IoT rôle qui pointe vers le rôle IAM qui fournit les AWS informations d'identification pour ce périphérique principal. Si l'alias de rôle portant ce nom n'existe pas dans votre répertoire Compte AWS, le logiciel AWS IoT Greengrass Core le crée et le pointe vers le rôle IAM que vous spécifiez.

Vous devez spécifier `--provision true` pour appliquer cet argument.

Par défaut : `GreengrassV2TokenExchangeRoleAlias`

`-ss, --setup-system-service`

(Facultatif) Vous pouvez configurer le logiciel AWS IoT Greengrass Core en tant que service système qui s'exécute au démarrage de cet appareil. Le nom du service système est `greengrass`. Pour de plus amples informations, veuillez consulter [Configurer le noyau Greengrass en tant que service système](#).

Sur les systèmes d'exploitation Linux, cet argument nécessite que le système d'initialisation `systemd` soit disponible sur le périphérique.

 Important

Sur les appareils Windows Core, vous devez configurer le logiciel AWS IoT Greengrass Core en tant que service système.

Par défaut : `false`

`-u, --component-default-user`

Le nom ou l'ID de l'utilisateur utilisé par le logiciel AWS IoT Greengrass Core pour exécuter les composants. Par exemple, vous pouvez définir **ggc_user**. Cette valeur est obligatoire lorsque vous exécutez le programme d'installation sur les systèmes d'exploitation Windows.

Sur les systèmes d'exploitation Linux, vous pouvez également éventuellement spécifier le groupe. Spécifiez l'utilisateur et le groupe séparés par deux points. Par exemple, **ggc_user:ggc_group**.

Les considérations supplémentaires suivantes s'appliquent aux systèmes d'exploitation Linux :

- Si vous exécutez le composant en tant que root, l'utilisateur du composant par défaut est celui défini dans le fichier de configuration. Si le fichier de configuration ne définit aucun utilisateur, la valeur par défaut est. `ggc_user:ggc_group` S'ils `ggc_group` existent `ggc_user` ou non, le logiciel les crée.
- Si vous l'exécutez en tant qu'utilisateur non root, le logiciel AWS IoT Greengrass Core utilise cet utilisateur pour exécuter les composants.
- Si vous ne spécifiez aucun groupe, le logiciel AWS IoT Greengrass Core utilise le groupe principal de l'utilisateur du système.

Pour de plus amples informations, veuillez consulter [Configurer l'utilisateur qui exécute les composants](#).

`-d, --deploy-dev-tools`

(Facultatif) Vous pouvez télécharger et déployer le composant [Greengrass CLI](#) sur ce périphérique principal. Vous pouvez utiliser cet outil pour développer et déboguer des composants sur ce périphérique principal.

 Important

Nous vous recommandons d'utiliser ce composant uniquement dans les environnements de développement, et non dans les environnements de production. Ce composant permet

d'accéder à des informations et à des opérations dont vous n'avez généralement pas besoin dans un environnement de production. Respectez le principe du moindre privilège en déployant ce composant uniquement sur les appareils principaux là où vous en avez besoin.

Vous devez spécifier `--provision true` pour appliquer cet argument.

Par défaut : `false`

`-init, --init-config`

(Facultatif) Le chemin d'accès au fichier de configuration à utiliser pour installer le logiciel AWS IoT Greengrass Core. Vous pouvez utiliser cette option pour configurer de nouveaux périphériques principaux avec une configuration de noyau spécifique, par exemple.

 Important

Le fichier de configuration que vous spécifiez fusionne avec le fichier de configuration existant sur le périphérique principal. Cela inclut les composants et les configurations des composants du périphérique principal. Nous recommandons que le fichier de configuration répertorie uniquement les configurations que vous essayez de modifier.

`-tp, --trusted-plugin`

(Facultatif) Le chemin d'accès à un fichier JAR à charger en tant que plugin sécurisé. Utilisez cette option pour fournir des fichiers JAR du plug-in de provisionnement, par exemple pour l'installation avec le [provisionnement du parc](#) ou le [provisionnement personnalisé](#), ou pour l'installation avec la clé privée et le certificat dans un module de sécurité [matériel](#).

`-s, --start`

(Facultatif) Vous pouvez démarrer le logiciel AWS IoT Greengrass Core après son installation et, éventuellement, provisionner les ressources.

Par défaut : `true`

Exécutez le logiciel AWS IoT Greengrass Core

Après avoir [installé le logiciel AWS IoT Greengrass Core](#), exécutez-le pour connecter votre appareil à AWS IoT Greengrass.

Lorsque vous installez le logiciel AWS IoT Greengrass Core, vous pouvez spécifier s'il convient de l'installer en tant que service système avec [systemd](#). Si vous choisissez cette option, le programme d'installation exécute le logiciel pour vous et le configure pour qu'il s'exécute au démarrage de votre appareil.

Important

Sur les appareils Windows Core, vous devez configurer le logiciel AWS IoT Greengrass Core en tant que service système.

Rubriques

- [Vérifiez si le logiciel AWS IoT Greengrass Core fonctionne en tant que service système](#)
- [Exécutez le logiciel AWS IoT Greengrass Core en tant que service système](#)
- [Exécutez le logiciel AWS IoT Greengrass Core sans service système](#)

Vérifiez si le logiciel AWS IoT Greengrass Core fonctionne en tant que service système

Lorsque vous installez le logiciel AWS IoT Greengrass Core, vous pouvez spécifier l'argument `--setup-system-service true` pour installer le logiciel AWS IoT Greengrass Core en tant que service système. Les appareils Linux nécessitent le [système d'initialisation systemd](#) pour configurer le logiciel AWS IoT Greengrass Core en tant que service système. Si vous utilisez cette option, le programme d'installation exécute le logiciel pour vous et le configure pour qu'il s'exécute au démarrage de votre appareil. Le programme d'installation affiche le message suivant s'il installe correctement le logiciel AWS IoT Greengrass Core en tant que service système.

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

Si vous avez déjà installé le logiciel AWS IoT Greengrass Core et que vous ne disposez pas de la sortie du programme d'installation, vous pouvez vérifier si le logiciel est installé en tant que service système.

Pour vérifier si le logiciel AWS IoT Greengrass Core est installé en tant que service système

- Exécutez la commande suivante pour vérifier l'état du service système Greengrass.

Linux or Unix (systemd)

```
sudo systemctl status greengrass.service
```

La réponse ressemble à l'exemple suivant si le logiciel AWS IoT Greengrass Core est installé en tant que service système et actif.

```
# greengrass.service - Greengrass Core
  Loaded: loaded (/etc/systemd/system/greengrass.service; enabled; vendor
  preset: disabled)
  Active: active (running) since Thu 2021-02-11 01:33:44 UTC; 4 days ago
  Main PID: 16107 (sh)
  CGroup: /system.slice/greengrass.service
          ##16107 /bin/sh /greengrass/v2/alts/current/distro/bin/loader
          ##16111 java -Dlog.store=FILE -Droot=/greengrass/v2 -jar /greengrass/
  v2/alts/current/distro/lib/Greengrass...
```

S'il `systemctl greengrass.service` n'est pas trouvé, le logiciel AWS IoT Greengrass Core n'est pas installé en tant que service système. Pour exécuter le logiciel, voir [Exécutez le logiciel AWS IoT Greengrass Core sans service système](#).

Windows Command Prompt (CMD)

```
sc query greengrass
```

La réponse ressemble à l'exemple suivant si le logiciel AWS IoT Greengrass Core est installé en tant que service Windows et actif.

```
SERVICE_NAME: greengrass
        TYPE               : 10  WIN32_OWN_PROCESS
        STATE                : 4   RUNNING
                                (STOPPABLE, NOT_PAUSABLE, ACCEPTS_SHUTDOWN)
        WIN32_EXIT_CODE       : 0   (0x0)
        SERVICE_EXIT_CODE    : 0   (0x0)
        CHECKPOINT            : 0x0
        WAIT_HINT             : 0x0
```

PowerShell

```
Get-Service greengrass
```

La réponse ressemble à l'exemple suivant si le logiciel AWS IoT Greengrass Core est installé en tant que service Windows et actif.

Status	Name	DisplayName
Running	greengrass	greengrass

Exécutez le logiciel AWS IoT Greengrass Core en tant que service système

Si le logiciel AWS IoT Greengrass Core est installé en tant que service système, vous pouvez utiliser le gestionnaire de services système pour démarrer, arrêter et gérer le logiciel. Pour de plus amples informations, veuillez consulter [Configurer le noyau Greengrass en tant que service système](#).

Pour exécuter le logiciel AWS IoT Greengrass Core

- Exécutez la commande suivante pour démarrer le logiciel AWS IoT Greengrass Core.

Linux or Unix (systemd)

```
sudo systemctl start greengrass.service
```

Windows Command Prompt (CMD)

```
sc start greengrass
```

PowerShell

```
Start-Service greengrass
```

Exécutez le logiciel AWS IoT Greengrass Core sans service système

Sur les appareils Linux principaux, si le logiciel AWS IoT Greengrass Core n'est pas installé en tant que service système, vous pouvez exécuter le script de chargement du logiciel pour exécuter le logiciel.

Pour exécuter le logiciel AWS IoT Greengrass Core sans service système

- Exécutez la commande suivante pour démarrer le logiciel AWS IoT Greengrass Core. Si vous exécutez cette commande dans un terminal, vous devez laisser la session du terminal ouverte pour que le logiciel AWS IoT Greengrass Core continue de fonctionner.
- Remplacez `/greengrass/v2` ou `C:\greengrass\v2` par le dossier racine Greengrass que vous utilisez.

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

Le logiciel imprime le message suivant s'il démarre correctement.

```
Launched Nucleus successfully.
```

Exécuter le logiciel AWS IoT Greengrass Core dans un conteneur Docker

AWS IoT Greengrass peut être configuré pour fonctionner dans un conteneur Docker. Docker est une plate-forme qui fournit les outils nécessaires pour créer, exécuter, tester et déployer des applications basées sur des conteneurs Linux. Lorsque vous exécutez une image AWS IoT Greengrass Docker, vous pouvez choisir de fournir vos AWS informations d'identification au conteneur Docker et d'autoriser le programme d'installation du logiciel AWS IoT Greengrass Core à provisionner automatiquement les AWS ressources dont un périphérique principal Greengrass a besoin pour fonctionner. Si vous ne souhaitez pas fournir AWS d'informations d'identification, vous pouvez provisionner manuellement les AWS ressources et exécuter le logiciel AWS IoT Greengrass Core dans le conteneur Docker.

Rubriques

- [Exigences et plateformes prises en charge](#)

- [AWS IoT Greengrass Téléchargements du logiciel Docker](#)
- [Choisissez le mode de provisionnement AWS des ressources](#)
- [Créez l'image du AWS IoT Greengrass conteneur à partir d'un Dockerfile](#)
- [Exécuter AWS IoT Greengrass dans un conteneur Docker avec provisionnement automatique des ressources](#)
- [Exécuter AWS IoT Greengrass dans un conteneur Docker avec provisionnement manuel des ressources](#)
- [Résolution des problèmes AWS IoT Greengrass dans un conteneur Docker](#)

Exigences et plateformes prises en charge

Les ordinateurs hôtes doivent répondre aux exigences minimales suivantes pour installer et exécuter le logiciel AWS IoT Greengrass Core dans un conteneur Docker :

- Système d'exploitation basé sur Linux doté d'une connexion Internet.
- [Docker Engine](#) version 18.09 ou ultérieure.
- (Facultatif) [Docker Compose](#) version 1.22 ou ultérieure. Docker Compose n'est requis que si vous souhaitez utiliser la CLI Docker Compose pour exécuter vos images Docker.

Pour exécuter les composants de la fonction Lambda dans le conteneur Docker, vous devez configurer le conteneur pour répondre à des exigences supplémentaires. Pour de plus amples informations, veuillez consulter [Exigences relatives à la fonction Lambda](#).

Exécuter les composants en mode processus

AWS IoT Greengrass ne prend pas en charge l'exécution de fonctions Lambda ou de composants AWS fournis par Lambda dans un environnement d'exécution isolé à l'intérieur du AWS IoT Greengrass conteneur Docker. Vous devez exécuter ces composants en mode processus sans aucune isolation.

Lorsque vous configurez un composant de fonction Lambda, définissez le mode d'isolation sur Aucun conteneur. Pour de plus amples informations, veuillez consulter [Exécuter AWS Lambda des fonctions](#).

Lorsque vous déployez l'un des composants AWS fournis ci-dessous, mettez à jour la configuration de chaque composant pour définir le `containerMode` paramètre sur `NoContainer` Pour plus

d'informations sur les mises à jour de configuration, consultez [Mettre à jour les configurations des composants](#).

- [CloudWatch métriques](#)
- [Défenseur de l'appareil](#)
- [Firehose](#)
- [Adaptateur de protocole Modbus-RTU](#)
- [Amazon SNS](#)

AWS IoT Greengrass Téléchargements du logiciel Docker

AWS IoT Greengrass fournit un Dockerfile pour créer une image de conteneur sur laquelle le logiciel AWS IoT Greengrass Core et ses dépendances sont installés sur une image de base Amazon Linux 2 (x86_64). Vous pouvez modifier l'image de base dans le Dockerfile pour l'exécuter AWS IoT Greengrass sur une architecture de plate-forme différente.

Téléchargez le package Dockerfile depuis. [GitHub](#)

Le Dockerfile utilise une ancienne version de Greengrass. Vous devez mettre à jour le fichier pour utiliser la version de Greengrass que vous souhaitez. Pour plus d'informations sur la création de l'image du AWS IoT Greengrass conteneur à partir du Dockerfile, consultez. [Créez l'image du AWS IoT Greengrass conteneur à partir d'un Dockerfile](#)

Choisissez le mode de provisionnement AWS des ressources

Lorsque vous installez le logiciel AWS IoT Greengrass Core dans un conteneur Docker, vous pouvez choisir de provisionner automatiquement les AWS ressources dont un appareil Greengrass a besoin pour fonctionner ou d'utiliser des ressources que vous provisionnez manuellement.

- Approvisionnement automatique des ressources : le programme d'installation AWS IoT provisionne l' AWS IoT objet, le groupe d'objets, le rôle IAM et l'alias de AWS IoT rôle lorsque vous exécutez l'image du AWS IoT Greengrass conteneur pour la première fois. Le programme d'installation peut également déployer les outils de développement locaux sur le périphérique principal, afin que vous puissiez utiliser l'appareil pour développer et tester des composants logiciels personnalisés. Pour provisionner automatiquement ces ressources, vous devez fournir des AWS informations d'identification en tant que variables d'environnement à l'image Docker.

Pour utiliser le provisionnement automatique, vous devez définir la variable d'environnement Docker `PROVISION=true` et monter un fichier d'informations d'identification pour fournir vos AWS informations d'identification au conteneur.

- Approvisionnement manuel des ressources : si vous ne souhaitez pas fournir AWS d'informations d'identification au conteneur, vous pouvez configurer manuellement les AWS ressources avant d'exécuter l'image du AWS IoT Greengrass conteneur. Vous devez créer un fichier de configuration pour fournir des informations sur ces ressources au programme d'installation du logiciel AWS IoT Greengrass Core dans le conteneur Docker.

Pour utiliser le provisionnement manuel, vous devez définir la variable d'environnement Docker. `PROVISION=false` Le provisionnement manuel est l'option par défaut.

Pour de plus amples informations, veuillez consulter [Créez l'image du AWS IoT Greengrass conteneur à partir d'un Dockerfile](#).

Créez l'image du AWS IoT Greengrass conteneur à partir d'un Dockerfile

AWS fournit un Dockerfile que vous pouvez télécharger et utiliser pour exécuter le logiciel AWS IoT Greengrass Core dans un conteneur Docker. Les Dockerfiles contiennent le code source permettant de créer des images de AWS IoT Greengrass conteneurs.

Avant de créer une image de AWS IoT Greengrass conteneur, vous devez configurer votre Dockerfile pour sélectionner la version du logiciel AWS IoT Greengrass Core que vous souhaitez installer. Vous pouvez également configurer des variables d'environnement pour choisir le mode de provisionnement des ressources lors de l'installation et personnaliser d'autres options d'installation. Cette section décrit comment configurer et créer une image AWS IoT Greengrass Docker à partir d'un Dockerfile.

Téléchargez le package Dockerfile

Vous pouvez télécharger le package AWS IoT Greengrass Dockerfile à l'adresse suivante : GitHub

[Référentiel Docker AWS Greengrass](#)

Après avoir téléchargé le package, extrayez le contenu `download-directory/aws-greengrass-docker-nucleus-version` dans le dossier de votre ordinateur. Le Dockerfile utilise une ancienne version de Greengrass. Vous devez mettre à jour le fichier pour utiliser la version de Greengrass que vous souhaitez.

Spécifiez la version du logiciel AWS IoT Greengrass Core

Utilisez l'argument `build` suivant dans le Dockerfile pour spécifier la version du logiciel AWS IoT Greengrass Core que vous souhaitez utiliser dans l'image AWS IoT Greengrass Docker. Par défaut, le Dockerfile utilise la dernière version du logiciel AWS IoT Greengrass Core.

GREENGRASS_RELEASE_VERSION

Version du logiciel AWS IoT Greengrass Core. Par défaut, le Dockerfile télécharge la dernière version disponible du noyau Greengrass. Définissez la valeur sur la version du noyau que vous souhaitez télécharger.

Définir les variables d'environnement

Les variables d'environnement vous permettent de personnaliser la façon dont le logiciel AWS IoT Greengrass Core est installé dans le conteneur Docker. Vous pouvez définir des variables d'environnement pour votre image AWS IoT Greengrass Docker de différentes manières.

- Pour utiliser les mêmes variables d'environnement pour créer plusieurs images, définissez les variables d'environnement directement dans le Dockerfile.
- Si vous avez l'habitude de démarrer votre conteneur, transmettez des variables d'environnement en tant qu'arguments dans la commande ou définissez des variables d'environnement dans un fichier de variables d'environnement, puis transmettez le fichier en argument. Pour plus d'informations sur la définition des variables d'environnement dans Docker, consultez les [variables d'environnement](#) dans la documentation Docker.
- Si vous avez l'habitude de démarrer votre conteneur, définissez les variables d'environnement dans un fichier de variables d'environnement, puis transmettez le fichier en argument. Pour plus d'informations sur la définition des variables d'environnement dans Compose, consultez la [documentation Docker](#).

Vous pouvez configurer les variables d'environnement suivantes pour l'image AWS IoT Greengrass Docker.

Note

Ne modifiez pas la `TINI_KILL_PROCESS_GROUP` variable dans le Dockerfile. Cette variable permet le transfert SIGTERM PIDs à tous les membres du groupe PID afin que le logiciel

AWS IoT Greengrass Core puisse s'arrêter correctement lorsque le conteneur Docker est arrêté.

GGC_ROOT_PATH

(Facultatif) Le chemin d'accès au dossier du conteneur à utiliser comme racine pour le logiciel AWS IoT Greengrass Core.

Par défaut : `/greengrass/v2`

PROVISION

(Facultatif) Détermine si le AWS IoT Greengrass Core fournit AWS des ressources.

- Si vous le spécifiez `true`, le logiciel AWS IoT Greengrass Core enregistre l'image du conteneur en tant qu' AWS IoT objet et fournit les AWS ressources dont le périphérique principal Greengrass a besoin. Le logiciel de AWS IoT Greengrass base fournit n' AWS IoT importe quel objet, (facultatif) un AWS IoT groupe d'objets, un rôle IAM et un alias de AWS IoT rôle. Pour de plus amples informations, veuillez consulter [Exécuter AWS IoT Greengrass dans un conteneur Docker avec provisionnement automatique des ressources](#).
- Si vous le spécifiez `false`, vous devez créer un fichier de configuration à fournir au programme d'installation AWS IoT Greengrass Core qui indique d'utiliser les AWS ressources et les certificats que vous avez créés manuellement. Pour de plus amples informations, veuillez consulter [Exécuter AWS IoT Greengrass dans un conteneur Docker avec provisionnement manuel des ressources](#).

Par défaut: `false`

AWS_REGION

(Facultatif) Le Région AWS logiciel AWS IoT Greengrass Core utilise pour récupérer ou créer les AWS ressources requises.

Par défaut: `us-east-1`.

THING_NAME

(Facultatif) Le nom de l' AWS IoT objet que vous enregistrez en tant que périphérique principal. Si l'objet portant ce nom n'existe pas dans votre ordinateur Compte AWS, c'est le logiciel AWS IoT Greengrass Core qui le crée.

Vous devez spécifier `PROVISION=true` pour appliquer cet argument.

Par défaut : `GreengrassV2IotThing_` plus un UUID aléatoire.

THING_GROUP_NAME

(Facultatif) Le nom du groupe d'objets auquel vous ajoutez ce périphérique principal est. AWS IoT Si un déploiement cible ce groupe d'objets, celui-ci et les autres appareils principaux de ce groupe reçoivent ce déploiement lorsqu'ils se connectent à celui-ci AWS IoT Greengrass. AWS IoT Si le groupe d'objets portant ce nom n'existe pas dans votre Compte AWS répertoire, le logiciel AWS IoT Greengrass Core le crée.

Vous devez spécifier `PROVISION=true` pour appliquer cet argument.

TES_ROLE_NAME

(Facultatif) Nom du rôle IAM à utiliser pour obtenir des AWS informations d'identification permettant à l'appareil principal de Greengrass d'interagir avec les AWS services. Si le rôle portant ce nom n'existe pas dans votre répertoire Compte AWS, le logiciel AWS IoT Greengrass Core le crée avec la `GreengrassV2TokenExchangeRoleAccess` politique. Ce rôle n'a pas accès à vos compartiments S3 dans lesquels vous hébergez des artefacts de composants. Vous devez donc ajouter des autorisations aux compartiments et objets S3 de vos artefacts lorsque vous créez un composant. Pour de plus amples informations, veuillez consulter [Autoriser les appareils principaux à interagir avec les AWS services](#).

Par défaut: `GreengrassV2TokenExchangeRole`

TES_ROLE_ALIAS_NAME

(Facultatif) Le nom de l'alias de AWS IoT rôle qui pointe vers le rôle IAM qui fournit les AWS informations d'identification pour le périphérique principal de Greengrass. Si l'alias de rôle portant ce nom n'existe pas dans votre répertoire Compte AWS, le logiciel AWS IoT Greengrass Core le crée et le pointe vers le rôle IAM que vous spécifiez.

Par défaut : `GreengrassV2TokenExchangeRoleAlias`

COMPONENT_DEFAULT_USER

(Facultatif) Le nom ou l'ID de l'utilisateur et du groupe du système que le logiciel AWS IoT Greengrass Core utilise pour exécuter les composants. Spécifiez l'utilisateur et le groupe, séparés par deux points. Le groupe est facultatif. Par exemple, vous définissez **`ggc_user:ggc_group`** ou **`ggc_user`**.

- Si vous exécutez en tant que root, l'utilisateur et le groupe définis par le fichier de configuration sont utilisés par défaut. Si le fichier de configuration ne définit pas d'utilisateur ni de groupe, la

valeur par défaut est. `ggc_user : ggc_group` S'ils `ggc_group` existent `ggc_user` ou non, le logiciel les crée.

- Si vous l'exécutez en tant qu'utilisateur non root, le logiciel AWS IoT Greengrass Core utilise cet utilisateur pour exécuter les composants.
- Si vous ne spécifiez aucun groupe, le logiciel AWS IoT Greengrass Core utilise le groupe principal de l'utilisateur du système.

Pour de plus amples informations, veuillez consulter [Configurer l'utilisateur qui exécute les composants](#).

DEPLOY_DEV_TOOLS

Définit s'il faut télécharger et déployer le [composant Greengrass CLI](#) dans l'image du conteneur. Vous pouvez utiliser la CLI Greengrass pour développer et déboguer des composants localement.

Important

Nous vous recommandons d'utiliser ce composant uniquement dans les environnements de développement, et non dans les environnements de production. Ce composant permet d'accéder aux informations et aux opérations dont vous n'avez généralement pas besoin dans un environnement de production. Suivez le principe du moindre privilège en déployant ce composant uniquement sur les appareils principaux là où vous en avez besoin.

Par défaut : `false`

INIT_CONFIG

(Facultatif) Le chemin d'accès au fichier de configuration à utiliser pour installer le logiciel AWS IoT Greengrass Core. Vous pouvez utiliser cette option pour configurer de nouveaux appareils principaux Greengrass avec une configuration de noyau spécifique, ou pour spécifier des ressources provisionnées manuellement, par exemple. Vous devez monter votre fichier de configuration sur le chemin que vous spécifiez dans cet argument.

TRUSTED_PLUGIN

Cette fonctionnalité est disponible pour les versions 2.4.0 et ultérieures du composant [Greengrass nucleus](#).

(Facultatif) Le chemin d'accès à un fichier JAR à charger en tant que plugin sécurisé. Utilisez cette option pour fournir des fichiers JAR de plug-in de provisionnement, par exemple à installer avec le provisionnement de [flotte ou le provisionnement personnalisé](#).

THING_POLICY_NAME

Cette fonctionnalité est disponible pour les versions 2.4.0 et ultérieures du composant [Greengrass](#) nucleus.

(Facultatif) Nom de la AWS IoT politique à associer au certificat d' AWS IoT objet de ce périphérique principal. Si la AWS IoT politique portant ce nom n'existe pas dans votre ordinateur, Compte AWS le logiciel AWS IoT Greengrass Core la crée.

Vous devez spécifier `PROVISION=true` pour appliquer cet argument.

Note

Le logiciel AWS IoT Greengrass Core crée une AWS IoT politique permissive par défaut. Vous pouvez limiter cette politique ou créer une politique personnalisée dans laquelle vous limitez les autorisations pour votre cas d'utilisation. Pour de plus amples informations, veuillez consulter [AWS IoT Politique minimale pour les appareils AWS IoT Greengrass V2 principaux](#).

Spécifiez les dépendances à installer

L'instruction RUN du AWS IoT Greengrass Dockerfile prépare l'environnement du conteneur pour exécuter le programme d'installation du logiciel AWS IoT Greengrass Core. Vous pouvez personnaliser les dépendances installées avant que le programme d'installation du logiciel AWS IoT Greengrass Core ne s'exécute dans le conteneur Docker.

Construisez l' AWS IoT Greengrass image

Utilisez le AWS IoT Greengrass Dockerfile pour créer une image de AWS IoT Greengrass conteneur. Vous pouvez utiliser la CLI Docker ou la CLI Docker Compose pour créer l'image et démarrer le conteneur. Vous pouvez également utiliser la CLI Docker pour créer l'image, puis utiliser Docker Compose pour démarrer votre conteneur à partir de cette image.

Docker

1. Sur la machine hôte, exécutez la commande suivante pour passer au répertoire contenant le Dockerfile configuré.

```
cd download-directory/aws-greengrass-docker-nucleus-version
```

2. Exécutez la commande suivante pour créer l'image du AWS IoT Greengrass conteneur à partir du Dockerfile.

```
sudo docker build -t "platform/aws-iot-greengrass:nucleus-version" ./
```

Docker Compose

1. Sur la machine hôte, exécutez la commande suivante pour passer au répertoire contenant le Dockerfile et le fichier Compose.

```
cd download-directory/aws-greengrass-docker-nucleus-version
```

2. Exécutez la commande suivante pour utiliser le fichier Compose pour créer l'image du AWS IoT Greengrass conteneur.

```
docker-compose -f docker-compose.yml build
```

Vous avez créé avec succès l'image du AWS IoT Greengrass conteneur. Le logiciel AWS IoT Greengrass Core est installé sur l'image Docker. Vous pouvez désormais exécuter le logiciel AWS IoT Greengrass Core dans un conteneur Docker.

Exécuter AWS IoT Greengrass dans un conteneur Docker avec provisionnement automatique des ressources

Ce didacticiel explique comment installer et exécuter le logiciel AWS IoT Greengrass Core dans un conteneur Docker avec des AWS ressources automatiquement provisionnées et des outils de développement locaux. Vous pouvez utiliser cet environnement de développement pour explorer les AWS IoT Greengrass fonctionnalités d'un conteneur Docker. Le logiciel nécessite des AWS informations d'identification pour fournir ces ressources et déployer les outils de développement locaux.

Si vous ne pouvez pas fournir AWS d'informations d'identification au conteneur, vous pouvez fournir les AWS ressources dont le périphérique principal a besoin pour fonctionner. Vous pouvez également déployer les outils de développement sur un appareil principal pour l'utiliser comme périphérique de développement. Cela vous permet de fournir moins d'autorisations à l'appareil lorsque vous exécutez le conteneur. Pour de plus amples informations, veuillez consulter [Exécuter AWS IoT Greengrass dans un conteneur Docker avec provisionnement manuel des ressources](#).

Prérequis

Pour terminer ce didacticiel, vous avez besoin des éléments suivants.

- Un Compte AWS. Si vous n'en avez pas, veuillez consulter [Configurez un Compte AWS](#).
- Un utilisateur AWS IAM autorisé à fournir les ressources AWS IoT et IAM pour un appareil principal Greengrass. Le programme d'installation du logiciel AWS IoT Greengrass Core utilise vos AWS informations d'identification pour provisionner automatiquement ces ressources. Pour plus d'informations sur la politique IAM minimale permettant de provisionner automatiquement les ressources, consultez [Politique IAM minimale permettant au programme d'installation de provisionner les ressources](#).
- Une image AWS IoT Greengrass Docker. Vous pouvez [créer une image à partir du AWS IoT Greengrass Dockerfile](#).
- L'ordinateur hôte sur lequel vous exécutez le conteneur Docker doit répondre aux exigences suivantes :
 - Système d'exploitation basé sur Linux doté d'une connexion Internet.
 - [Docker Engine](#) version 18.09 ou ultérieure.
 - (Facultatif) [Docker Compose](#) version 1.22 ou ultérieure. Docker Compose n'est requis que si vous souhaitez utiliser la CLI Docker Compose pour exécuter vos images Docker.

Configuration de vos informations d'identification pour l' AWS

Au cours de cette étape, vous créez un fichier d'informations d'identification sur l'ordinateur hôte contenant vos informations AWS de sécurité. Lorsque vous exécutez l'image AWS IoT Greengrass Docker, vous devez monter le dossier contenant ce fichier d'identification / `root/.aws/` dans le conteneur Docker. Le AWS IoT Greengrass programme d'installation utilise ces informations d'identification pour provisionner des ressources dans votre Compte AWS. Pour plus d'informations sur la politique IAM minimale requise par le programme d'installation pour provisionner

automatiquement les ressources, consultez [Politique IAM minimale permettant au programme d'installation de provisionner les ressources](#).

1. Récupérez l'un des éléments suivants.

- Informations d'identification à long terme pour un utilisateur IAM. Pour plus d'informations sur la façon de récupérer des informations d'identification à long terme, consultez [la section Gestion des clés d'accès pour les utilisateurs IAM](#) dans le guide de l'utilisateur IAM.
- (Recommandé) Informations d'identification temporaires pour un rôle IAM. Pour plus d'informations sur la façon de récupérer des informations d'identification temporaires, consultez la section [Utilisation des informations d'identification de sécurité temporaires AWS CLI dans le](#) guide de l'utilisateur IAM.

2. Créez un dossier dans lequel vous placerez votre fichier d'informations d'identification.

```
mkdir ./greengrass-v2-credentials
```

3. Utilisez un éditeur de texte pour créer un fichier de configuration nommé `credentials` dans le `./greengrass-v2-credentials` dossier.

Par exemple, vous pouvez exécuter la commande suivante pour utiliser GNU nano pour créer le `credentials` fichier.

```
nano ./greengrass-v2-credentials/credentials
```

4. Ajoutez vos AWS informations d'identification au `credentials` fichier au format suivant.

```
[default]
aws_access_key_id = AKIAIOSFODNN7EXAMPLE
aws_secret_access_key = wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
aws_session_token
= AQoEXAMPLEH4aoAH0gNCAPy...truncated...zrkuWJ0gQs8IZZaIv2BXIa2R40lgk
```

Inclure uniquement `aws_session_token` pour les informations d'identification temporaires.

 Important

Supprimez le fichier d'informations d'identification de l'ordinateur hôte après avoir démarré le AWS IoT Greengrass conteneur. Si vous ne supprimez pas le fichier d'informations d'

AWS identification, celles-ci resteront montées dans le conteneur. Pour de plus amples informations, veuillez consulter [Exécuter le logiciel AWS IoT Greengrass Core dans un conteneur](#).

Création d'un fichier d'environnement

Ce didacticiel utilise un fichier d'environnement pour définir les variables d'environnement qui seront transmises au programme d'installation du logiciel AWS IoT Greengrass Core dans le conteneur Docker. Vous pouvez également utiliser `l'--envargument -e or` dans votre `docker run` commande pour définir des variables d'environnement dans le conteneur Docker ou vous pouvez définir les variables dans [un environnement bloc](#) du `docker-compose.yml` fichier.

1. Utilisez un éditeur de texte pour créer un fichier d'environnement nommé `.env`.

Par exemple, sur un système basé sur Linux, vous pouvez exécuter la commande suivante pour utiliser GNU nano afin de créer le `.env` dans le répertoire actuel.

```
nano .env
```

2. Copiez le contenu suivant dans le fichier.

```
GGC_ROOT_PATH=/greengrass/v2
AWS_REGION=region
PROVISION=true
THING_NAME=MyGreengrassCore
THING_GROUP_NAME=MyGreengrassCoreGroup
TES_ROLE_NAME=GreengrassV2TokenExchangeRole
TES_ROLE_ALIAS_NAME=GreengrassCoreTokenExchangeRoleAlias
COMPONENT_DEFAULT_USER=ggc_user:ggc_group
```

Remplacez ensuite les valeurs suivantes.

- */greengrass/v2*. Le dossier racine de Greengrass que vous souhaitez utiliser pour l'installation. Vous utilisez la variable d'GGC_ROOT d'environnement pour définir cette valeur.
- *region*. L' Région AWS endroit où vous avez créé les ressources.
- *MyGreengrassCore*. Le nom du AWS IoT truc. Si l'objet n'existe pas, le programme d'installation le crée. Le programme d'installation télécharge les certificats pour s'authentifier en tant qu' AWS IoT objet.

- ***MyGreengrassCoreGroup***. Le nom du groupe AWS IoT d'objets. Si le groupe d'objets n'existe pas, le programme d'installation le crée et y ajoute l'objet. Si le groupe d'objets existe et fait l'objet d'un déploiement actif, le périphérique principal télécharge et exécute le logiciel spécifié par le déploiement.
- ***GreengrassV2TokenExchangeRole***. Remplacez-le par le nom du rôle d'échange de jetons IAM qui permet au périphérique principal de Greengrass d'obtenir AWS des informations d'identification temporaires. Si le rôle n'existe pas, le programme d'installation le crée, puis crée et attache une politique nommée ***GreengrassV2TokenExchangeRole*** Access. Pour de plus amples informations, veuillez consulter [Autoriser les appareils principaux à interagir avec les AWS services](#).
- ***GreengrassCoreTokenExchangeRoleAlias***. Alias du rôle d'échange de jetons. Si l'alias de rôle n'existe pas, le programme d'installation le crée et le pointe vers le rôle d'échange de jetons IAM que vous spécifiez. Pour plus d'informations, veuillez consulter la rubrique

Note

Vous pouvez définir la variable d'ENVIRONNEMENT `DEPLOY_DEV_TOOLSENVIRONNEMENT` sur `true` pour déployer le [composant Greengrass CLI](#), ce qui vous permet de développer des composants personnalisés dans le conteneur Docker. Nous vous recommandons d'utiliser ce composant uniquement dans les environnements de développement, et non dans les environnements de production. Ce composant permet d'accéder aux informations et aux opérations dont vous n'avez généralement pas besoin dans un environnement de production. Suivez le principe du moindre privilège en déployant ce composant uniquement sur les appareils principaux là où vous en avez besoin.

Exécuter le logiciel AWS IoT Greengrass Core dans un conteneur

Ce didacticiel vous montre comment démarrer l'image Docker que vous avez créée dans un conteneur Docker. Vous pouvez utiliser la CLI Docker ou la CLI Docker Compose pour exécuter l'image logicielle AWS IoT Greengrass Core dans un conteneur Docker.

Docker

1. Exécutez la commande suivante pour démarrer le conteneur Docker.

```
docker run --rm --init -it --name docker-image \
```



```
-v path/to/greengrass-v2-credentials:/root/.aws/:ro \  
--env-file .env \  
-p 8883 \  
your-container-image:version
```

Cet exemple de commande utilise les arguments suivants pour [docker run](#) :

- [--rm](#). Nettoie le conteneur à sa sortie.
- [--init](#). Utilise un processus d'initialisation dans le conteneur.

 Note

L'`--init` argument est nécessaire pour arrêter le logiciel AWS IoT Greengrass Core lorsque vous arrêtez le conteneur Docker.

- [-it](#). (Facultatif) Exécute le conteneur Docker au premier plan en tant que processus interactif. Vous pouvez le remplacer par l'`-d` argument pour exécuter le conteneur Docker en mode détaché à la place. Pour plus d'informations, consultez la section [Détachée ou avant-plan](#) dans la documentation Docker.
- [--name](#). Exécute un conteneur nommé `aws-iot-greengrass`
- [-v](#). Monte un volume dans le conteneur Docker pour que le fichier de configuration et les fichiers de certificat puissent être AWS IoT Greengrass exécutés dans le conteneur.
- [--env-file](#). (Facultatif) Spécifie le fichier d'environnement pour définir les variables d'environnement qui seront transmises au programme d'installation du logiciel AWS IoT Greengrass Core dans le conteneur Docker. Cet argument n'est obligatoire que si vous avez créé un [fichier d'environnement](#) pour définir des variables d'environnement. Si vous n'avez pas créé de fichier d'environnement, vous pouvez utiliser des `--env` arguments pour définir des variables d'environnement directement dans votre commande Docker run.
- [-p](#). (Facultatif) Publie le port du conteneur 8883 sur la machine hôte. Cet argument est obligatoire si vous souhaitez vous connecter et communiquer via MQTT car il AWS IoT Greengrass utilise le port 8883 pour le trafic MQTT. Pour ouvrir d'autres ports, utilisez des `-p` arguments supplémentaires.

 Note

Pour exécuter votre conteneur Docker avec une sécurité accrue, vous pouvez utiliser les `--cap-add` arguments `--cap-drop` et pour activer de manière sélective les fonctionnalités Linux de votre conteneur. Pour plus d'informations, consultez [Privilèges d'exécution et fonctionnalités Linux](#) dans la documentation Docker.

2. Supprimez les informations d'identification `./greengrass-v2-credentials` de l'appareil hôte.

```
rm -rf ./greengrass-v2-credentials
```

 Important

Vous supprimez ces informations d'identification, car elles fournissent des autorisations étendues dont le périphérique principal n'a besoin que lors de la configuration. Si vous ne supprimez pas ces informations d'identification, les composants Greengrass et les autres processus exécutés dans le conteneur peuvent y accéder. Si vous devez fournir des AWS informations d'identification à un composant Greengrass, utilisez le service d'échange de jetons. Pour de plus amples informations, veuillez consulter [Interagissez avec les AWS services](#).

Docker Compose

1. Utilisez un éditeur de texte pour créer un fichier Docker Compose nommé `docker-compose.yml`.

Par exemple, sur un système basé sur Linux, vous pouvez exécuter la commande suivante pour utiliser GNU nano afin de créer le `docker-compose.yml` dans le répertoire actuel.

```
nano docker-compose.yml
```

 Note

Vous pouvez également télécharger et utiliser la dernière version du fichier Compose AWS fourni à partir de [GitHub](#).

2. Ajoutez le contenu suivant au fichier Compose. Votre fichier doit être similaire à l'exemple suivant : *docker-image* Remplacez-le par le nom de votre image Docker.

```
version: '3.7'

services:
  greengrass:
    init: true
    container_name: aws-iot-greengrass
    image: docker-image
    volumes:
      - ./greengrass-v2-credentials:/root/.aws/:ro
    env_file: .env
    ports:
      - "8883:8883"
```

Les paramètres suivants de cet exemple de fichier Compose sont facultatifs :

- **ports**—Publie les ports du conteneur 8883 sur la machine hôte. Ce paramètre est obligatoire si vous souhaitez vous connecter et communiquer via MQTT car il AWS IoT Greengrass utilise le port 8883 pour le trafic MQTT.
- **env_file**—Spécifie le fichier d'environnement pour définir les variables d'environnement qui seront transmises au programme d'installation du logiciel AWS IoT Greengrass Core dans le conteneur Docker. Ce paramètre n'est obligatoire que si vous avez créé un [fichier d'environnement](#) pour définir des variables d'environnement. Si vous n'avez pas créé de fichier d'environnement, vous pouvez utiliser le paramètre d'[environnement](#) pour définir les variables directement dans votre fichier Compose.

 Note

Pour exécuter votre conteneur Docker avec une sécurité accrue, vous pouvez utiliser `cap_drop` et `cap_add` dans votre fichier Compose pour activer de manière sélective

les fonctionnalités Linux de votre conteneur. Pour plus d'informations, consultez [Privilèges d'exécution et fonctionnalités Linux](#) dans la documentation Docker.

3. Exécutez la commande suivante pour démarrer le conteneur Docker.

```
docker-compose -f docker-compose.yml up
```

4. Supprimez les informations d'identification `./greengrass-v2-credentials` de l'appareil hôte.

```
rm -rf ./greengrass-v2-credentials
```

Important

Vous supprimez ces informations d'identification, car elles fournissent des autorisations étendues dont le périphérique principal n'a besoin que lors de la configuration. Si vous ne supprimez pas ces informations d'identification, les composants Greengrass et les autres processus exécutés dans le conteneur peuvent y accéder. Si vous devez fournir des AWS informations d'identification à un composant Greengrass, utilisez le service d'échange de jetons. Pour de plus amples informations, veuillez consulter [Interagissez avec les AWS services](#).

Étapes suivantes

AWS IoT Greengrass Le logiciel de base s'exécute désormais dans un conteneur Docker. Exécutez la commande suivante pour récupérer l'ID du conteneur en cours d'exécution.

```
docker ps
```

Vous pouvez ensuite exécuter la commande suivante pour accéder au conteneur et explorer les logiciels AWS IoT Greengrass principaux exécutés à l'intérieur du conteneur.

```
docker exec -it container-id /bin/bash
```

Pour plus d'informations sur la création d'un composant simple, voir [Étape 4 : développer et tester un composant sur votre appareil](#) dans [Tutoriel : Débuter avec AWS IoT Greengrass V2](#)

 Note

Lorsque vous exécutez `docker exec` des commandes dans le conteneur Docker, ces commandes ne sont pas enregistrées dans les journaux Docker. Pour enregistrer vos commandes dans les journaux Docker, attachez un shell interactif au conteneur Docker. Pour de plus amples informations, veuillez consulter [Attachez un shell interactif au conteneur Docker](#).

Le fichier journal AWS IoT Greengrass Core est appelé `greengrass.log` et se trouve dans `/greengrass/v2/logs`. Les fichiers journaux des composants se trouvent également dans le même répertoire. Pour copier les journaux de Greengrass dans un répertoire temporaire de l'hôte, exécutez la commande suivante :

```
docker cp container-id:/greengrass/v2/logs /tmp/logs
```

Si vous souhaitez conserver les journaux après la sortie ou la suppression d'un conteneur, nous vous recommandons de lier uniquement le `/greengrass/v2/logs` répertoire au répertoire des journaux temporaires sur l'hôte au lieu de monter l'intégralité du répertoire Greengrass. Pour de plus amples informations, veuillez consulter [Persister les logs Greengrass en dehors du conteneur Docker](#).

Pour arrêter un conteneur AWS IoT Greengrass Docker en cours d'exécution, exécutez `docker stop oudocker-compose -f docker-compose.yml stop`. Cette action est envoyée SIGTERM au processus Greengrass et arrête tous les processus associés qui ont été lancés dans le conteneur. Le conteneur Docker est initialisé avec l'`docker-init` exécutable en tant que processus PID 1, ce qui permet de supprimer les processus zombies restants. Pour plus d'informations, consultez la section [Spécifier un processus d'initialisation](#) dans la documentation Docker.

Pour plus d'informations sur la résolution des problèmes liés AWS IoT Greengrass à l'exécution dans un conteneur Docker, consultez [Résolution des problèmes AWS IoT Greengrass dans un conteneur Docker](#).

Exécuter AWS IoT Greengrass dans un conteneur Docker avec provisionnement manuel des ressources

Ce didacticiel explique comment installer et exécuter le logiciel AWS IoT Greengrass Core dans un conteneur Docker avec des ressources provisionnées AWS manuellement.

Rubriques

- [Prérequis](#)
- [Récupérer des points de AWS IoT terminaison](#)
- [Créez n'importe AWS IoT quoi](#)
- [Créez le certificat d'objet](#)
- [Création d'un rôle d'échange de jetons](#)
- [Télécharger les certificats sur l'appareil](#)
- [Création d'un fichier de configuration](#)
- [Création d'un fichier d'environnement](#)
- [Exécuter le logiciel de AWS IoT Greengrass base dans un conteneur](#)
- [Étapes suivantes](#)

Prérequis

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

- Un Compte AWS. Si vous n'en avez pas, veuillez consulter [Configurez un Compte AWS](#).
- Une image AWS IoT Greengrass Docker. Vous pouvez [créer une image à partir du AWS IoT Greengrass Dockerfile](#).
- L'ordinateur hôte sur lequel vous exécutez le conteneur Docker doit répondre aux exigences suivantes :
 - Système d'exploitation basé sur Linux doté d'une connexion Internet.
 - [Docker Engine](#) version 18.09 ou ultérieure.
 - (Facultatif) [Docker Compose](#) version 1.22 ou ultérieure. Docker Compose n'est requis que si vous souhaitez utiliser la CLI Docker Compose pour exécuter vos images Docker.

Récupérer des points de AWS IoT terminaison

Obtenez les AWS IoT points de terminaison qui vous Compte AWS conviennent et enregistrez-les pour les utiliser ultérieurement. Votre appareil utilise ces points de terminaison pour se connecter à AWS IoT. Procédez comme suit :

1. Obtenez le point de terminaison de AWS IoT données pour votre Compte AWS.

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

La réponse ressemble à l'exemple suivant, si la demande aboutit.

```
{
  "endpointAddress": "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
}
```

2. Obtenez le point de terminaison des informations d' AWS IoT identification pour votre Compte AWS.

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

La réponse ressemble à l'exemple suivant, si la demande aboutit.

```
{
  "endpointAddress": "device-credentials-prefix.credentials.iot.us-
west-2.amazonaws.com"
}
```

Créez n'importe AWS IoT quoi

AWS IoT les objets représentent les appareils et les entités logiques auxquels ils se connectent AWS IoT. Les appareils Greengrass Core sont AWS IoT des objets. Lorsque vous enregistrez un appareil en tant qu' AWS IoT objet, celui-ci peut utiliser un certificat numérique pour s'authentifier. AWS

Dans cette section, vous allez créer un AWS IoT objet qui représente votre appareil.

Pour créer AWS IoT quelque chose

1. Créez n'importe AWS IoT quel objet pour votre appareil. Sur votre ordinateur de développement, exécutez la commande suivante.
 - Remplacez *MyGreengrassCore* par le nom de l'objet à utiliser. Ce nom est également le nom de votre appareil principal Greengrass.

 Note

Le nom de l'objet ne peut pas contenir de caractères deux-points (:).

```
aws iot create-thing --thing-name MyGreengrassCore
```

La réponse ressemble à l'exemple suivant, si la demande aboutit.

```
{
  "thingName": "MyGreengrassCore",
  "thingArn": "arn:aws:iot:us-west-2:123456789012:thing/MyGreengrassCore",
  "thingId": "8cb4b6cd-268e-495d-b5b9-1713d71dbf42"
}
```

2. (Facultatif) Ajoutez l' AWS IoT objet à un nouveau groupe d'objets ou à un groupe d'objets existant. Vous utilisez des groupes d'objets pour gérer des flottes d'appareils principaux de Greengrass. Lorsque vous déployez des composants logiciels sur vos appareils, vous pouvez cibler des appareils individuels ou des groupes d'appareils. Vous pouvez ajouter un appareil à un groupe d'objets avec un déploiement Greengrass actif pour déployer les composants logiciels de ce groupe d'objets sur l'appareil. Procédez comme suit :

- a. (Facultatif) Créez un AWS IoT groupe d'objets.

- Remplacez *MyGreengrassCoreGroup* par le nom du groupe d'objets à créer.

 Note

Le nom du groupe d'objets ne peut pas contenir de deux-points (:).

```
aws iot create-thing-group --thing-group-name MyGreengrassCoreGroup
```

La réponse ressemble à l'exemple suivant, si la demande aboutit.

```
{
  "thingGroupName": "MyGreengrassCoreGroup",
}
```



```
"thingGroupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/MyGreengrassCoreGroup",
"thingGroupId": "4df721e1-ff9f-4f97-92dd-02db4e3f03aa"
}
```

b. Ajoutez l' AWS IoT objet à un groupe d'objets.

- Remplacez *MyGreengrassCore* par le nom de votre AWS IoT objet.
- Remplacez *MyGreengrassCoreGroup* par le nom du groupe d'objets.

```
aws iot add-thing-to-thing-group --thing-name MyGreengrassCore --thing-group-name MyGreengrassCoreGroup
```

La commande n'a aucune sortie si la demande aboutit.

Créez le certificat d'objet

Lorsque vous enregistrez un appareil en tant qu' AWS IoT objet, celui-ci peut utiliser un certificat numérique pour s'authentifier. AWS Ce certificat permet à l'appareil de communiquer avec AWS IoT et AWS IoT Greengrass.

Dans cette section, vous allez créer et télécharger des certificats que votre appareil peut utiliser pour se connecter AWS.

Pour créer le certificat d'objet

1. Créez un dossier dans lequel vous téléchargerez les certificats de l' AWS IoT objet.

```
mkdir greengrass-v2-certs
```

2. Créez et téléchargez les certificats AWS IoT correspondants.

```
aws iot create-keys-and-certificate --set-as-active --certificate-pem-outfile greengrass-v2-certs/device.pem.crt --public-key-outfile greengrass-v2-certs/public.pem.key --private-key-outfile greengrass-v2-certs/private.pem.key
```

La réponse ressemble à l'exemple suivant, si la demande aboutit.

```
{
```

```

"certificateArn": "arn:aws:iot:us-west-2:123456789012:cert/
aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4",
"certificateId":
"aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4",
"certificatePem": "-----BEGIN CERTIFICATE-----
MIICiTCCAfICCQD6m7oRw0uX0jANBgkqhkiG9w
0BAQUFADCBIDELMAkGA1UEBhMCVVMxCzAJBgNVBAGTAldBMRAwDgYDVQQHEwdTZ
WF0dGx1MQ8wDQYDVQQKEwZBbWF6b24xFDASBgNVBAwTC0lBTsBDb25zb2x1MRIw
EAYDVQQDEw1UZXRhbnQ2L2sYWMxH2AdBgkqhkiG9w0BCQEWEG5vb25lQGFTYXpvi5
jb20wHhcNMTEwNDI1MjA0NTIxWhcNMTEwNDI1MjA0NTIxWjCBiDELMAkGA1UEBh
MCVVMxCzAJBgNVBAGTAldBMRAwDgYDVQQHEwdTZWF0dGx1MQ8wDQYDVQQKEwZBb
WF6b24xFDASBgNVBAwTC0lBTsBDb25zb2x1MRIwEAYDVQQDEw1UZXRhbnQ2L2sYWMx
H2AdBgkqhkiG9w0BCQEWEG5vb25lQGFTYXpvi5jb20wgZ8wDQYJKoZIhvcNAQEE
BBQADgY0AMIGJAoGBAMaK0dn+a4GmWIWJ21uUSfwfEvySWtC2XADZ4nB+BLygVI
k60CpiwsZ3G93vUEI03IyNoH/f0wYK8m9TrDHudUZg3qX4waLG5M43q7Wgc/MbQ
ITx0USQv7c7ugFFDzQGBzZswY6786m86gpEibb30hjZnzcvcQAaRHhd1QWIMm2nr
AgMBAAEwDQYJKoZIhvcNAQEFBQADgYEAtCu4nUhVVxYUntneD9+h8Mg9q6q+auN
KyExzyLwaxlAoo7TJHidbtS4J5iNmZgXL0FkbFFBjvSfpJI1J00zbhNYS5f6Guo
EDmFJl0ZxBHjJnyp3780D8uTs7fLvJx79LjSTbNYiytVbZPQUQ5Yaxu2jXnimvw
3rrszlaEXAMPLE=
-----END CERTIFICATE-----",
"keyPair": {
  "PublicKey": "-----BEGIN PUBLIC KEY-----\
MIIBIjANBgkqhkiG9w0BAQ0CAQ8AMIIBCgKCAQEAEXAMPLE1nnyJwKSMHw4h\
MMEXAMPLEUuuN/dMAS3fyce8DW/4+EXAMPLEYjmoF/YVF/gHr99VEEXAMPLE5VF13\
59VK7cEXAMPLE67GK+y+jikqX0gHh/xJTtwo
+sGpWEXAMPLEDz18x0d2ka4tCzuWEXAMPLEahJbYkCPUBSU8opVkr7qkEXAMPLE1DR6sx2Hocli00Ltu6Fkw91swQWE
\GB3ZPrNh0PzQYvjUSTZeccyNCx2EXAMPLEvp9mQ0UXP6plfgxwKRX2fEXAMPLEda\
hJLXkX3rHU2xbxJSq7D+XEXAMPLEcw+LyFhI5mgFR188eGdsAEXAMPLE1nI9EesG\
FQIDAQAB\
-----END PUBLIC KEY-----\
",
  "PrivateKey": "-----BEGIN RSA PRIVATE KEY-----\
key omitted for security reasons\
-----END RSA PRIVATE KEY-----\
"
}
}

```

Enregistrez le nom de ressource Amazon (ARN) du certificat afin de l'utiliser pour configurer le certificat ultérieurement.

Configurez ensuite le certificat d'objet. Pour de plus amples informations, veuillez consulter [Configurer le certificat d'objet](#).

Création d'un rôle d'échange de jetons

Les appareils principaux de Greengrass utilisent un rôle de service IAM, appelé rôle d'échange de jetons, pour autoriser les appels aux services. AWS L'appareil utilise le fournisseur AWS IoT d'informations d'identification pour obtenir des AWS informations d'identification temporaires pour ce rôle, ce qui lui permet d'interagir avec Amazon Logs AWS IoT, d'envoyer des journaux à Amazon CloudWatch Logs et de télécharger des artefacts de composants personnalisés depuis Amazon S3. Pour de plus amples informations, veuillez consulter [Autoriser les appareils principaux à interagir avec les AWS services](#).

Vous utilisez un alias de AWS IoT rôle pour configurer le rôle d'échange de jetons pour les appareils principaux de Greengrass. Les alias de rôle vous permettent de modifier le rôle d'échange de jetons d'un appareil tout en conservant la même configuration de l'appareil. Pour plus d'informations, consultez la section [Autorisation des appels directs vers AWS des services](#) dans le Guide du AWS IoT Core développeur.

Dans cette section, vous allez créer un rôle IAM d'échange de jetons et un alias de AWS IoT rôle pointant vers le rôle. Si vous avez déjà configuré un appareil principal Greengrass, vous pouvez utiliser son rôle d'échange de jetons et son alias de rôle au lieu d'en créer de nouveaux. Ensuite, vous configurez l'appareil pour AWS IoT qu'il utilise ce rôle et cet alias.

Pour créer un rôle IAM d'échange de jetons

1. Créez un rôle IAM que votre appareil peut utiliser comme rôle d'échange de jetons. Procédez comme suit :
 - a. Créez un fichier contenant le document de politique de confiance requis par le rôle d'échange de jetons.

Par exemple, sur un système basé sur Linux, vous pouvez exécuter la commande suivante pour utiliser GNU nano pour créer le fichier.

```
nano device-role-trust-policy.json
```

Copiez le code JSON suivant dans le fichier.

```
{
```

```
"Version": "2012-10-17",
"Statement": [
  {
    "Effect": "Allow",
    "Principal": {
      "Service": "credentials.iot.amazonaws.com"
    },
    "Action": "sts:AssumeRole"
  }
]
```

- b. Créez le rôle d'échange de jetons avec le document de politique de confiance.
- Remplacez *GreengrassV2TokenExchangeRole* par le nom du rôle IAM à créer.

```
aws iam create-role --role-name GreengrassV2TokenExchangeRole --assume-role-
policy-document file://device-role-trust-policy.json
```

La réponse ressemble à l'exemple suivant, si la demande aboutit.

```
{
  "Role": {
    "Path": "/",
    "RoleName": "GreengrassV2TokenExchangeRole",
    "RoleId": "AR0AZ2YMUHYHK50KM77FB",
    "Arn": "arn:aws:iam::123456789012:role/GreengrassV2TokenExchangeRole",
    "CreateDate": "2021-02-06T00:13:29+00:00",
    "AssumeRolePolicyDocument": {
      "Version": "2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Principal": {
            "Service": "credentials.iot.amazonaws.com"
          },
          "Action": "sts:AssumeRole"
        }
      ]
    }
  }
}
```

- c. Créez un fichier contenant le document de politique d'accès requis par le rôle d'échange de jetons.

Par exemple, sur un système basé sur Linux, vous pouvez exécuter la commande suivante pour utiliser GNU nano pour créer le fichier.

```
nano device-role-access-policy.json
```

Copiez le code JSON suivant dans le fichier.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams",
        "s3:GetBucketLocation"
      ],
      "Resource": "*"
    }
  ]
}
```

Note

Cette politique d'accès n'autorise pas l'accès aux artefacts des composants dans les compartiments S3. Pour déployer des composants personnalisés qui définissent des artefacts dans Amazon S3, vous devez ajouter des autorisations au rôle afin de permettre à votre appareil principal de récupérer les artefacts des composants. Pour de plus amples informations, veuillez consulter [Autoriser l'accès aux compartiments S3 pour les artefacts des composants](#).

Si vous ne possédez pas encore de compartiment S3 pour les artefacts des composants, vous pouvez ajouter ces autorisations ultérieurement après avoir créé un compartiment.

- d. Créez la politique IAM à partir du document de stratégie.
- Remplacez *GreengrassV2TokenExchangeRoleAccess* par le nom de la politique IAM à créer.

```
aws iam create-policy --policy-name GreengrassV2TokenExchangeRoleAccess --  
policy-document file://device-role-access-policy.json
```

La réponse ressemble à l'exemple suivant, si la demande aboutit.

```
{  
  "Policy": {  
    "PolicyName": "GreengrassV2TokenExchangeRoleAccess",  
    "PolicyId": "ANPAZ2YMUHYHACI7C5Z66",  
    "Arn": "arn:aws:iam::123456789012:policy/  
GreengrassV2TokenExchangeRoleAccess",  
    "Path": "/",  
    "DefaultVersionId": "v1",  
    "AttachmentCount": 0,  
    "PermissionsBoundaryUsageCount": 0,  
    "IsAttachable": true,  
    "CreateDate": "2021-02-06T00:37:17+00:00",  
    "UpdateDate": "2021-02-06T00:37:17+00:00"  
  }  
}
```

- e. Associez la politique IAM au rôle d'échange de jetons.
- Remplacez *GreengrassV2TokenExchangeRole* par le nom du rôle IAM.
 - Remplacez l'ARN de la stratégie par l'ARN de la stratégie IAM que vous avez créée à l'étape précédente.

```
aws iam attach-role-policy --role-name GreengrassV2TokenExchangeRole --policy-  
arn arn:aws:iam::123456789012:policy/GreengrassV2TokenExchangeRoleAccess
```

La commande n'a aucune sortie si la demande aboutit.

2. Créez un alias de AWS IoT rôle qui pointe vers le rôle d'échange de jetons.

- Remplacez *GreengrassCoreTokenExchangeRoleAlias* par le nom de l'alias de rôle à créer.
- Remplacez l'ARN du rôle par l'ARN du rôle IAM que vous avez créé à l'étape précédente.

```
aws iot create-role-alias --role-alias GreengrassCoreTokenExchangeRoleAlias --role-arn arn:aws:iam::123456789012:role/GreengrassV2TokenExchangeRole
```

La réponse ressemble à l'exemple suivant, si la demande aboutit.

```
{
  "roleAlias": "GreengrassCoreTokenExchangeRoleAlias",
  "roleAliasArn": "arn:aws:iot:us-west-2:123456789012:rolealias/GreengrassCoreTokenExchangeRoleAlias"
}
```

Note

Pour créer un alias de rôle, vous devez être autorisé à transmettre le rôle IAM d'échange de jetons à AWS IoT. Si vous recevez un message d'erreur lorsque vous essayez de créer un alias de rôle, vérifiez que votre AWS utilisateur dispose de cette autorisation. Pour plus d'informations, consultez la section [Octroi à un utilisateur des autorisations lui permettant de transférer un rôle à un AWS service](#) dans le Guide de Gestion des identités et des accès AWS l'utilisateur.

3. Créez et attachez une AWS IoT politique qui permet à votre appareil principal Greengrass d'utiliser l'alias de rôle pour assumer le rôle d'échange de jetons. Si vous avez déjà configuré un appareil principal Greengrass, vous pouvez associer sa AWS IoT politique d'alias de rôle au lieu d'en créer une nouvelle. Procédez comme suit :
 - a. (Facultatif) Créez un fichier contenant le document AWS IoT de politique requis par l'alias de rôle.

Par exemple, sur un système basé sur Linux, vous pouvez exécuter la commande suivante pour utiliser GNU nano pour créer le fichier.

```
nano greengrass-v2-iot-role-alias-policy.json
```

Copiez le code JSON suivant dans le fichier.

- Remplacez l'ARN de la ressource par l'ARN de votre alias de rôle.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iot:AssumeRoleWithCertificate",
      "Resource": "arn:aws:iot:us-west-2:123456789012:rolealias/
GreengrassCoreTokenExchangeRoleAlias"
    }
  ]
}
```

b. Créez une AWS IoT politique à partir du document de stratégie.

- Remplacez *GreengrassCoreTokenExchangeRoleAliasPolicy* par le nom de la AWS IoT politique à créer.

```
aws iot create-policy --policy-name GreengrassCoreTokenExchangeRoleAliasPolicy
--policy-document file://greengrass-v2-iot-role-alias-policy.json
```

La réponse ressemble à l'exemple suivant, si la demande aboutit.

```
{
  "policyName": "GreengrassCoreTokenExchangeRoleAliasPolicy",
  "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassCoreTokenExchangeRoleAliasPolicy",
  "policyDocument": "{
    \\\"Version\\\": \\\"2012-10-17\\\",
    \\\"Statement\\\": [
      {
        \\\"Effect\\\": \\\"Allow\\\",
        \\\"Action\\\": \\\"iot:AssumeRoleWithCertificate\\\",
        \\\"Resource\\\": \\\"arn:aws:iot:us-west-2:123456789012:rolealias/
GreengrassCoreTokenExchangeRoleAlias\\\"
      }
    ]
  }"
```



```
  }",  
  "policyVersionId": "1"  
}
```

- c. Joignez la AWS IoT politique au certificat de l' AWS IoT objet.
- Remplacez *GreengrassCoreTokenExchangeRoleAliasPolicy* par le nom de la AWS IoT politique d'alias de rôle.
 - Remplacez l'ARN cible par l'ARN du certificat associé à votre AWS IoT objet.

```
aws iot attach-policy --policy-name GreengrassCoreTokenExchangeRoleAliasPolicy  
--target arn:aws:iot:us-west-2:123456789012:cert/  
aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4
```

La commande n'a aucune sortie si la demande aboutit.

Télécharger les certificats sur l'appareil

Auparavant, vous avez téléchargé le certificat de votre appareil sur votre ordinateur de développement. Dans cette section, vous allez télécharger le certificat de l'autorité de certification racine (CA) Amazon. Ensuite, si vous prévoyez d'exécuter le logiciel AWS IoT Greengrass Core dans Docker sur un autre ordinateur que votre ordinateur de développement, vous copiez les certificats sur cet ordinateur hôte. Le logiciel AWS IoT Greengrass Core utilise ces certificats pour se connecter au service AWS IoT cloud.

Pour télécharger des certificats sur l'appareil

1. Sur votre ordinateur de développement, téléchargez le certificat Amazon Root Certificate Authority (CA). AWS IoT les certificats sont associés par défaut au certificat de l'autorité de certification racine d'Amazon.

Linux or Unix

```
sudo curl -o ./greengrass-v2-certs/AmazonRootCA1.pem https://  
www.amazontrust.com/repository/AmazonRootCA1.pem
```

Windows Command Prompt (CMD)

```
curl -o .\greengrass-v2-certs\AmazonRootCA1.pem https://www.amazontrust.com/repository/AmazonRootCA1.pem
```

PowerShell

```
iwr -Uri https://www.amazontrust.com/repository/AmazonRootCA1.pem -OutFile .\greengrass-v2-certs\AmazonRootCA1.pem
```

2. Si vous prévoyez d'exécuter le logiciel AWS IoT Greengrass Core dans Docker sur un appareil différent de celui de votre ordinateur de développement, copiez les certificats sur l'ordinateur hôte. Si SSH et SCP sont activés sur l'ordinateur de développement et sur l'ordinateur hôte, vous pouvez utiliser la scp commande de votre ordinateur de développement pour transférer les certificats. Remplacez *device-ip-address* par l'adresse IP de votre ordinateur hôte.

```
scp -r greengrass-v2-certs/ device-ip-address:~
```

Création d'un fichier de configuration

1. Sur l'ordinateur hôte, créez un dossier dans lequel vous placez votre fichier de configuration.

```
mkdir ./greengrass-v2-config
```

2. Utilisez un éditeur de texte pour créer un fichier de configuration nommé config.yaml dans le ./greengrass-v2-config dossier.

Par exemple, vous pouvez exécuter la commande suivante pour utiliser GNU nano afin de créer le config.yaml.

```
nano ./greengrass-v2-config/config.yaml
```

3. Copiez le contenu YAML suivant dans le fichier. Ce fichier de configuration partiel spécifie les paramètres du système et les paramètres du noyau de Greengrass.

```
---
system:
  certificateFilePath: "/tmp/certs/device.pem.crt"
```

```
privateKeyPath: "/tmp/certs/private.pem.key"
rootCaPath: "/tmp/certs/AmazonRootCA1.pem"
rootpath: "/greengrass/v2"
thingName: "MyGreengrassCore"
services:
  aws.greengrass.Nucleus:
    componentType: "NUCLEUS"
    version: "nucleus-version"
    configuration:
      awsRegion: "region"
      iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
      iotDataEndpoint: "device-data-prefix-ats.iot.region.amazonaws.com"
      iotCredEndpoint: "device-credentials-prefix.credentials.region.amazonaws.com"
```

Remplacez ensuite les valeurs suivantes :

- `/tmp/certs`. Le répertoire du conteneur Docker dans lequel vous montez les certificats téléchargés lorsque vous démarrez le conteneur.
- `/greengrass/v2`. Le dossier racine de Greengrass que vous souhaitez utiliser pour l'installation. Vous utilisez la variable d'GGC_ROOTEnvironnement pour définir cette valeur.
- `MyGreengrassCore`. Le nom du AWS IoT truc.
- `nucleus-version`. Version du logiciel AWS IoT Greengrass Core à installer. Cette valeur doit correspondre à la version de l'image Docker ou du Dockerfile que vous avez téléchargé. Si vous avez téléchargé l'image Greengrass Docker avec la `latest` balise, utilisez-la **docker inspect image-id** pour voir la version de l'image.
- `region`. L' Région AWS endroit où vous avez créé vos AWS IoT ressources. Vous devez également spécifier la même valeur pour la variable d'AWS_REGIONEnvironnement dans votre [fichier d'environnement](#).
- `GreengrassCoreTokenExchangeRoleAlias`. Alias du rôle d'échange de jetons.
- `device-data-prefix`. Le préfixe de votre point de terminaison AWS IoT de données.
- `device-credentials-prefix`. Le préfixe du point de terminaison de vos AWS IoT informations d'identification.

Création d'un fichier d'environnement

Ce didacticiel utilise un fichier d'environnement pour définir les variables d'environnement qui seront transmises au programme d'installation du logiciel AWS IoT Greengrass Core dans le conteneur

Docker. Vous pouvez également utiliser `l'--envargument -e or` dans votre docker `run` commande pour définir des variables d'environnement dans le conteneur Docker ou vous pouvez définir les variables dans [un environnement bloc](#) du `docker-compose.yml` fichier.

1. Utilisez un éditeur de texte pour créer un fichier d'environnement nommé `.env`.

Par exemple, sur un système basé sur Linux, vous pouvez exécuter la commande suivante pour utiliser GNU nano afin de créer le `.env` dans le répertoire actuel.

```
nano .env
```

2. Copiez le contenu suivant dans le fichier.

```
GGC_ROOT_PATH=/greengrass/v2
AWS_REGION=region
PROVISION=false
COMPONENT_DEFAULT_USER=ggc_user:ggc_group
INIT_CONFIG=/tmp/config/config.yaml
```

Remplacez ensuite les valeurs suivantes.

- */greengrass/v2*. Le chemin d'accès au dossier racine à utiliser pour installer le logiciel AWS IoT Greengrass Core.
- *region*. L' Région AWS endroit où vous avez créé vos AWS IoT ressources. Vous devez spécifier la même valeur pour le paramètre `awsRegion` de configuration dans votre [fichier de configuration](#).
- */tmp/config/*. Le dossier dans lequel vous montez le fichier de configuration lorsque vous démarrez le conteneur Docker.

Note

Vous pouvez définir la variable d'`DEPLOY_DEV_TOOLS` environnement sur `true` pour déployer le [composant Greengrass CLI](#), ce qui vous permet de développer des composants personnalisés dans le conteneur Docker. Nous vous recommandons d'utiliser ce composant uniquement dans les environnements de développement, et non dans les environnements de production. Ce composant permet d'accéder à des informations et à des opérations dont vous n'avez généralement pas besoin dans un

environnement de production. Respectez le principe du moindre privilège en déployant ce composant uniquement sur les appareils principaux là où vous en avez besoin.

Exécuter le logiciel de AWS IoT Greengrass base dans un conteneur

Ce didacticiel explique comment démarrer l'image Docker que vous avez créée dans un conteneur Docker. Vous pouvez utiliser la CLI Docker ou la CLI Docker Compose pour exécuter l'image logicielle AWS IoT Greengrass Core dans un conteneur Docker.

Docker

- Ce didacticiel vous montre comment démarrer l'image Docker que vous avez créée dans un conteneur Docker.

```
docker run --rm --init -it --name docker-image \  
-v path/to/greengrass-v2-config:/tmp/config:ro \  
-v path/to/greengrass-v2-certs:/tmp/certs:ro \  
--env-file .env \  
-p 8883 \  
your-container-image:version
```

Cet exemple de commande utilise les arguments suivants pour [docker run](#) :

- [--rm](#). Nettoie le conteneur à sa sortie.
- [--init](#). Utilise un processus d'initialisation dans le conteneur.

Note

L'`--init` argument est nécessaire pour arrêter le logiciel AWS IoT Greengrass Core lorsque vous arrêtez le conteneur Docker.

- [-it](#). (Facultatif) Exécute le conteneur Docker au premier plan en tant que processus interactif. Vous pouvez le remplacer par l'`-d` argument pour exécuter le conteneur Docker en mode détaché à la place. Pour plus d'informations, consultez la section [Détachée ou avant-plan](#) dans la documentation Docker.
- [--name](#). Exécute un conteneur nommé `aws-iot-greengrass`

- [-v](#). Monte un volume dans le conteneur Docker pour que le fichier de configuration et les fichiers de certificat puissent être AWS IoT Greengrass exécutés dans le conteneur.
- [--env-file](#). (Facultatif) Spécifie le fichier d'environnement pour définir les variables d'environnement qui seront transmises au programme d'installation du logiciel AWS IoT Greengrass Core dans le conteneur Docker. Cet argument n'est obligatoire que si vous avez créé un [fichier d'environnement](#) pour définir des variables d'environnement. Si vous n'avez pas créé de fichier d'environnement, vous pouvez utiliser des `--env` arguments pour définir des variables d'environnement directement dans votre commande Docker run.
- [-p](#). (Facultatif) Publie le port du conteneur 8883 sur la machine hôte. Cet argument est obligatoire si vous souhaitez vous connecter et communiquer via MQTT car il AWS IoT Greengrass utilise le port 8883 pour le trafic MQTT. Pour ouvrir d'autres ports, utilisez des `-p` arguments supplémentaires.

 Note

Pour exécuter votre conteneur Docker avec une sécurité accrue, vous pouvez utiliser les `--cap-add` arguments `--cap-drop` et pour activer de manière sélective les fonctionnalités Linux de votre conteneur. Pour plus d'informations, consultez [Privilèges d'exécution et fonctionnalités Linux](#) dans la documentation Docker.

Docker Compose

1. Utilisez un éditeur de texte pour créer un fichier Docker Compose nommé `docker-compose.yml`.

Par exemple, sur un système basé sur Linux, vous pouvez exécuter la commande suivante pour utiliser GNU nano afin de créer le `docker-compose.yml` dans le répertoire actuel.

```
nano docker-compose.yml
```

 Note

Vous pouvez également télécharger et utiliser la dernière version du fichier Compose AWS fourni à partir de [GitHub](#).

2. Ajoutez le contenu suivant au fichier Compose. Votre fichier doit être similaire à l'exemple suivant : `your-container-name:version` Remplacez-le par le nom de votre image Docker.

```
version: '3.7'

services:
  greengrass:
    init: true
    build:
      context: .
    container_name: aws-iot-greengrass
    image: your-container-name:version
    volumes:
      - /path/to/greengrass-v2-config:/tmp/config:ro
      - /path/to/greengrass-v2-certs:/tmp/certs:ro
    env_file: .env
    ports:
      - "8883:8883"
```

Les paramètres suivants de cet exemple de fichier Compose sont facultatifs :

- `ports`—Publie les ports du conteneur 8883 sur la machine hôte. Ce paramètre est obligatoire si vous souhaitez vous connecter et communiquer via MQTT car il AWS IoT Greengrass utilise le port 8883 pour le trafic MQTT.
- `env_file`—Spécifie le fichier d'environnement pour définir les variables d'environnement qui seront transmises au programme d'installation du logiciel AWS IoT Greengrass Core dans le conteneur Docker. Ce paramètre n'est obligatoire que si vous avez créé un [fichier d'environnement](#) pour définir des variables d'environnement. Si vous n'avez pas créé de fichier d'environnement, vous pouvez utiliser le paramètre d'[environnement](#) pour définir les variables directement dans votre fichier Compose.

Note

Pour exécuter votre conteneur Docker avec une sécurité accrue, vous pouvez utiliser `cap_drop` et `cap_add` dans votre fichier Compose pour activer de manière sélective les fonctionnalités Linux de votre conteneur. Pour plus d'informations, consultez [Privilèges d'exécution et fonctionnalités Linux](#) dans la documentation Docker.

3. Exécutez la commande suivante pour démarrer le conteneur.

```
docker-compose -f docker-compose.yml up
```

Étapes suivantes

AWS IoT Greengrass Le logiciel de base s'exécute désormais dans un conteneur Docker. Exécutez la commande suivante pour récupérer l'ID du conteneur en cours d'exécution.

```
docker ps
```

Vous pouvez ensuite exécuter la commande suivante pour accéder au conteneur et explorer les logiciels AWS IoT Greengrass principaux exécutés à l'intérieur du conteneur.

```
docker exec -it container-id /bin/bash
```

Pour plus d'informations sur la création d'un composant simple, voir [Étape 4 : développer et tester un composant sur votre appareil](#) [Tutoriel : Débuter avec AWS IoT Greengrass V2](#)

Note

Lorsque vous exécutez `docker exec` des commandes dans le conteneur Docker, ces commandes ne sont pas enregistrées dans les journaux Docker. Pour enregistrer vos commandes dans les journaux Docker, attachez un shell interactif au conteneur Docker. Pour de plus amples informations, veuillez consulter [Attachez un shell interactif au conteneur Docker](#).

Le fichier journal AWS IoT Greengrass Core est appelé `greengrass.log` et se trouve dans `/greengrass/v2/logs`. Les fichiers journaux des composants se trouvent également dans le même répertoire. Pour copier les journaux de Greengrass dans un répertoire temporaire de l'hôte, exécutez la commande suivante :

```
docker cp container-id:/greengrass/v2/logs /tmp/logs
```

Si vous souhaitez conserver les journaux après la sortie ou la suppression d'un conteneur, nous vous recommandons de lier uniquement le `/greengrass/v2/logs` répertoire au répertoire des journaux

temporaires sur l'hôte au lieu de monter l'intégralité du répertoire Greengrass. Pour de plus amples informations, veuillez consulter [Persister les logs Greengrass en dehors du conteneur Docker](#).

Pour arrêter un conteneur AWS IoT Greengrass Docker en cours d'exécution, exécutez `docker stop oudocker-compose -f docker-compose.yml stop`. Cette action est envoyée SIGTERM au processus Greengrass et arrête tous les processus associés qui ont été lancés dans le conteneur. Le conteneur Docker est initialisé avec l'`docker-init` exécutable en tant que processus PID 1, ce qui permet de supprimer les processus zombies restants. Pour plus d'informations, consultez la section [Spécifier un processus d'initialisation](#) dans la documentation Docker.

Pour plus d'informations sur la résolution des problèmes liés AWS IoT Greengrass à l'exécution dans un conteneur Docker, consultez [Résolution des problèmes AWS IoT Greengrass dans un conteneur Docker](#).

Résolution des problèmes AWS IoT Greengrass dans un conteneur Docker

Utilisez les informations suivantes pour résoudre les problèmes liés à l'exécution AWS IoT Greengrass dans un conteneur Docker et pour résoudre les problèmes liés AWS IoT Greengrass au conteneur Docker.

Rubriques

- [Résolution des problèmes liés à l'exécution du conteneur Docker](#)
- [Débogage AWS IoT Greengrass dans un conteneur Docker](#)

Résolution des problèmes liés à l'exécution du conteneur Docker

Utilisez les informations suivantes pour résoudre les problèmes liés à l'exécution AWS IoT Greengrass dans un conteneur Docker.

Rubriques

- [Erreur : impossible d'effectuer une connexion interactive à partir d'un appareil autre que TTY](#)
- [Erreur : options inconnues : - no-include-email](#)
- [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'opération : L'utilisateur : arn:aws:iam : ::user/ <user-name>n'account-idest pas autorisé à effectuer : ecr : on resource : * GetAuthorizationToken](#)
- [Erreur : vous avez atteint votre limite de taux d'attraction](#)

Erreur : impossible d'effectuer une connexion interactive à partir d'un appareil autre que TTY

Cette erreur peut se produire lorsque vous exécutez la `aws ecr get-login-password` commande. Assurez-vous d'avoir installé la dernière AWS CLI version 2 ou version 1. Nous vous recommandons d'utiliser la AWS CLI version 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

Cette erreur peut se produire lorsque vous exécutez la `aws ecr get-login` commande. Assurez-vous que la dernière AWS CLI version est installée (par exemple, `Run :pip install awscli --upgrade --user`). 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.

Erreur : Un pare-feu bloque le partage de fichiers entre les fenêtres et les conteneurs.

Vous pouvez recevoir cette erreur ou un `Firewall Detected` message 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/ <user-name>` n'**account-id** 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 : vous avez atteint votre limite de taux d'attraction

Docker Hub limite le nombre de pull requests que les utilisateurs anonymes et gratuits de Docker Hub peuvent effectuer. Si vous dépassez les limites de débit pour les pull requests anonymes ou gratuites destinées aux utilisateurs, vous recevez l'une des erreurs suivantes :

```
ERROR: toomanyrequests: Too Many Requests.
```

```
You have reached your pull rate limit.
```

Pour résoudre ces erreurs, vous pouvez attendre quelques heures avant d'essayer une autre pull request. Si vous prévoyez de soumettre régulièrement un grand nombre de pull requests, consultez le [site Web de Docker Hub](https://docs.docker.com/docker-hub/) pour obtenir des informations sur les limites de débit et les options d'authentification et de mise à niveau de votre compte Docker.

Débogage 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.

Persister les logs Greengrass en dehors du conteneur Docker

Après avoir arrêté un AWS IoT Greengrass conteneur, vous pouvez utiliser la `docker cp` commande suivante pour copier les journaux Greengrass du conteneur Docker vers un répertoire de journaux temporaire.

```
docker cp container-id:/greengrass/v2/logs /tmp/logs
```

Pour conserver les journaux même après la sortie ou la suppression d'un conteneur, vous devez exécuter le conteneur AWS IoT Greengrass Docker après avoir monté le répertoire par liaison. `/greengrass/v2/logs`

Pour monter le `/greengrass/v2/logs` répertoire par liaison, effectuez l'une des opérations suivantes lorsque vous exécutez un nouveau conteneur AWS IoT Greengrass Docker.

- Incluez `-v /tmp/logs:/greengrass/v2/logs:ro` dans votre `docker run` commande.

Modifiez le `volumes` bloc dans le fichier Compose pour inclure la ligne suivante avant d'exécuter votre `docker-compose up` commande.

```
volumes:  
- /tmp/logs:/greengrass/v2/logs:ro
```

Vous pouvez ensuite consulter vos journaux `/tmp/logs` sur votre hôte pour voir les journaux de Greengrass lorsqu' AWS IoT Greengrass il est exécuté dans le conteneur Docker.

Pour plus d'informations sur l'exécution des conteneurs Greengrass Docker, consultez et [Exécuter AWS IoT Greengrass dans Docker avec provisionnement manuel](#) [Exécuter AWS IoT Greengrass dans Docker avec provisionnement automatique](#)

Attachez un shell interactif au conteneur Docker

Lorsque vous exécutez `docker exec` des commandes dans le conteneur Docker, ces commandes ne sont pas capturées dans les journaux Docker. L'enregistrement de vos commandes dans les journaux Docker peut vous aider à étudier l'état du conteneur Greengrass Docker. Effectuez l'une des actions suivantes :

- Exécutez la commande suivante dans un terminal séparé pour associer l'entrée, la sortie et l'erreur standard de votre terminal au conteneur en cours d'exécution. Cela vous permet de visualiser et de contrôler le conteneur Docker depuis votre terminal actuel.

```
docker attach container-id
```

- Exécutez la commande suivante dans un terminal séparé. Cela vous permet d'exécuter vos commandes en mode interactif, même si le conteneur n'est pas attaché.

```
docker exec -it container-id sh -c "command" > /proc/1/fd/1"
```

Pour un AWS IoT Greengrass dépannage général, voir [Résolution des problèmes](#).

Configuration du logiciel AWS IoT Greengrass de base

Le logiciel AWS IoT Greengrass Core propose des options que vous pouvez utiliser pour configurer le logiciel. Vous pouvez créer des déploiements pour configurer le logiciel AWS IoT Greengrass principal sur chaque appareil principal.

Rubriques

- [Déployez le composant Greengrass nucleus](#)
- [Configurer le noyau Greengrass en tant que service système](#)
- [Contrôlez l'allocation de mémoire grâce aux options JVM](#)
- [Configurer l'utilisateur qui exécute les composants](#)
- [Configuration des limites de ressources système pour les composants](#)

- [Connexion au port 443 ou via un proxy réseau](#)
- [Utiliser un certificat d'appareil signé par une autorité de certification privée](#)
- [Configurer les délais d'expiration et les paramètres de cache du MQTT](#)
- [Configurer Greengrass Nucleus sur le réseau IPv6](#)

Déployez le composant Greengrass nucleus

AWS IoT Greengrass fournit le logiciel AWS IoT Greengrass Core sous forme de composant que vous pouvez déployer sur vos appareils principaux Greengrass. Vous pouvez créer un déploiement pour appliquer la même configuration à plusieurs appareils principaux de Greengrass. Pour plus d'informations, consultez [Noyau de Greengrass](#) et [Mettre à jour le logiciel AWS IoT Greengrass principal \(OTA\)](#).

Configurer le noyau Greengrass en tant que service système

Vous devez configurer le logiciel AWS IoT Greengrass Core en tant que service système dans le système d'initialisation de votre appareil pour effectuer les opérations suivantes :

- Démarrez le logiciel AWS IoT Greengrass Core au démarrage de l'appareil. Il s'agit d'une bonne pratique si vous gérez de grands parcs d'appareils.
- Installez et exécutez les composants du plugin. Plusieurs composants AWS fournis sont des modules d'extension, ce qui leur permet de s'interfacer directement avec le noyau de Greengrass. Pour plus d'informations sur les types de composants, consultez [Types de composants](#).
- Appliquez les mises à jour over-the-air (OTA) au logiciel principal de l'appareil AWS IoT Greengrass principal. Pour de plus amples informations, veuillez consulter [Mettre à jour le logiciel AWS IoT Greengrass principal \(OTA\)](#).
- Permettez aux composants de redémarrer le logiciel AWS IoT Greengrass principal ou le périphérique principal lorsqu'un déploiement met à jour le composant vers une nouvelle version ou met à jour certains paramètres de configuration. Pour plus d'informations, consultez [l'étape du cycle de vie du bootstrap](#).

Important

Sur les appareils Windows Core, vous devez configurer le logiciel AWS IoT Greengrass Core en tant que service système.

Rubriques

- [Configurer le noyau en tant que service système \(Linux\)](#)
- [Configuration du noyau en tant que service système \(Windows\)](#)

Configurer le noyau en tant que service système (Linux)

Les appareils Linux prennent en charge différents systèmes d'initialisation, tels que `initd`, `systemd` et `systemV`. Vous utilisez `--setup-system-service true` cet argument lorsque vous installez le logiciel AWS IoT Greengrass Core pour démarrer le noyau en tant que service système et le configurez pour qu'il soit lancé au démarrage du périphérique. Le programme d'installation configure le logiciel AWS IoT Greengrass Core en tant que service système avec `systemd`.

Vous pouvez également configurer manuellement le noyau pour qu'il s'exécute en tant que service système. L'exemple suivant est un fichier de service pour `systemd`.

```
[Unit]
Description=Greengrass Core

[Service]
Type=simple
PIDFile=/greengrass/v2/alts/loader.pid
RemainAfterExit=no
Restart=on-failure
RestartSec=10
ExecStart=/bin/sh /greengrass/v2/alts/current/distro/bin/loader

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

Après avoir configuré le service système, vous pouvez exécuter les commandes suivantes pour configurer le démarrage du périphérique au démarrage et pour démarrer ou arrêter le logiciel AWS IoT Greengrass Core.

- Pour vérifier l'état du service (`systemd`)

```
sudo systemctl status greengrass.service
```

- Pour permettre au noyau de démarrer au démarrage de l'appareil.

```
sudo systemctl enable greengrass.service
```

- Pour empêcher le noyau de démarrer au démarrage de l'appareil.

```
sudo systemctl disable greengrass.service
```

- Pour démarrer le logiciel AWS IoT Greengrass Core.

```
sudo systemctl start greengrass.service
```

- Pour arrêter le logiciel AWS IoT Greengrass Core.

```
sudo systemctl stop greengrass.service
```

Configuration du noyau en tant que service système (Windows)

Vous utilisez `--setup-system-service true` cet argument lorsque vous installez le logiciel AWS IoT Greengrass Core pour démarrer le noyau en tant que service Windows et le configurez pour qu'il démarre au démarrage du périphérique.

Après avoir configuré le service, vous pouvez exécuter les commandes suivantes pour configurer le démarrage du périphérique au démarrage et pour démarrer ou arrêter le logiciel AWS IoT Greengrass Core. Vous devez exécuter l'invite de commande ou PowerShell en tant qu'administrateur pour exécuter ces commandes.

Windows Command Prompt (CMD)

- Pour vérifier l'état du service

```
sc query "greengrass"
```

- Pour permettre au noyau de démarrer au démarrage de l'appareil.

```
sc config "greengrass" start=auto
```

- Pour empêcher le noyau de démarrer au démarrage de l'appareil.

```
sc config "greengrass" start=disabled
```

- Pour démarrer le logiciel AWS IoT Greengrass Core.

```
sc start "greengrass"
```

- Pour arrêter le logiciel AWS IoT Greengrass Core.

```
sc stop "greengrass"
```

Note

Sur les appareils Windows, le logiciel AWS IoT Greengrass Core ignore ce signal d'arrêt lorsqu'il arrête les processus des composants Greengrass. Si le logiciel AWS IoT Greengrass Core ignore le signal d'arrêt lorsque vous exécutez cette commande, attendez quelques secondes, puis réessayez.

PowerShell

- Pour vérifier l'état du service

```
Get-Service -Name "greengrass"
```

- Pour permettre au noyau de démarrer au démarrage de l'appareil.

```
Set-Service -Name "greengrass" -Status stopped -StartupType automatic
```

- Pour empêcher le noyau de démarrer au démarrage de l'appareil.

```
Set-Service -Name "greengrass" -Status stopped -StartupType disabled
```

- Pour démarrer le logiciel AWS IoT Greengrass Core.

```
Start-Service -Name "greengrass"
```

- Pour arrêter le logiciel AWS IoT Greengrass Core.

```
Stop-Service -Name "greengrass"
```


 Note

Sur les appareils Windows, le logiciel AWS IoT Greengrass Core ignore ce signal d'arrêt lorsqu'il arrête les processus des composants Greengrass. Si le logiciel AWS IoT Greengrass Core ignore le signal d'arrêt lorsque vous exécutez cette commande, attendez quelques secondes, puis réessayez.

Contrôlez l'allocation de mémoire grâce aux options JVM

Si vous utilisez AWS IoT Greengrass un appareil dont la mémoire est limitée, vous pouvez utiliser les options de machine virtuelle Java (JVM) pour contrôler la taille maximale du tas, les modes de collecte des déchets et les options du compilateur, qui contrôlent la quantité de mémoire utilisée par le logiciel AWS IoT Greengrass Core. La taille du segment de mémoire dans la JVM détermine la quantité de mémoire qu'une application peut utiliser avant la [collecte des déchets](#) ou avant que l'application ne manque de mémoire. La taille de tas maximale indique la quantité maximale de mémoire que la machine virtuelle Java peut allouer lorsqu'elle augmente le tas dans le cadre d'une activité intensive.

Pour contrôler l'allocation de mémoire, créez un nouveau déploiement ou révisez un déploiement existant qui inclut le composant noyau, et spécifiez vos options JVM dans le paramètre de `jvmOptions` configuration de la [configuration du composant noyau](#).

Selon vos besoins, vous pouvez exécuter le logiciel AWS IoT Greengrass Core avec une allocation de mémoire réduite ou avec une allocation de mémoire minimale.

Allocation de mémoire réduite

Pour exécuter le logiciel AWS IoT Greengrass Core avec une allocation de mémoire réduite, nous vous recommandons d'utiliser l'exemple de mise à jour de fusion de configuration suivant pour définir les options JVM dans votre configuration Nucleus :

```
{
  "jvmOptions": "-XX:+UseSerialGC -XX:TieredStopAtLevel=1"
}
```

Allocation de mémoire minimale

Pour exécuter le logiciel AWS IoT Greengrass Core avec une allocation de mémoire minimale, nous vous recommandons d'utiliser l'exemple de mise à jour de fusion de configuration suivant pour définir les options JVM dans votre configuration Nucleus :

```
{  
  "jvmOptions": "-Xmx32m -XX:+UseSerialGC -Xint"  
}
```

Important

L'exécution du logiciel AWS IoT Greengrass Core avec une allocation de mémoire minimale peut avoir un impact significatif sur les performances des systèmes peu performants, car la JVM effectue plus de traitement lorsqu'elle utilise moins de mémoire. Nous vous recommandons de régler les options pour équilibrer vos besoins en termes de mémoire et de performances.

Ces exemples de mises à jour de fusion de configuration utilisent les options JVM suivantes :

`-XX:+UseSerialGC`

Spécifie l'utilisation de la collecte des déchets en série pour l'espace de mémoire JVM. Le ramasse-miettes en série est plus lent, mais utilise moins de mémoire que les autres implémentations de collecte de déchets JVM.

`-XX:TieredStopAtLevel=1`

Indique à la JVM d'utiliser le compilateur Java just-in-time (JIT) une seule fois. Comme le code compilé JIT utilise de l'espace dans la mémoire de l'appareil, l'utilisation du compilateur JIT à plusieurs reprises consomme plus de mémoire qu'une seule compilation.

`-XmxNNm`

Définit la taille maximale du segment de mémoire JVM.

Important

Si vous définissez une taille de tas maximale trop faible, vous risquez de ralentir les performances ou de provoquer out-of-memory des erreurs. Nous vous recommandons de mesurer votre utilisation actuelle du tas avant de définir une

taille maximale avec l'option `-Xmx`. Configurez votre JVM avec l'option `-XX:NativeMemoryTracking=detail` JVM. Mesurez ensuite votre utilisation actuelle du tas à l'aide de la commande `VM.native_memory` dans l'utilitaire [jcmd](#).

Si la mesure du tas n'est pas une option, utilisez-la `-Xmx64m` comme valeur de départ pour limiter la taille du tas à 64 Mo. Vous pouvez ensuite diminuer progressivement la taille maximale du tas à partir de là. Pour une allocation de mémoire minimale, `-Xmx32m` utilisez-la comme valeur de départ pour limiter la taille du tas à 32 Mo.

Vous pouvez augmenter ou diminuer la `-Xmx` valeur en fonction de vos besoins réels ; toutefois, nous vous recommandons vivement de ne pas définir la taille maximale du segment de mémoire en dessous de 16 Mo. La taille de segment de mémoire JVM nécessaire peut également varier au fil du temps en fonction des composants du plug-in déployés sur le périphérique principal. Si la taille maximale du tas est trop faible pour votre environnement, le logiciel AWS IoT Greengrass Core peut rencontrer des erreurs inattendues en raison d'une mémoire insuffisante. Si les performances sont plus lentes ou si vous rencontrez des erreurs en raison d'une mémoire insuffisante, revenez à un paramètre dont le fonctionnement a été vérifié. Par exemple, si la taille normale de votre tas engagé est 41428KB, utilisez-le `-Xmx40m` pour limiter légèrement l'utilisation du tas.

`-Xint`

Indique à la JVM de ne pas utiliser le compilateur just-in-time (JIT). Au lieu de cela, la JVM s'exécute en mode interprété uniquement. Ce mode est plus lent (potentiellement 20 fois plus lent pour les déploiements sur des systèmes bas de gamme) que l'exécution de code compilé JIT ; toutefois, le code compilé n'utilise aucun espace en mémoire.

Pour plus d'informations sur la création de mises à jour de fusion de configuration, consultez [Mettre à jour les configurations des composants](#).

Configurer l'utilisateur qui exécute les composants

Le logiciel AWS IoT Greengrass Core peut exécuter des processus de composants en tant qu'utilisateur du système et en tant que groupe différent de celui qui exécute le logiciel. Cela augmente la sécurité, car vous pouvez exécuter le logiciel AWS IoT Greengrass principal en tant que root ou en tant qu'utilisateur administrateur, sans accorder ces autorisations aux composants qui s'exécutent sur le périphérique principal.

Le tableau suivant indique les types de composants que le logiciel AWS IoT Greengrass Core peut exécuter en tant qu'utilisateur que vous spécifiez. Pour de plus amples informations, veuillez consulter [Types de composants](#).

Type de composant	Configuration de l'utilisateur du composant
Noyau	 Non
Plugin	 Non
Générique	 Oui
Lambda (non conteneurisé)	 Oui
Lambda (conteneurisé)	 Oui

Vous devez créer l'utilisateur du composant avant de pouvoir le spécifier dans une configuration de déploiement. Sur les appareils Windows, vous devez également enregistrer le nom d'utilisateur et le mot de passe de l'utilisateur dans l'instance du gestionnaire d'informations d'identification du LocalSystem compte. Pour de plus amples informations, veuillez consulter [Configuration d'un utilisateur de composant sur les appareils Windows](#).

Lorsque vous configurez l'utilisateur du composant sur un appareil basé sur Linux, vous pouvez éventuellement spécifier un groupe. Vous spécifiez l'utilisateur et le groupe séparés par deux points (:) au format suivant `:user:group`. Si vous ne spécifiez aucun groupe, le logiciel AWS IoT Greengrass Core utilise par défaut le groupe principal de l'utilisateur. Vous pouvez utiliser le nom ou l'ID pour identifier l'utilisateur et le groupe.

Sur les appareils basés sur Linux, vous pouvez également exécuter des composants en tant qu'utilisateur du système qui n'existe pas, également appelé utilisateur inconnu, pour renforcer la sécurité. Un processus Linux peut signaler tout autre processus exécuté par le même utilisateur. Un utilisateur inconnu n'exécute aucun autre processus. Vous pouvez donc exécuter des composants en tant qu'utilisateur inconnu pour empêcher les composants de signaler d'autres composants sur le périphérique principal. Pour exécuter des composants en tant qu'utilisateur inconnu, spécifiez un ID utilisateur qui n'existe pas sur le périphérique principal. Vous pouvez également spécifier un ID de groupe qui n'existe pas pour fonctionner en tant que groupe inconnu.

Vous pouvez configurer l'utilisateur pour chaque composant et pour chaque périphérique principal.

- Configuration pour un composant

Vous pouvez configurer chaque composant pour qu'il s'exécute avec un utilisateur spécifique à ce composant. Lorsque vous créez un déploiement, vous pouvez spécifier l'utilisateur pour chaque composant dans la `runWith` configuration de ce composant. Le logiciel AWS IoT Greengrass Core exécute les composants en tant qu'utilisateur spécifié si vous les configurez. Sinon, il exécute par défaut les composants en tant qu'utilisateur par défaut que vous configurez pour le périphérique principal. Pour plus d'informations sur la spécification de l'utilisateur du composant dans la configuration de déploiement, consultez le paramètre `runWith` de configuration dans [Créer des déploiements](#).

- Configuration de l'utilisateur par défaut pour un appareil principal

Vous pouvez configurer un utilisateur par défaut que le logiciel AWS IoT Greengrass Core utilise pour exécuter les composants. Lorsque le logiciel AWS IoT Greengrass Core exécute un composant, il vérifie si vous avez spécifié un utilisateur pour ce composant et l'utilise pour exécuter le composant. Si le composant ne spécifie aucun utilisateur, le logiciel AWS IoT Greengrass Core exécute le composant en tant qu'utilisateur par défaut que vous avez configuré pour le périphérique principal. Pour de plus amples informations, veuillez consulter [Configuration de l'utilisateur du composant par défaut](#).

 Note

Sur les appareils Windows, vous devez spécifier au moins un utilisateur par défaut pour exécuter les composants.

Sur les appareils basés sur Linux, les considérations suivantes s'appliquent si vous ne configurez pas un utilisateur pour exécuter des composants :

- Si vous exécutez le logiciel AWS IoT Greengrass Core en tant qu'utilisateur root, le logiciel n'exécutera aucun composant. Vous devez spécifier un utilisateur par défaut pour exécuter les composants si vous exécutez en tant que root.
- Si vous exécutez le logiciel AWS IoT Greengrass Core en tant qu'utilisateur non root, le logiciel exécute les composants sous le nom de cet utilisateur.

Rubriques

- [Configuration d'un utilisateur de composant sur les appareils Windows](#)
- [Configuration de l'utilisateur du composant par défaut](#)

Configuration d'un utilisateur de composant sur les appareils Windows

Pour configurer un utilisateur de composants sur un appareil Windows

1. Créez l'utilisateur du composant dans le LocalSystem compte de l'appareil.

```
net user /add component-user password
```

2. Utilisez l'[PsExec utilitaire Microsoft](#) pour stocker le nom d'utilisateur et le mot de passe de l'utilisateur du composant dans l'instance Credential Manager du LocalSystem compte.

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

 Note

Sur les appareils Windows, le LocalSystem compte exécute le noyau Greengrass et vous devez utiliser PsExec l'utilitaire pour stocker les informations utilisateur du composant dans LocalSystem le compte. L'application Credential Manager stocke ces informations

dans le compte Windows de l'utilisateur actuellement connecté, plutôt que dans le LocalSystem compte.

Configuration de l'utilisateur du composant par défaut

Vous pouvez utiliser un déploiement pour configurer l'utilisateur par défaut sur un appareil principal. Dans ce déploiement, vous mettez à jour la configuration des [composants du noyau](#).

Note

Vous pouvez également définir l'utilisateur par défaut lorsque vous installez le logiciel AWS IoT Greengrass Core avec l'option `--component-default-user`. Pour de plus amples informations, veuillez consulter [Installation du logiciel AWS IoT Greengrass de base](#).

[Créez un déploiement](#) qui spécifie la mise à jour de configuration suivante pour le `aws.greengrass.Nucleus` composant.

Linux

```
{
  "runWithDefault": {
    "posixUser": "ggc_user:ggc_group"
  }
}
```

Windows

```
{
  "runWithDefault": {
    "windowsUser": "ggc_user"
  }
}
```

Note

L'utilisateur que vous spécifiez doit exister, et le nom d'utilisateur et le mot de passe de cet utilisateur doivent être stockés dans l'instance du gestionnaire d'informations

d'identification du LocalSystem compte sur votre appareil Windows. Pour de plus amples informations, veuillez consulter [Configuration d'un utilisateur de composant sur les appareils Windows](#).

L'exemple suivant définit un déploiement pour un appareil basé sur Linux configuré en tant qu'utilisateur par défaut et ggc_user ggc_group en tant que groupe par défaut. La mise à jour merge de configuration nécessite un objet JSON sérialisé.

```
{
  "components": {
    "aws.greengrass.Nucleus": {
      "version": "2.16.1",
      "configurationUpdate": {
        "merge": "{\"runWithDefault\":{\"posixUser\":\"ggc_user:ggc_group\"}}\"
      }
    }
  }
}
```

Configuration des limites de ressources système pour les composants

Note

Cette fonctionnalité est disponible pour les versions 2.4.0 et ultérieures du composant [Greengrass](#) nucleus. AWS IoT Greengrass ne prend actuellement pas en charge cette fonctionnalité sur les appareils Windows principaux.

Vous pouvez configurer l'utilisation maximale du processeur et de la RAM que les processus de chaque composant peuvent utiliser sur le périphérique principal.

Le tableau suivant indique les types de composants compatibles avec les limites de ressources du système. Pour de plus amples informations, veuillez consulter [Types de composants](#).

Type de composant	Configurer les limites des ressources du système
Noyau	 Non
Plugin	 Non
Générique	 Oui
Lambda (non conteneurisé)	 Oui
Lambda (conteneurisé)	 Non

 Important

Les limites de ressources système ne sont pas prises en charge lorsque vous [exécutez le logiciel AWS IoT Greengrass Core dans un conteneur Docker](#).

Vous pouvez configurer les limites de ressources système pour chaque composant et pour chaque périphérique principal.

- Configuration pour un composant

Vous pouvez configurer chaque composant avec des limites de ressources système spécifiques à ce composant. Lorsque vous créez un déploiement, vous pouvez spécifier les limites de ressources système pour chaque composant du déploiement. Si le composant prend en charge les limites de ressources du système, le logiciel AWS IoT Greengrass principal applique ces limites aux processus du composant. Si vous ne spécifiez aucune limite de ressources système pour un composant, le logiciel AWS IoT Greengrass Core utilise les valeurs par défaut que vous avez configurées pour le périphérique principal. Pour de plus amples informations, veuillez consulter [Créer des déploiements](#).

- Configuration des valeurs par défaut pour un périphérique principal

Vous pouvez configurer les limites de ressources système par défaut que le logiciel AWS IoT Greengrass Core applique aux composants compatibles avec ces limites. Lorsque le logiciel AWS IoT Greengrass Core exécute un composant, il applique les limites de ressources système que vous spécifiez pour ce composant. Si ce composant ne spécifie pas de limites de ressources système, le logiciel AWS IoT Greengrass Core applique les limites de ressources système par défaut que vous configurez pour le périphérique principal. Si vous ne spécifiez pas de limites de ressources système par défaut, le logiciel AWS IoT Greengrass Core n'applique aucune limite de ressources système par défaut. Pour de plus amples informations, veuillez consulter [Configuration des limites de ressources système par défaut](#).

Configuration des limites de ressources système par défaut

Vous pouvez déployer le [composant Greengrass nucleus](#) pour configurer les limites de ressources système par défaut pour un périphérique principal. Pour configurer les limites de ressources système par défaut, [créez un déploiement](#) qui spécifie la mise à jour de configuration suivante pour le `aws.greengrass.Nucleus` composant.

```
{
  "runWithDefault": {
    "systemResourceLimits": {
      "cpu": cpuTimeLimit,
      "memory": memoryLimitInKb
    }
  }
}
```

L'exemple suivant définit un déploiement qui configure la limite de temps du processeur à 2, ce qui équivaut à 50 % d'utilisation sur un appareil doté de 4 cœurs de processeur. Cet exemple configure également l'utilisation de la mémoire à 100 Mo.

```
{
  "components": {
    "aws.greengrass.Nucleus": {
      "version": "2.16.1",
      "configurationUpdate": {
        "merge": "{\"runWithDefault\":{\"systemResourceLimits\":{\"cpus\":2,\"memory\":102400}}}"
      }
    }
  }
}
```

Connexion au port 443 ou via un proxy réseau

AWS IoT Greengrass les périphériques principaux communiquent avec eux AWS IoT Core à l'aide du protocole de messagerie MQTT avec 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 exiger que tout le trafic passe par un proxy avant de se connecter à Internet.

Note

Les appareils principaux de Greengrass qui exécutent le [composant Greengrass nucleus v2.0.3](#) et versions antérieures utilisent le port 8443 pour se connecter au point de terminaison du plan de données. AWS IoT Greengrass Ces appareils doivent être en mesure de se connecter à ce point de terminaison sur le port 8443. Pour de plus amples informations, veuillez consulter [Autoriser le trafic des appareils via un proxy ou un pare-feu](#).

Pour activer la communication dans ces scénarios, AWS IoT Greengrass propose les options de configuration suivantes :

- Communication MQTT via le port 443. Si votre réseau autorise les connexions au port 443, vous pouvez configurer le périphérique principal de Greengrass pour utiliser le port 443 pour le trafic MQTT au lieu du port par défaut 8883. Il peut s'agir d'une connexion directe au port 443 ou d'une connexion via un serveur réseau proxy. Contrairement à la configuration par défaut, qui utilise l'authentification client basée sur des certificats, le MQTT sur le port 443 utilise le [rôle de service de périphérique](#) pour l'authentification.

Pour de plus amples informations, veuillez consulter [Configurer MQTT sur le port 443](#).

- Communication HTTPS via le port 443. Le logiciel AWS IoT Greengrass Core envoie le trafic HTTPS via le port 8443 par défaut, mais vous pouvez le configurer pour utiliser le port 443. AWS IoT Greengrass utilise l'extension TLS ALPN ([Application Layer Protocol Network](#)) pour activer cette connexion. Comme pour la configuration par défaut, le protocole HTTPS sur le port 443 utilise une authentification client basée sur des certificats.

 Important

Pour utiliser ALPN et activer la communication HTTPS sur le port 443, votre appareil principal doit exécuter Java 8 update 252 ou version ultérieure. Toutes les mises à jour de Java version 9 et ultérieures sont également compatibles avec ALPN.

Pour de plus amples informations, veuillez consulter [Configuration du protocole HTTPS sur le port 443](#).

- Connexion via un proxy réseau. Vous pouvez configurer un serveur proxy réseau pour servir d'intermédiaire pour la connexion au périphérique principal de Greengrass. AWS IoT Greengrass prend en charge l'authentification de base pour les proxys HTTP et HTTPS.

Les appareils principaux de Greengrass doivent exécuter [Greengrass nucleus](#) v2.5.0 ou version ultérieure pour utiliser les proxys HTTPS.

Le logiciel AWS IoT Greengrass Core transmet la configuration du proxy aux composants via les variables ALL_PROXYHTTP_PROXY,HTTPS_PROXY, et d'NO_PROXYenvironnement. Les composants doivent utiliser ces paramètres pour se connecter via le proxy. Les composants utilisent des bibliothèques communes (telles que boto3, cURL et le requests package python) qui utilisent généralement ces variables d'environnement par défaut pour établir des connexions. Si un composant spécifie également ces variables d'environnement, AWS IoT Greengrass il ne les remplace pas.

Pour de plus amples informations, veuillez consulter [Configuration d'un proxy réseau](#).

Configurer MQTT sur le port 443

Vous pouvez configurer MQTT sur le port 443 sur les périphériques principaux existants ou lorsque vous installez le logiciel AWS IoT Greengrass Core sur un nouveau périphérique principal.

Rubriques

- [Configurer MQTT sur le port 443 sur les périphériques principaux existants](#)
- [Configurer MQTT sur le port 443 lors de l'installation](#)

Configurer MQTT sur le port 443 sur les périphériques principaux existants

Vous pouvez utiliser un déploiement pour configurer MQTT sur le port 443 sur un périphérique à cœur unique ou sur un groupe de périphériques principaux. Dans ce déploiement, vous mettez à jour la configuration des [composants du noyau](#). Le noyau redémarre lorsque vous mettez à jour sa `mqtt` configuration.

Pour configurer MQTT sur le port 443, [créez un déploiement](#) qui spécifie la mise à jour de configuration suivante pour le `aws.greengrass.Nucleus` composant.

```
{
  "mqtt": {
    "port": 443
  }
}
```

L'exemple suivant définit un déploiement qui configure MQTT sur le port 443. La mise à jour `merge` de configuration nécessite un objet JSON sérialisé.

```
{
  "components": {
    "aws.greengrass.Nucleus": {
      "version": "2.16.1",
      "configurationUpdate": {
        "merge": "{\"mqtt\":{\"port\":443}}"
      }
    }
  }
}
```

```
}
```

Configurer MQTT sur le port 443 lors de l'installation

Vous pouvez configurer MQTT sur le port 443 lorsque vous installez le logiciel AWS IoT Greengrass Core sur un périphérique principal. Utilisez l'argument du `--init-config` programme d'installation pour configurer MQTT sur le port 443. Vous pouvez spécifier cet argument lors de l'installation à l'aide du [provisionnement manuel](#), du provisionnement du [parc](#) ou du [provisionnement personnalisé](#).

Configuration du protocole HTTPS sur le port 443

Cette fonctionnalité nécessite la [Noyau de Greengrass](#) version 2.0.4 ou ultérieure.

Vous pouvez configurer le protocole HTTPS sur le port 443 sur les appareils principaux existants ou lorsque vous installez le logiciel AWS IoT Greengrass principal sur un nouveau périphérique principal.

Rubriques

- [Configurer le protocole HTTPS sur le port 443 sur les appareils principaux existants](#)
- [Configuration du protocole HTTPS sur le port 443 lors de l'installation](#)

Configurer le protocole HTTPS sur le port 443 sur les appareils principaux existants

Vous pouvez utiliser un déploiement pour configurer le protocole HTTPS sur le port 443 sur un périphérique central ou un groupe de périphériques principaux. Dans ce déploiement, vous mettez à jour la configuration des [composants du noyau](#).

Pour configurer le protocole HTTPS sur le port 443, [créez un déploiement](#) qui spécifie la mise à jour de configuration suivante pour le `aws.greengrass.Nucleus` composant.

```
{
  "greengrassDataPlanePort": 443
}
```

L'exemple suivant définit un déploiement qui configure le protocole HTTPS sur le port 443. La mise à jour merge de configuration nécessite un objet JSON sérialisé.

```
{
  "components": {
    "aws.greengrass.Nucleus": {
      "version": "2.16.1",
```

```
"configurationUpdate": {  
  "merge": "{\"greengrassDataPlanePort\":443}"  
}  
}  
}  
}
```

Configuration du protocole HTTPS sur le port 443 lors de l'installation

Vous pouvez configurer le protocole HTTPS sur le port 443 lorsque vous installez le logiciel AWS IoT Greengrass Core sur un périphérique principal. Utilisez l'argument du `--init-config` programme d'installation pour configurer le protocole HTTPS sur le port 443. Vous pouvez spécifier cet argument lors de l'installation à l'aide du [provisionnement manuel](#), du provisionnement du [parc ou du provisionnement personnalisé](#).

Configuration d'un proxy réseau

Suivez la procédure décrite dans cette section pour configurer les appareils principaux de Greengrass afin qu'ils se connectent à Internet via un proxy réseau HTTP ou HTTPS. Pour plus d'informations sur les points de terminaison et les ports utilisés par les appareils principaux, consultez [Autoriser le trafic des appareils via un proxy ou un pare-feu](#).

Important

Si votre appareil principal exécute une version du [noyau Greengrass](#) antérieure à la version 2.4.0, le rôle de votre appareil doit autoriser les autorisations suivantes pour utiliser un proxy réseau :

- `iot:Connect`
- `iot:Publish`
- `iot:Receive`
- `iot:Subscribe`

Cela est nécessaire car le périphérique utilise les AWS informations d'identification du service d'échange de jetons pour authentifier les connexions MQTT auxquelles. AWS IoT L'appareil utilise MQTT pour recevoir et installer des déploiements depuis le AWS Cloud. Votre appareil ne fonctionnera donc pas si vous ne définissez pas ces autorisations sur son rôle. Les appareils utilisent généralement des certificats X.509 pour authentifier les connexions MQTT, mais ils ne peuvent pas le faire pour s'authentifier lorsqu'ils utilisent un proxy.

Pour plus d'informations sur la façon de configurer le rôle de l'appareil, consultez [Autoriser les appareils principaux à interagir avec les AWS services](#).

Rubriques

- [Configuration d'un proxy réseau sur les appareils principaux existants](#)
- [Configuration d'un proxy réseau lors de l'installation](#)
- [Permettre au périphérique principal de faire confiance à un proxy HTTPS](#)
- [L'objet NetworkProxy](#)

Configuration d'un proxy réseau sur les appareils principaux existants

Vous pouvez utiliser un déploiement pour configurer un proxy réseau sur un périphérique central unique ou un groupe de périphériques principaux. Dans ce déploiement, vous mettez à jour la configuration des [composants du noyau](#). Le noyau redémarre lorsque vous mettez à jour sa networkProxy configuration.

Pour configurer un proxy réseau, [créez un déploiement](#) pour le `aws.greengrass.Nucleus` composant qui fusionne la mise à jour de configuration suivante. Cette mise à jour de configuration contient l'objet [NetworkProxy](#).

```
{
  "networkProxy": {
    "noProxyAddresses": "http://192.168.0.1,www.example.com",
    "proxy": {
      "url": "https://my-proxy-server:1100"
    }
  }
}
```

L'exemple suivant définit un déploiement qui configure un proxy réseau. La mise à jour merge de configuration nécessite un objet JSON sérialisé.

```
{
  "components": {
    "aws.greengrass.Nucleus": {
      "version": "2.16.1",
      "configurationUpdate": {
```



```

    "merge": "{\\"networkProxy\\":{\\"noProxyAddresses\\":
\\"http://192.168.0.1,www.example.com\\",\\"proxy\\":{\\"url\\":\\"https://my-proxy-
server:1100\\",\\"username\\":\\"Mary_Major\\",\\"password\\":\\"pass@word1357\\"}}}"
  }
}
}
}

```

Configuration d'un proxy réseau lors de l'installation

Vous pouvez configurer un proxy réseau lorsque vous installez le logiciel AWS IoT Greengrass Core sur un périphérique principal. Utilisez l'argument du programme d'--init-configinstallation pour configurer le proxy réseau. Vous pouvez spécifier cet argument lors de l'installation à l'aide du [provisionnement manuel](#), du provisionnement du [parc ou du provisionnement personnalisé](#).

Permettre au périphérique principal de faire confiance à un proxy HTTPS

Lorsque vous configurez un périphérique principal pour utiliser un proxy HTTPS, vous devez ajouter la chaîne de certificats du serveur proxy à celle du périphérique principal pour lui permettre de faire confiance au proxy HTTPS. Dans le cas contraire, le périphérique principal risque de rencontrer des erreurs lorsqu'il tente d'acheminer le trafic via le proxy. Ajoutez le certificat CA du serveur proxy au fichier de certificat CA racine Amazon de l'appareil principal.

Pour permettre au périphérique principal de faire confiance au proxy HTTPS

1. Recherchez le fichier de certificat Amazon Root CA sur l'appareil principal.
 - Si vous avez installé le logiciel AWS IoT Greengrass Core avec [provisionnement automatique](#), le fichier de certificat racine de l'autorité de certification Amazon existe à `/greengrass/v2/rootCA.pem`adresse.
 - Si vous avez installé le logiciel AWS IoT Greengrass Core avec un [provisionnement manuel ou par flotte](#), le fichier de certificat racine de l'autorité de certification Amazon peut se trouver à `/greengrass/v2/AmazonRootCA1.pem` l'adresse.

Si le certificat CA racine d'Amazon n'existe pas à ces emplacements, enregistrez l'`system.rootCaPath`établissement `/greengrass/v2/config/effectiveConfig.yaml` pour trouver son emplacement.

2. Ajoutez le contenu du fichier de certificat CA du serveur proxy au fichier de certificat CA racine d'Amazon.

L'exemple suivant montre un certificat CA de serveur proxy ajouté au fichier de certificat CA racine d'Amazon.

```
-----BEGIN CERTIFICATE-----
MIIEFTCCA v2gAwIQWgIVAMHSAzWG/5YVRYtRQ0xXUTEpHuEmApzGCSqGSIb3DQEK
\nCwUAhuL9MQswCQwJVUzEPMAVUzEYMBYGA1UECgwP1hem9uLmNvbSBjbmuMRww
... content of proxy CA certificate ...
+vHIR1t0e5JAm5\noTIZGoFbK82A0/n07f/t5PSIDAim9V3Gc3pSXxCCAQoFYnui
GaPULGk1gCE84a0X\n7Rp/1ND/PuMZ/s8Yj1kY2NmYmNjMCAXDTE5MTEyN2cM216
gJMIADggEPADf2/m45hzEXAMPLE=
-----END CERTIFICATE-----

-----BEGIN CERTIFICATE-----
MIIDQTCCAimgF6AwIBAgITBmyfz/5mjAo54vB4ikPmljZKyjANJmApzyMZFo6qBg
ADA5MQswCQYDVQQGEwJVUzEPMA0tMVT8QtPHRh8jrdkGA1UEChMGDV3QQDExBBKW
... content of root CA certificate ...
o/ufQJQWUCyziar1hem9uMRkwFwYVPSHCb2XV4cdFyQzR1KldZwgJcIQ6XUDgHaa
5MsI+yMRQ+hDaXJiobl dXgjUka642M4UwtBV8oK2xJNDd2ZhwLnoQdeXeGADKkpy
rqXRfKoQnoZsG4q5WTP46EXAMPLE
-----END CERTIFICATE-----
```

L'objet NetworkProxy

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

`noProxyAddresses`

(Facultatif) Liste séparée par des virgules d'adresses IP ou de noms d'hôtes exemptés du proxy.

`proxy`

Le proxy auquel se connecter. Cet objet contient les informations suivantes :

`url`

URL du serveur proxy au format `scheme://userinfo@host:port`.

- `scheme`— Le schéma, qui doit être `http` ou `https`.

⚠ Important

Les appareils principaux de Greengrass doivent exécuter [Greengrass nucleus v2.5.0](#) ou version ultérieure pour utiliser les proxys HTTPS.

Si vous configurez un proxy HTTPS, vous devez ajouter le certificat CA du serveur proxy au certificat CA racine Amazon de l'appareil principal. Pour de plus amples informations, veuillez consulter [Permettre au périphérique principal de faire confiance à un proxy HTTPS](#).

- `userinfo`— (Facultatif) Les informations relatives au nom d'utilisateur et au mot de passe. Si vous spécifiez ces informations dans `url`, le périphérique principal de Greengrass ignore les `username` et `password` champs.
- `host`— Le nom d'hôte ou l'adresse IP du serveur proxy.
- `port`— (Facultatif) Le numéro de port. Si vous ne spécifiez pas le port, le périphérique principal de Greengrass utilise les valeurs par défaut suivantes :
 - `http`— 80
 - `https`— 443

`username`

(Facultatif) Le nom d'utilisateur qui authentifie le serveur proxy.

`password`

(Facultatif) Le mot de passe qui authentifie le serveur proxy.

Utiliser un certificat d'appareil signé par une autorité de certification privée

Si vous utilisez une autorité de certification privée (CA) personnalisée, vous devez définir le noyau **greengrassDataPlaneEndpoint** de Greengrass sur **iotdata**. Vous pouvez définir cette option lors du déploiement ou de l'installation à l'aide de l'[argument `--init-config` installer](#).

Vous pouvez personnaliser le point de terminaison du plan de données Greengrass auquel l'appareil se connecte. Vous pouvez définir cette option de configuration **iotdata** pour définir le point de terminaison du plan de données Greengrass sur le même point de terminaison que le point de terminaison de données IoT, que vous pouvez spécifier avec le **iotDataEndpoint**.

Configurer les délais d'expiration et les paramètres de cache du MQTT

Dans l' AWS IoT Greengrass environnement, les composants peuvent utiliser MQTT pour communiquer avec AWS IoT Core. Le logiciel AWS IoT Greengrass Core gère les messages MQTT pour les composants. Lorsque le périphérique principal perd la connexion au AWS Cloud, le logiciel met en cache les messages MQTT pour réessayer ultérieurement lorsque la connexion sera rétablie. Vous pouvez configurer des paramètres tels que les délais d'expiration des messages et la taille du cache. Pour plus d'informations, consultez les paramètres de `mqtt.spooler` configuration `mqtt` et du composant [Greengrass nucleus](#).

AWS IoT Core impose des quotas de service à son courtier de messages MQTT. Ces quotas peuvent s'appliquer aux messages que vous envoyez entre des appareils principaux et AWS IoT Core. Pour plus d'informations, consultez la section sur les [quotas de service du courtier de AWS IoT Core messages](#) dans le Références générales AWS.

Configurer Greengrass Nucleus sur le réseau IPv6

[Greengrass Nucleus communique avec AWS IoT Core Greengrass. APIs](#) APIs Support de Greengrass dans un environnement à IPv6 double pile.

Pour activer les points de terminaison à double pile pour : IPv6

- Ajoutez les propriétés `aws.useDualstackEndpoint=true` du système et `java.net.preferIPv6Addresses=true` `jvmOptions`
- Définissez `s3EndpointType` sur `DUALSTACK`

Définissez cette option lors du [déploiement](#) ou configurez-la manuellement avec l'[argument du --init-config programme d'installation](#). Consultez la section [Utilisation des points de terminaison à double pile Amazon S3](#) pour plus de détails.

Exemple code pour le déploiement :

```
{
  "jvmOptions": "-Daws.useDualstackEndpoint=true",
  "s3EndpointType": "DUALSTACK"
}
```

Exemple **config.yaml** par le biais d'un approvisionnement manuel :

```
---
system:
  ...
services:
  aws.greengrass.Nucleus:
    ...
    configuration:
      ...
      jvmOptions: "-Daws.useDualstackEndpoint=true -Djava.net.preferIPv6Addresses=true"
      s3EndpointType: "DUALSTACK"
```

Mettre à jour le logiciel AWS IoT Greengrass principal (OTA)

Le logiciel de AWS IoT Greengrass base comprend le [composant Greengrass nucleus](#) et d'autres composants optionnels que vous pouvez déployer sur vos appareils pour effectuer des mises à jour over-the-air (OTA) du logiciel. Cette fonctionnalité est intégrée au logiciel de AWS IoT Greengrass base.

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.

Rubriques

- [Prérequis](#)
- [Considérations relatives aux appareils principaux](#)
- [Comportement de mise à jour du noyau Greengrass](#)
- [Effectuer une mise à jour OTA](#)

Prérequis

Les exigences suivantes s'appliquent au déploiement des mises à jour OTA du logiciel AWS IoT Greengrass principal :

- L'appareil principal de Greengrass doit être connecté au pour AWS Cloud recevoir le déploiement.
- L'appareil principal Greengrass doit être correctement configuré et doté de certificats et de clés pour l'authentification avec et. AWS IoT Core AWS IoT Greengrass
- Le logiciel de AWS IoT Greengrass base doit être configuré et exécuté en tant que service système. Les mises à jour OTA ne fonctionnent pas si vous exécutez le noyau à partir du fichier JAR, Greengrass.jar. Pour de plus amples informations, veuillez consulter [Configurer le noyau Greengrass en tant que service système](#).

Considérations relatives aux appareils principaux

Avant d'effectuer une mise à jour OTA, soyez conscient de l'impact sur les principaux appareils que vous mettez à jour et sur leurs appareils clients connectés :

- Le noyau de Greengrass s'arrête.
- Tous les composants exécutés sur le périphérique principal s'arrêtent également. Si ces composants écrivent dans des ressources locales, ils risquent de laisser ces ressources dans un état incorrect s'ils ne sont pas correctement arrêtés. Les composants peuvent utiliser [la communication entre processus](#) pour demander au composant du noyau de différer la mise à jour jusqu'à ce qu'ils aient nettoyé les ressources qu'ils utilisent.
- Lorsque le composant du noyau est arrêté, le périphérique principal perd ses connexions avec les périphériques locaux AWS Cloud et les périphériques. L'appareil principal n'achemine pas les messages depuis les appareils clients lorsqu'il est éteint.
- Les fonctions Lambda de longue durée qui s'exécutent en tant que composants perdent leurs informations d'état dynamique et abandonnent toutes les tâches en attente.

Comportement de mise à jour du noyau Greengrass

Lorsque vous déployez un composant, AWS IoT Greengrass installe les dernières versions prises en charge de toutes les dépendances de ce composant. De ce fait, les nouvelles versions de correctif des composants publics AWS fournis peuvent être automatiquement déployées sur vos appareils principaux si vous ajoutez de nouveaux appareils à un groupe d'objets ou si vous mettez à jour le déploiement qui cible ces appareils. Certaines mises à jour automatiques, telles que la mise à jour du noyau, peuvent provoquer le redémarrage inattendu de vos appareils.

Lorsque la version du [composant Greengrass noyau](#) change, le logiciel AWS IoT Greengrass Core, qui inclut le noyau et tous les autres composants de votre appareil, redémarre pour appliquer

les modifications. En raison de [l'impact sur les appareils principaux](#) lorsque le composant Nucleus est mis à jour, vous souhaitez peut-être contrôler le moment où une nouvelle version du correctif Nucleus est déployée sur vos appareils. Pour ce faire, vous devez inclure directement le composant Greengrass nucleus dans votre déploiement. L'inclusion directe d'un composant signifie que vous incluez une version spécifique de ce composant dans votre configuration de déploiement et que vous ne comptez pas sur les dépendances des composants pour déployer ce composant sur vos appareils. Pour plus d'informations sur la définition des dépendances dans vos recettes de composants, consultez [Format de recette](#).

Consultez le tableau suivant pour comprendre le comportement de mise à jour du composant Greengrass nucleus en fonction de vos actions et de vos configurations de déploiement.

Action	Configuration de déploiement	Comportement de mise à jour
Ajoutez de nouveaux appareils à un groupe d'objets ciblé par un déploiement existant sans modifier le déploiement.	Le déploiement n'inclut pas directement le noyau Greengrass.	Sur les nouveaux appareils , installe la dernière version du correctif de Nucleus qui répond à toutes les exigences de dépendance des composants.
	Le déploiement inclut directement au moins un composant AWS fourni, ou inclut un composant personnalisé qui dépend d'un composant AWS fourni ou du noyau Greengrass.	Sur les appareils existants, ne met pas à jour la version installée du noyau.
Ajoutez de nouveaux appareils à un groupe d'objets ciblé par un déploiement existant sans modifier le déploiement.	Le déploiement inclut directement une version spécifique du noyau Greengrass.	Sur les nouveaux appareils , installe la version de noyau spécifiée. Sur les appareils existants, ne met pas à jour la version installée du noyau.
Créez un nouveau déploiement ou révisez un déploiement existant.	Le déploiement n'inclut pas directement le noyau Greengrass.	Installe sur tous les appareils ciblés la dernière version du correctif du noyau qui répond

Action	Configuration de déploiement	Comportement de mise à jour
	Le déploiement inclut directement au moins un composant AWS fourni, ou inclut un composant personnalisé qui dépend d'un composant AWS fourni ou du noyau Greengrass.	à toutes les exigences de dépendance des composants, y compris sur tous les nouveaux appareils que vous ajoutez au groupe d'objets ciblé.
Créez un nouveau déploiement ou révissez un déploiement existant.	Le déploiement inclut directement une version spécifique du noyau Greengrass.	Installe la version du noyau spécifiée sur tous les appareils ciblés, y compris les nouveaux appareils que vous ajoutez au groupe d'objets ciblé.

Effectuer une mise à jour OTA

Pour effectuer une mise à jour OTA, [créez un déploiement](#) qui inclut le [composant Nucleus](#) et la version à installer.

Désinstallez le logiciel AWS IoT Greengrass Core

Vous pouvez désinstaller le logiciel AWS IoT Greengrass Core pour le supprimer d'un appareil que vous ne souhaitez pas utiliser comme appareil principal Greengrass. Vous pouvez également suivre ces étapes pour nettoyer une installation qui échoue.

Pour désinstaller le logiciel AWS IoT Greengrass Core

1. Si vous exécutez le logiciel en tant que service système, vous devez arrêter, désactiver et supprimer le service. Exécutez les commandes suivantes en fonction de votre système d'exploitation.

Linux

1. Arrêtez le service .


```
sudo systemctl stop greengrass.service
```

2. Désactivez le service.

```
sudo systemctl disable greengrass.service
```

3. Supprimez le service.

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

4. Vérifiez que le service est supprimé.

```
sudo systemctl daemon-reload && sudo systemctl reset-failed
```

Windows (Command Prompt)

Note

Vous devez exécuter Command Prompt en tant qu'administrateur pour exécuter ces commandes.

1. Arrêtez le service .

```
sc stop "greengrass"
```

2. Désactivez le service.

```
sc config "greengrass" start=disabled
```

3. Supprimez le service.

```
sc delete "greengrass"
```

4. Redémarrez le périphérique.

Windows (PowerShell)



Note

Vous devez exécuter ces commandes en PowerShell tant qu'administrateur.

1. Arrêtez le service .

```
Stop-Service -Name "greengrass"
```

2. Désactivez le service.

```
Set-Service -Name "greengrass" -Status stopped -StartupType disabled
```

3. Supprimez le service.

- Pour la PowerShell version 6.0 et les versions ultérieures :

```
Remove-Service -Name "greengrass" -Confirm:$false -Verbose
```

- Pour PowerShell les versions antérieures à 6.0 :

```
Get-Item HKLM:\SYSTEM\CurrentControlSet\Services\greengrass | Remove-Item  
-Force -Verbose
```

4. Redémarrez le périphérique.

2. Supprimez le dossier racine de l'appareil. Remplacez */greengrass/v2* ou *C:\greengrass\v2* par le chemin d'accès au dossier racine.

Linux

```
sudo rm -rf /greengrass/v2
```

Windows (Command Prompt)

```
rmdir /s /q C:\greengrass\v2
```

Windows (PowerShell)

```
cmd.exe /c "rmdir /s /q C:\greengrass\v2"
```

3. Supprimez le périphérique principal du AWS IoT Greengrass service. Cette étape supprime les informations d'état du périphérique principal du AWS Cloud. Assurez-vous de terminer cette étape si vous prévoyez de réinstaller le logiciel AWS IoT Greengrass Core sur un périphérique principal portant le même nom.
 - Pour supprimer un périphérique principal de la AWS IoT Greengrass console, procédez comme suit :
 - a. Accédez à la [console AWS IoT Greengrass](#).
 - b. Choisissez les appareils Core.
 - c. Choisissez le périphérique principal à supprimer.
 - d. Sélectionnez Delete (Supprimer).
 - e. Dans le mode de confirmation, choisissez Supprimer.
 - Pour supprimer un périphérique principal doté du AWS Command Line Interface, utilisez l'[DeleteCoreDevice](#) opération. Exécutez la commande suivante et *MyGreengrassCore* remplacez-la par le nom du périphérique principal.

```
aws greengrassv2 delete-core-device --core-device-thing-name MyGreengrassCore
```

AWS IoT Greengrass V2 tutoriels

AWS IoT Greengrass est un service qui vous permet d'exécuter des AWS Lambda fonctions, des modèles d'apprentissage automatique et d'autres codes sur des appareils périphériques. Cela vous permet de traiter les données localement, de réduire la latence et les coûts de bande passante tout en maintenant une communication sécurisée avec le cloud.

Vous pouvez suivre les didacticiels suivants pour en savoir plus sur la AWS IoT Greengrass V2 et ses fonctionnalités.

Rubriques

- [Didacticiels vidéo](#)
- [Tutoriel : Développement d'un composant Greengrass qui reporte les mises à jour des composants](#)
- [Tutoriel : Interagissez avec des appareils IoT locaux via MQTT](#)
- [Tutoriel : Sécurisez Greengrass Nucleus avec le module Trusted Platform \(TPM\)](#)
- [Tutoriel : Sécurisez AWS IoT Greengrass Nucleus Lite avec le module Trusted Platform \(TPM\)](#)
- [Tutoriel : Démarrez avec SageMaker AI Edge Manager](#)
- [Tutoriel : Effectuer une inférence de classification d'images d'échantillons à l'aide TensorFlow de Lite](#)
- [Tutoriel : effectuer une inférence de classification d'images d'échantillons sur des images provenant d'un appareil photo à l'aide TensorFlow de Lite](#)

Didacticiels vidéo

Vidéos

La liste suivante inclut des didacticiels vidéo pour AWS IoT Greengrass V2.

- [AWS IoT Greengrass Tutoriel Nucleus Lite : Mise en route](#)

10 avril 2025 — Dans ce didacticiel, vous découvrirez Greengrass Nucleus Lite et vous expliquerez quand le sélectionner pour vos AWS IoT appareils. Vous apprendrez à installer Nucleus Lite et à y déployer vos composants.

- [AWS IoT Greengrass V2 tutoriel : Mise en route](#)

16 décembre 2021 — Commencez avec AWS IoT Greengrass V2. Dans ce didacticiel, vous allez apprendre à installer Greengrass et à effectuer votre premier déploiement.

- [AWS IoT Greengrass V2 développement de composants avec console de débogage locale](#)

21 août 2023 — Découvrez comment utiliser la console de débogage locale Greengrass V2 pour développer et déboguer rapidement vos composants.

- [Connectez les appareils clients MQTT à AWS IoT Greengrass V2](#)

2 août 2023 — Dans ce didacticiel, vous apprendrez comment configurer un périphérique principal pour interagir avec les AWS IoT appareils locaux qui se connectent au périphérique principal via MQTT.

- [AWS IoT Greengrass V2 tutoriel : Utilisation des applications MQTT](#)

16 décembre 2021 — Créez, testez et déployez un composant AWS IoT Greengrass V2 natif pour publier et souscrire à MQTT via AWS IoT Core.

Tutoriel : Développement d'un composant Greengrass qui reporte les mises à jour des composants

Vous pouvez suivre ce didacticiel pour développer un composant qui reporte les mises à jour over-the-air de déploiement. Lorsque vous déployez des mises à jour sur vos appareils, vous souhaiterez peut-être retarder les mises à jour en fonction de certaines conditions, telles que les suivantes :

- Le niveau de batterie de l'appareil est faible.
- L'appareil exécute un processus ou une tâche qui ne peut pas être interrompu.
- L'appareil dispose d'une connexion Internet limitée ou coûteuse.

Note

Un composant est un module logiciel qui s'exécute sur les appareils AWS IoT Greengrass principaux. Les composants vous permettent de créer et de gérer des applications complexes sous forme de composants distincts que vous pouvez réutiliser d'un appareil principal de Greengrass à un autre.

Dans ce didacticiel, vous allez effectuer les opérations suivantes :

1. Installez le Greengrass Development Kit CLI (GDK CLI) sur votre ordinateur de développement. La CLI GDK fournit des fonctionnalités qui vous aident à développer des composants Greengrass personnalisés.
2. Développez un composant Hello World qui reporte les mises à jour des composants lorsque le niveau de batterie de l'appareil principal est inférieur à un seuil. Ce composant s'abonne aux notifications de mise à jour à l'aide de l'opération [SubscribeToComponentUpdates](#)IPC. Lorsqu'il reçoit la notification, il vérifie si le niveau de la batterie est inférieur à un seuil personnalisable. Si le niveau de la batterie est inférieur au seuil, la mise à jour est différée de 30 secondes en utilisant le fonctionnement [DeferComponentUpdate](#)IPC. Vous développez ce composant sur votre ordinateur de développement à l'aide de la CLI GDK.

Note

Ce composant lit le niveau de batterie à partir d'un fichier que vous créez sur l'appareil principal pour imiter une vraie batterie. Vous pouvez donc suivre ce didacticiel sur un appareil principal sans batterie.

3. Publiez ce composant sur le AWS IoT Greengrass service.
4. Déployez ce composant depuis AWS Cloud le périphérique principal de Greengrass pour le tester. Ensuite, vous modifiez le niveau de batterie virtuel sur le périphérique principal et vous créez des déploiements supplémentaires pour voir comment le périphérique principal reporte les mises à jour lorsque le niveau de batterie est faible.

Vous pouvez vous attendre à consacrer 20 à 30 minutes à ce didacticiel.

Prérequis

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

- Un Compte AWS. Si vous n'en avez pas, veuillez consulter [Configurez un Compte AWS](#).
- Un utilisateur Gestion des identités et des accès AWS (IAM) doté d'autorisations d'administrateur.
- Un appareil Greengrass Core doté d'une connexion Internet. Pour plus d'informations sur la configuration d'un périphérique principal, consultez [Configuration des appareils AWS IoT Greengrass principaux](#).

- [Python](#) 3.6 ou version ultérieure installé pour tous les utilisateurs sur le périphérique principal et ajouté à la variable d'PATHenvironnement. Sous Windows, le lanceur Python pour Windows doit également être installé pour tous les utilisateurs.

Important

Sous Windows, Python ne s'installe pas pour tous les utilisateurs par défaut. Lorsque vous installez Python, vous devez personnaliser l'installation pour la configurer afin que le logiciel AWS IoT Greengrass Core exécute des scripts Python. Par exemple, si vous utilisez le programme d'installation graphique Python, procédez comme suit :

1. Sélectionnez Installer le lanceur pour tous les utilisateurs (recommandé).
2. Choisissez Customize installation.
3. Choisissez Next.
4. Sélectionnez Install for all users.
5. Sélectionnez Add Python to environment variables.
6. Choisissez Installer.

Pour plus d'informations, consultez la section [Utilisation de Python sous Windows](#) dans la documentation de Python 3.

- Un ordinateur de développement de type Windows, macOS ou Unix doté d'une connexion Internet.
- [Python](#) 3.6 ou version ultérieure installé sur votre ordinateur de développement.
- [Git](#) installé sur votre ordinateur de développement.
- AWS Command Line Interface (AWS CLI) installé et configuré avec des informations d'identification sur votre ordinateur de développement. Pour plus d'informations, consultez les [sections Installation, mise à jour et désinstallation du AWS CLI](#) et [Configuration du AWS CLI dans le](#) guide de l'AWS Command Line Interface utilisateur.

Note

Si vous utilisez un Raspberry Pi ou un autre appareil ARM 32 bits, installez la AWS CLI V1. AWS CLI La V2 n'est pas disponible pour les appareils ARM 32 bits. Pour plus d'informations, voir [Installation, mise à jour et désinstallation de la AWS CLI version 1](#).

Étape 1 : Installation de la CLI du kit de développement Greengrass

Le [Greengrass Development Kit CLI \(GDK CLI\)](#) fournit des fonctionnalités qui vous aident à développer des composants Greengrass personnalisés. Vous pouvez utiliser la CLI GDK pour créer, créer et publier des composants personnalisés.

Si vous n'avez pas installé la CLI GDK sur votre ordinateur de développement, procédez comme suit pour l'installer.

Pour installer la dernière version de la CLI GDK

1. Sur votre ordinateur de développement, exécutez la commande suivante pour installer la dernière version de la CLI GDK à partir de son [GitHub référentiel](#).

```
python3 -m pip install -U git+https://github.com/aws-greengrass/aws-greengrass-gdk-cli.git@v1.6.2
```

2. Exécutez la commande suivante pour vérifier que la CLI GDK a été correctement installée.

```
gdk --help
```

Si la gdk commande est introuvable, ajoutez son dossier dans PATH.

- Sur les appareils Linux, ajoutez `/home/MyUser/.local/bin` à PATH et remplacez-le par `MyUser` le nom de votre utilisateur.
- Sur les appareils Windows, ajoutez `PythonPath\Scripts` à PATH et remplacez-le par `PythonPath` le chemin d'accès au dossier Python de votre appareil.

Étape 2 : développer un composant qui reporte les mises à jour

Dans cette section, vous allez développer un composant Hello World en Python qui reporte les mises à jour des composants lorsque le niveau de batterie de l'appareil principal est inférieur à un seuil que vous configurez lorsque vous déployez le composant. Dans ce composant, vous utilisez l'[interface de communication interprocessus \(IPC\)](#) de la Kit SDK des appareils AWS IoT version v2 pour Python. Vous utilisez l'opération [SubscribeToComponentUpdates](#)IPC pour recevoir des notifications lorsque le périphérique principal reçoit un déploiement. Ensuite, vous utilisez l'opération [DeferComponentUpdate](#)IPC pour différer ou accuser réception de la mise à jour en fonction du niveau de batterie de l'appareil.

Pour développer un composant Hello World qui reporte les mises à jour

1. Sur votre ordinateur de développement, créez un dossier pour le code source du composant.

```
mkdir com.example.BatteryAwareHelloWorld
cd com.example.BatteryAwareHelloWorld
```

2. Utilisez un éditeur de texte pour créer un fichier nommé `gdk-config.json`. La CLI GDK lit le [fichier de configuration de la CLI GDK](#), nommé `gdk-config.json`, pour créer et publier des composants. Ce fichier de configuration se trouve à la racine du dossier du composant.

Par exemple, sur un système basé sur Linux, vous pouvez exécuter la commande suivante pour utiliser GNU nano pour créer le fichier.

```
nano gdk-config.json
```

Copiez le code JSON suivant dans le fichier.

- Remplacez *Amazon* par votre nom.
- *us-west-2* Remplacez-le par celui Région AWS où fonctionne votre appareil principal. La CLI GDK y publie le composant. Région AWS
- *greengrass-component-artifacts* Remplacez-le par le préfixe du compartiment S3 à utiliser. Lorsque vous utilisez la CLI GDK pour publier le composant, la CLI GDK télécharge les artefacts du composant dans le compartiment S3 dont le nom est formé à partir de cette valeur, du Région AWS, et de votre Compte AWS identifiant au format suivant : *bucketPrefix-region-accountId*

Par exemple, si vous spécifiez **greengrass-component-artifacts** et **us-west-2** que votre Compte AWS identifiant est **123456789012**, la CLI GDK utilise le compartiment S3 nommé **greengrass-component-artifacts-us-west-2-123456789012**.

```
{
  "component": {
    "com.example.BatteryAwareHelloWorld": {
      "author": "Amazon",
      "version": "NEXT_PATCH",
      "build": {
        "build_system" : "zip"
      }
    },
  },
}
```

```
    "publish": {
        "region": "us-west-2",
        "bucket": "greengrass-component-artifacts"
    }
},
"gdk_version": "1.0.0"
}
```

Le fichier de configuration spécifie les éléments suivants :

- Version à utiliser lorsque la CLI GDK publie le composant Greengrass sur AWS IoT Greengrass le service cloud. NEXT_PATCH indique de choisir la prochaine version du correctif après la dernière version disponible dans le service AWS IoT Greengrass cloud. Si le composant n'a pas encore de version dans le service AWS IoT Greengrass cloud, la CLI GDK l'utilise 1.0.0.
 - Le système de génération du composant. Lorsque vous utilisez le système de zip génération, la CLI GDK empaquète la source du composant dans un fichier ZIP qui devient l'artefact unique du composant.
 - L' Région AWS endroit où la CLI GDK publie le composant Greengrass.
 - Le préfixe du compartiment S3 dans lequel la CLI GDK télécharge les artefacts du composant.
3. Utilisez un éditeur de texte pour créer le code source du composant dans un fichier nommé `main.py`.

Par exemple, sur un système basé sur Linux, vous pouvez exécuter la commande suivante pour utiliser GNU nano pour créer le fichier.

```
nano main.py
```

Copiez le code Python suivant dans le fichier.

```
import json
import os
import sys
import time
import traceback

from pathlib import Path
```

```
from awsiot.greengrasscoreipc.clientv2 import GreengrassCoreIPCClientV2

HELLO_WORLD_PRINT_INTERVAL = 15 # Seconds
DEFER_COMPONENT_UPDATE_INTERVAL = 30 * 1000 # Milliseconds

class BatteryAwareHelloWorldPrinter():
    def __init__(self, ipc_client: GreengrassCoreIPCClientV2, battery_file_path:
Path, battery_threshold: float):
        self.battery_file_path = battery_file_path
        self.battery_threshold = battery_threshold
        self.ipc_client = ipc_client
        self.subscription_operation = None

    def on_component_update_event(self, event):
        try:
            if event.pre_update_event is not None:
                if self.is_battery_below_threshold():
                    self.defer_update(event.pre_update_event.deployment_id)
                    print('Deferred update for deployment %s' %
                        event.pre_update_event.deployment_id)
                else:
                    self.acknowledge_update(
                        event.pre_update_event.deployment_id)
                    print('Acknowledged update for deployment %s' %
                        event.pre_update_event.deployment_id)
            elif event.post_update_event is not None:
                print('Applied update for deployment')
        except:
            traceback.print_exc()

    def subscribe_to_component_updates(self):
        if self.subscription_operation == None:
            # SubscribeToComponentUpdates returns a tuple with the response and the
            operation.
            _, self.subscription_operation =
self.ipc_client.subscribe_to_component_updates(
                on_stream_event=self.on_component_update_event)

    def close_subscription(self):
        if self.subscription_operation is not None:
            self.subscription_operation.close()
            self.subscription_operation = None
```

```
def defer_update(self, deployment_id):
    self.ipc_client.defer_component_update(
        deployment_id=deployment_id,
        recheck_after_ms=DEFER_COMPONENT_UPDATE_INTERVAL)

def acknowledge_update(self, deployment_id):
    # Specify recheck_after_ms=0 to acknowledge a component update.
    self.ipc_client.defer_component_update(
        deployment_id=deployment_id, recheck_after_ms=0)

def is_battery_below_threshold(self):
    return self.get_battery_level() < self.battery_threshold

def get_battery_level(self):
    # Read the battery level from the virtual battery level file.
    with self.battery_file_path.open('r') as f:
        data = json.load(f)
        return float(data['battery_level'])

def print_message(self):
    message = 'Hello, World!'
    if self.is_battery_below_threshold():
        message += ' Battery level (%d) is below threshold (%d), so the
component will defer updates' % (
            self.get_battery_level(), self.battery_threshold)
    else:
        message += ' Battery level (%d) is above threshold (%d), so the
component will acknowledge updates' % (
            self.get_battery_level(), self.battery_threshold)
    print(message)

def main():
    # Read the battery threshold and virtual battery file path from command-line
    args.
    args = sys.argv[1:]
    battery_threshold = float(args[0])
    battery_file_path = Path(args[1])
    print('Reading battery level from %s and deferring updates when below %d' % (
        str(battery_file_path), battery_threshold))

    try:
        # Create an IPC client and a Hello World printer that defers component
        updates.
```

```
ipc_client = GreengrassCoreIPCClientV2()
hello_world_printer = BatteryAwareHelloWorldPrinter(
    ipc_client, battery_file_path, battery_threshold)
hello_world_printer.subscribe_to_component_updates()
try:
    # Keep the main thread alive, or the process will exit.
    while True:
        hello_world_printer.print_message()
        time.sleep(HELLO_WORLD_PRINT_INTERVAL)
except InterruptedError:
    print('Subscription interrupted')
    hello_world_printer.close_subscription()
except Exception:
    print('Exception occurred', file=sys.stderr)
    traceback.print_exc()
    exit(1)

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

Cette application Python effectue les opérations suivantes :

- Lit le niveau de batterie de l'appareil principal à partir d'un fichier de niveau de batterie virtuel que vous créerez ultérieurement sur l'appareil principal. Ce fichier de niveau de batterie virtuel imite une vraie batterie. Vous pouvez donc suivre ce didacticiel sur les principaux appareils dépourvus de batterie.
- Lit les arguments de ligne de commande pour le seuil de batterie et le chemin d'accès au fichier de niveau de batterie virtuel. La recette du composant définit ces arguments de ligne de commande en fonction des paramètres de configuration, afin que vous puissiez personnaliser ces valeurs lorsque vous déployez le composant.
- Utilise le client IPC V2 dans le [Kit SDK des appareils AWS IoT V2 pour que Python](#) communique avec le logiciel AWS IoT Greengrass Core. Par rapport au client IPC d'origine, le client IPC V2 réduit la quantité de code que vous devez écrire pour utiliser IPC dans des composants personnalisés.
- S'abonne pour mettre à jour les notifications à l'aide de l'[SubscribeToComponentUpdates](#) opération IPC. Le logiciel AWS IoT Greengrass Core envoie des notifications avant et après chaque déploiement. Le composant appelle la fonction suivante chaque fois qu'il reçoit une notification. Si la notification concerne un déploiement

à venir, le composant vérifie si le niveau de la batterie est inférieur à un seuil. Si le niveau de la batterie est inférieur au seuil, le composant reporte la mise à jour de 30 secondes en utilisant le fonctionnement [DeferComponentUpdate](#)IPC. Dans le cas contraire, si le niveau de la batterie n'est pas inférieur au seuil, le composant accuse réception de la mise à jour afin que celle-ci puisse se poursuivre.

```
def on_component_update_event(self, event):
    try:
        if event.pre_update_event is not None:
            if self.is_battery_below_threshold():
                self.defer_update(event.pre_update_event.deployment_id)
                print('Deferred update for deployment %s' %
                      event.pre_update_event.deployment_id)
            else:
                self.acknowledge_update(
                    event.pre_update_event.deployment_id)
                print('Acknowledged update for deployment %s' %
                      event.pre_update_event.deployment_id)
        elif event.post_update_event is not None:
            print('Applied update for deployment')
    except:
        traceback.print_exc()
```

Note

Le logiciel AWS IoT Greengrass Core n'envoie pas de notifications de mise à jour pour les déploiements locaux. Vous déployez donc ce composant à l'aide du service AWS IoT Greengrass cloud pour le tester.

4. Utilisez un éditeur de texte pour créer la recette du composant dans un fichier nommé `recipe.json` ou `recipe.yaml`. La recette du composant définit les métadonnées du composant, les paramètres de configuration par défaut et les scripts de cycle de vie spécifiques à la plate-forme.

JSON

Par exemple, sur un système basé sur Linux, vous pouvez exécuter la commande suivante pour utiliser GNU nano pour créer le fichier.

```
nano recipe.json
```

Copiez le code JSON suivant dans le fichier.

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "COMPONENT_NAME",
  "ComponentVersion": "COMPONENT_VERSION",
  "ComponentDescription": "This Hello World component defers updates when the
battery level is below a threshold.",
  "ComponentPublisher": "COMPONENT_AUTHOR",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "BatteryThreshold": 50,
      "LinuxBatteryFilePath": "/home/ggc_user/virtual_battery.json",
      "WindowsBatteryFilePath": "C:\\Users\\ggc_user\\virtual_battery.json"
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "install": "python3 -m pip install --user awsiotsdk --upgrade",
        "Run": "python3 -u {artifacts:decompressedPath}/
com.example.BatteryAwareHelloWorld/main.py \"{configuration:/BatteryThreshold}\"
\"{configuration:/LinuxBatteryFilePath}\""
      },
      "Artifacts": [
        {
          "Uri": "s3://BUCKET_NAME/COMPONENT_NAME/COMPONENT_VERSION/
com.example.BatteryAwareHelloWorld.zip",
          "Unarchive": "ZIP"
        }
      ]
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
```

```

      "install": "py -3 -m pip install --user awsiotsdk --upgrade",
      "Run": "py -3 -u {artifacts:decompressedPath}/
com.example.BatteryAwareHelloWorld/main.py \"{configuration:/BatteryThreshold}\"
\"{configuration:/WindowsBatteryFilePath}\""
    },
    "Artifacts": [
      {
        "Uri": "s3://BUCKET_NAME/COMPONENT_NAME/COMPONENT_VERSION/
com.example.BatteryAwareHelloWorld.zip",
        "Unarchive": "ZIP"
      }
    ]
  }
]
}

```

YAML

Par exemple, sur un système basé sur Linux, vous pouvez exécuter la commande suivante pour utiliser GNU nano pour créer le fichier.

```
nano recipe.yaml
```

Copiez le code YAML suivant dans le fichier.

```

---
RecipeFormatVersion: "2020-01-25"
ComponentName: "COMPONENT_NAME"
ComponentVersion: "COMPONENT_VERSION"
ComponentDescription: "This Hello World component defers updates when the
  battery level is below a threshold."
ComponentPublisher: "COMPONENT_AUTHOR"
ComponentConfiguration:
  DefaultConfiguration:
    BatteryThreshold: 50
    LinuxBatteryFilePath: "/home/ggc_user/virtual_battery.json"
    WindowsBatteryFilePath: "C:\\Users\\ggc_user\\virtual_battery.json"
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      install: python3 -m pip install --user awsiotsdk --upgrade

```



```

    Run: python3 -u {artifacts:decompressedPath}/
com.example.BatteryAwareHelloWorld/main.py "{configuration:/BatteryThreshold}"
"{configuration:/LinuxBatteryFilePath}"
    Artifacts:
      - Uri: "s3://BUCKET_NAME/COMPONENT_NAME/COMPONENT_VERSION/
com.example.BatteryAwareHelloWorld.zip"
        Unarchive: ZIP
      - Platform:
        os: windows
    Lifecycle:
      install: py -3 -m pip install --user awsiot-sdk --upgrade
      Run: py -3 -u {artifacts:decompressedPath}/
com.example.BatteryAwareHelloWorld/main.py "{configuration:/BatteryThreshold}"
"{configuration:/WindowsBatteryFilePath}"
    Artifacts:
      - Uri: "s3://BUCKET_NAME/COMPONENT_NAME/COMPONENT_VERSION/
com.example.BatteryAwareHelloWorld.zip"
        Unarchive: ZIP

```

Cette recette précise les éléments suivants :

- Paramètres de configuration par défaut pour le seuil de batterie, le chemin du fichier de batterie virtuel sur les appareils principaux Linux et le chemin du fichier de batterie virtuel sur les appareils principaux de Windows.
- Un `install` cycle de vie qui installe la dernière version de la Kit SDK des appareils AWS IoT v2 pour Python.
- Un `run` cycle de vie qui exécute l'application Python dans `main.py`.
- Espaces réservés, tels que `COMPONENT_NAME` et `COMPONENT_VERSION`, où la CLI GDK remplace les informations lorsqu'elle crée la recette du composant.

Pour plus d'informations sur les recettes de composants, consultez [AWS IoT Greengrass référence de recette de composant](#).

Étape 3 : Publier le composant sur le AWS IoT Greengrass service

Dans cette section, vous publiez le composant Hello World sur le service AWS IoT Greengrass cloud. Une fois qu'un composant est disponible dans le service AWS IoT Greengrass cloud, vous pouvez le déployer sur les appareils principaux. Vous utilisez la CLI GDK pour publier le composant

depuis votre ordinateur de développement vers le service AWS IoT Greengrass cloud. La CLI GDK télécharge la recette et les artefacts du composant pour vous.

Pour publier le composant Hello World sur le AWS IoT Greengrass service

1. Exécutez la commande suivante pour créer le composant à l'aide de la CLI GDK. La [commande `component build`](#) crée une recette et des artefacts basés sur le fichier de configuration de la CLI GDK. Au cours de ce processus, la CLI GDK crée un fichier ZIP contenant le code source du composant.

```
gdk component build
```

Vous devriez voir des messages similaires à ceux de l'exemple suivant.

```
[2022-04-28 11:20:16] INFO - Getting project configuration from gdk-config.json
[2022-04-28 11:20:16] INFO - Found component recipe file 'recipe.yaml' in the
project directory.
[2022-04-28 11:20:16] INFO - Building the component
'com.example.BatteryAwareHelloWorld' with the given project configuration.
[2022-04-28 11:20:16] INFO - Using 'zip' build system to build the component.
[2022-04-28 11:20:16] WARNING - This component is identified as using 'zip' build
system. If this is incorrect, please exit and specify custom build command in the
'gdk-config.json'.
[2022-04-28 11:20:16] INFO - Zipping source code files of the component.
[2022-04-28 11:20:16] INFO - Copying over the build artifacts to the greengrass
component artifacts build folder.
[2022-04-28 11:20:16] INFO - Updating artifact URIs in the recipe.
[2022-04-28 11:20:16] INFO - Creating component recipe in 'C:\Users\finthomp
\greengrassv2\com.example.BatteryAwareHelloWorld\greengrass-build\recipes'.
```

2. Exécutez la commande suivante pour publier le composant sur le service AWS IoT Greengrass cloud. La [commande de publication du composant](#) télécharge l'artefact du fichier ZIP du composant dans un compartiment S3. Ensuite, il met à jour l'URI S3 du fichier ZIP dans la recette du composant et télécharge la recette vers le AWS IoT Greengrass service. Au cours de ce processus, la CLI GDK vérifie quelle version du composant Hello World est déjà disponible dans le service AWS IoT Greengrass cloud, afin de pouvoir choisir la version de correctif suivante après cette version. Si le composant n'existe pas encore, la CLI GDK utilise la version 1.0.0.

```
gdk component publish
```

Vous devriez voir des messages similaires à ceux de l'exemple suivant. La sortie indique la version du composant créée par la CLI GDK.

```
[2022-04-28 11:20:29] INFO - Getting project configuration from gdk-config.json
[2022-04-28 11:20:29] INFO - Found component recipe file 'recipe.yaml' in the
project directory.
[2022-04-28 11:20:29] INFO - Found credentials in shared credentials file: ~/.aws/
credentials
[2022-04-28 11:20:30] INFO - No private version of the component
'com.example.BatteryAwareHelloWorld' exist in the account. Using '1.0.0' as the
next version to create.
[2022-04-28 11:20:30] INFO - Publishing the component
'com.example.BatteryAwareHelloWorld' with the given project configuration.
[2022-04-28 11:20:30] INFO - Uploading the component built artifacts to s3 bucket.
[2022-04-28 11:20:30] INFO - Uploading component artifacts to S3
bucket: greengrass-component-artifacts-us-west-2-123456789012. If this is your
first time using this bucket, add the 's3:GetObject' permission to each core
device's token exchange role to allow it to download the component artifacts. For
more information, see https://docs.aws.amazon.com/greengrass/v2/developerguide/
device-service-role.html.
[2022-04-28 11:20:30] INFO - Not creating an artifacts bucket as it already exists.
[2022-04-28 11:20:30] INFO - Updating the component recipe
com.example.BatteryAwareHelloWorld-1.0.0.
[2022-04-28 11:20:31] INFO - Creating a new greengrass component
com.example.BatteryAwareHelloWorld-1.0.0
[2022-04-28 11:20:31] INFO - Created private version '1.0.0' of the component in
the account.'com.example.BatteryAwareHelloWorld'.
```

3. Copiez le nom du compartiment S3 depuis la sortie. Vous utiliserez le nom du compartiment ultérieurement pour autoriser le périphérique principal à télécharger les artefacts des composants à partir de ce compartiment.
4. (Facultatif) Affichez le composant dans la AWS IoT Greengrass console pour vérifier qu'il a été correctement chargé. Procédez comme suit :
 - a. Dans le menu de navigation de la [AWS IoT Greengrass console](#), sélectionnez Composants.
 - b. Sur la page Composants, choisissez l'onglet Mes composants, puis sélectionnez com.example.BatteryAwareHelloWorld.

Sur cette page, vous pouvez consulter la recette du composant ainsi que d'autres informations le concernant.

5. Autorisez le périphérique principal à accéder aux artefacts des composants dans le compartiment S3.

Chaque appareil principal possède un [rôle IAM principal](#) qui lui permet d'interagir avec le AWS cloud AWS IoT et d'envoyer des journaux vers celui-ci. Ce rôle de périphérique n'autorise pas l'accès aux compartiments S3 par défaut. Vous devez donc créer et associer une politique permettant au périphérique principal de récupérer les artefacts des composants du compartiment S3.

Si le rôle de votre appareil autorise déjà l'accès au compartiment S3, vous pouvez ignorer cette étape. Sinon, créez une politique IAM autorisant l'accès et associez-la au rôle, comme suit :

- a. Dans le menu de navigation de [la console IAM](#), choisissez Politiques, puis Create policy.
- b. Sur l'onglet JSON, remplacez le contenu de l'espace réservé par la stratégie suivante.
greengrass-component-artifacts-us-west-2-123456789012 Remplacez-le par le nom du compartiment S3 dans lequel la CLI GDK a téléchargé les artefacts du composant.

Par exemple, si vous l'avez spécifié **greengrass-component-artifacts us-west-2** dans le fichier de configuration de la CLI GDK et que votre Compte AWS identifiant est **123456789012**, la CLI GDK utilise le compartiment S3 nommé. *greengrass-component-artifacts-us-west-2-123456789012*

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": "arn:aws:s3:::greengrass-component-artifacts-us-west-2-123456789012/*"
    }
  ]
}
```

- c. Choisissez Suivant.

- d. Dans la section Détails de la politique, pour Nom, entrez **MyGreengrassV2ComponentArtifactPolicy**.
- e. Choisissez Create Policy (Créer une politique).
- f. Dans le menu de navigation de [la console IAM](#), choisissez Role, puis choisissez le nom du rôle pour le périphérique principal. Vous avez spécifié ce nom de rôle lors de l'installation du logiciel AWS IoT Greengrass Core. Si vous n'avez pas spécifié de nom, le nom par défaut est `GreengrassV2TokenExchangeRole`.
- g. Sous Autorisations, choisissez Ajouter des autorisations, puis choisissez Joindre des politiques.
- h. Sur la page Ajouter des autorisations, cochez la case à côté de la `MyGreengrassV2ComponentArtifactPolicy` politique que vous avez créée, puis choisissez Ajouter des autorisations.

Étape 4 : Déployer et tester le composant sur un appareil principal

Dans cette section, vous déployez le composant sur le périphérique principal pour tester ses fonctionnalités. Sur l'appareil principal, vous créez le fichier de niveau de batterie virtuel pour imiter une batterie réelle. Ensuite, vous créez des déploiements supplémentaires et vous observez les fichiers journaux des composants sur le périphérique principal pour voir le composant différer et accuser réception des mises à jour.

Pour déployer et tester le composant Hello World qui reporte les mises à jour

1. Utilisez un éditeur de texte pour créer un fichier virtuel de niveau de batterie. Ce fichier imite une vraie batterie.
 - Sur les appareils principaux de Linux, créez un fichier nommé `/home/ggc_user/virtual_battery.json`. Exécutez l'éditeur de texte avec `sudo` des autorisations.
 - Sur les appareils principaux de Windows, créez un fichier nommé `C:\Users\ggc_user\virtual_battery.json`. Exécutez l'éditeur de texte en tant qu'administrateur.

Par exemple, sur un système basé sur Linux, vous pouvez exécuter la commande suivante pour utiliser GNU nano pour créer le fichier.

```
sudo nano /home/ggc_user/virtual_battery.json
```

Copiez le code JSON suivant dans le fichier.

```
{
  "battery_level": 50
}
```

2. Déployez le composant Hello World sur l'appareil principal. Procédez comme suit :
 - a. Dans le menu de navigation de la [AWS IoT Greengrass console](#), sélectionnez Composants.
 - b. Sur la page Composants, choisissez l'onglet Mes composants, puis choisissez com.example.BatteryAwareHelloWorld.
 - c. Dans la page com.example.BatteryAwareHelloWorldpage, choisissez Déployer.
 - d. Dans Ajouter au déploiement, choisissez un déploiement existant à réviser ou choisissez de créer un nouveau déploiement, puis choisissez Suivant.
 - e. Si vous avez choisi de créer un nouveau déploiement, choisissez le périphérique principal ou le groupe d'objets cible pour le déploiement. Sur la page Spécifier la cible, sous Cible de déploiement, choisissez un périphérique principal ou un groupe d'objets, puis cliquez sur Suivant.
 - f. Sur la page Sélectionner les composants, vérifiez que com.example.BatteryAwareHelloWorldle composant est sélectionné, choisissez Next.
 - g. Sur la page Configurer les composants, sélectionnez com.example.BatteryAwareHelloWorld, puis procédez comme suit :
 - i. Choisissez Configure component (Configurer un composant).
 - ii. Dans la configuration com.example.BatteryAwareHelloWorldmodal, sous Mise à jour de la configuration, dans Configuration à fusionner, entrez la mise à jour de configuration suivante.

```
{
  "BatteryThreshold": 70
}
```

- iii. Choisissez Confirmer pour fermer le modal, puis cliquez sur Suivant.
 - h. Sur la page Confirmer les paramètres avancés, dans la section Politiques de déploiement, sous Politique de mise à jour des composants, vérifiez que l'option Notifier les composants est sélectionnée. L'option Notifier les composants est sélectionnée par défaut lorsque vous créez un nouveau déploiement.

- i. Sur la page Review (Révision), choisissez Deploy (Déployer).

Le déploiement peut prendre jusqu'à une minute.

3. Le logiciel AWS IoT Greengrass Core enregistre les données standard des processus des composants dans les fichiers journaux du logs dossier. Exécutez la commande suivante pour vérifier que le composant Hello World s'exécute et imprime des messages d'état.

Linux or Unix

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

Windows Command Prompt (CMD)

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

PowerShell

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

Vous devriez voir des messages similaires à ceux de l'exemple suivant.

```
Hello, World! Battery level (50) is below threshold (70), so the component will defer updates.
```

Note

Si le fichier n'existe pas, le déploiement n'est peut-être pas encore terminé. Si le fichier n'existe pas dans les 30 secondes, le déploiement a probablement échoué. Cela peut se produire si le périphérique principal n'est pas autorisé à télécharger les artefacts du composant depuis le compartiment S3, par exemple. Exécutez la commande suivante pour afficher le fichier journal du logiciel AWS IoT Greengrass Core. Ce fichier inclut les journaux du service de déploiement de l'appareil principal Greengrass.

Linux or Unix

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

Windows Command Prompt (CMD)

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

La type commande écrit le contenu du fichier sur le terminal. Exécutez cette commande plusieurs fois pour observer les modifications apportées au fichier.

PowerShell

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

4. Créez un nouveau déploiement sur le périphérique principal pour vérifier que le composant reporte la mise à jour. Procédez comme suit :
 - a. Dans le menu de navigation de la [AWS IoT Greengrass console](#), sélectionnez Déploiements.
 - b. Choisissez le déploiement que vous avez créé ou révisé précédemment.
 - c. Sur la page de déploiement, choisissez Revise.
 - d. Dans le mode de déploiement Revise, choisissez Revise le déploiement.
 - e. Choisissez Next à chaque étape, puis choisissez Deploy.
5. Exécutez la commande suivante pour consulter à nouveau les journaux du composant et vérifier que la mise à jour est différée.

Linux or Unix

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

Windows Command Prompt (CMD)

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

PowerShell

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


Vous devriez voir des messages similaires à ceux de l'exemple suivant. Le composant reporte la mise à jour de 30 secondes, de sorte qu'il imprime ce message à plusieurs reprises.

```
Deferred update for deployment 50722a95-a05f-4e2a-9414-da80103269aa.
```

6. Utilisez un éditeur de texte pour modifier le fichier de niveau de batterie virtuel et remplacer le niveau de batterie par une valeur supérieure au seuil, afin que le déploiement puisse se poursuivre.
 - Sur les appareils principaux de Linux, modifiez le fichier nommé `/home/ggc_user/virtual_battery.json`. Exécutez l'éditeur de texte avec `sudo` des autorisations.
 - Sur les appareils principaux de Windows, modifiez le fichier nommé `C:\Users\ggc_user\virtual_battery.json`. Exécutez l'éditeur de texte en tant qu'administrateur.

Par exemple, sur un système basé sur Linux, vous pouvez exécuter la commande suivante pour utiliser GNU nano pour créer le fichier.

```
sudo nano /home/ggc_user/virtual_battery.json
```

Réglez le niveau de la batterie sur 80.

```
{  
  "battery_level": 80  
}
```

7. Exécutez la commande suivante pour consulter à nouveau les journaux du composant et vérifier qu'il accuse réception de la mise à jour.

Linux or Unix

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

Windows Command Prompt (CMD)

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

PowerShell

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

Vous devriez voir des messages similaires aux exemples suivants.

```
Hello, World! Battery level (80) is above threshold (70), so the component will  
acknowledge updates.  
Acknowledged update for deployment f9499eb2-4a40-40a7-86c1-c89887d859f1.
```

Vous avez terminé ce didacticiel. Le composant Hello World reporte ou accuse réception des mises à jour en fonction du niveau de batterie de l'appareil principal. Pour plus d'informations sur les sujets abordés dans ce didacticiel, consultez les rubriques suivantes :

- [Développer des AWS IoT Greengrass composants](#)
- [Déployer AWS IoT Greengrass des composants sur des appareils](#)
- [Utilisez le Kit SDK des appareils AWS IoT pour communiquer avec le noyau de Greengrass, les autres composants et AWS IoT Core](#)
- [AWS IoT Greengrass Interface de ligne de commande du kit de développement](#)

Tutoriel : Interagissez avec des appareils IoT locaux via MQTT

Vous pouvez suivre ce didacticiel pour configurer un appareil principal afin qu'il interagisse avec des appareils IoT locaux, appelés appareils clients, qui se connectent au périphérique principal via MQTT. Dans ce didacticiel, vous allez configurer AWS IoT des éléments pour utiliser la découverte du cloud pour se connecter au périphérique principal en tant qu'appareils clients. Lorsque vous configurez la découverte du cloud, un appareil client peut envoyer une demande au service AWS IoT Greengrass cloud pour découvrir les appareils principaux. La réponse AWS IoT Greengrass inclut des informations de connectivité et des certificats pour les appareils principaux que vous configurez pour que le périphérique client découvre. Le périphérique client peut ensuite utiliser ces informations pour se connecter à un périphérique principal disponible où il peut communiquer via MQTT.

Dans ce didacticiel, vous allez effectuer les opérations suivantes :

1. Vérifiez et mettez à jour les autorisations de l'appareil principal, si nécessaire.

2. Associez les appareils clients au périphérique principal afin qu'ils puissent découvrir le périphérique principal à l'aide de la découverte du cloud.
3. Déployez les composants Greengrass sur l'appareil principal pour permettre le support des appareils clients.
4. Connectez les appareils clients à l'appareil principal et testez la communication avec le service AWS IoT Core cloud.
5. Développez un composant Greengrass personnalisé qui communique avec les appareils clients.
6. Développez un composant personnalisé qui interagit avec les ombres des [AWS IoT appareils](#) clients.

Ce didacticiel utilise un seul périphérique central et un seul appareil client. Vous pouvez également suivre le didacticiel pour connecter et tester plusieurs appareils clients.

Vous pouvez vous attendre à consacrer 30 à 60 minutes à ce didacticiel.

Prérequis

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

- Un Compte AWS. Si vous n'en avez pas, veuillez consulter [Configurez un Compte AWS](#).
- Un utilisateur Gestion des identités et des accès AWS (IAM) doté d'autorisations d'administrateur.
- Un appareil Greengrass Core. Pour plus d'informations sur la configuration d'un périphérique principal, consultez [Configuration des appareils AWS IoT Greengrass principaux](#).
- Le périphérique principal doit exécuter Greengrass nucleus v2.6.0 ou version ultérieure. Cette version inclut la prise en charge des caractères génériques dans les publish/subscribe communications locales et la prise en charge des ombres des appareils clients.

Note

La prise en charge des appareils clients nécessite Greengrass nucleus v2.2.0 ou version ultérieure. Cependant, ce didacticiel explore de nouvelles fonctionnalités, telles que la prise en charge des caractères génériques MQTT en local publish/subscribe et la prise en charge des ombres des appareils clients. Ces fonctionnalités nécessitent Greengrass nucleus v2.6.0 ou version ultérieure.

- Le périphérique principal doit se trouver sur le même réseau que les appareils clients pour se connecter.

- (Facultatif) Pour terminer les modules dans lesquels vous développez des composants Greengrass personnalisés, le périphérique principal doit exécuter la CLI Greengrass. Pour de plus amples informations, veuillez consulter [Installation de la CLI Greengrass](#).
- Un AWS IoT objet à connecter en tant qu'appareil client dans ce didacticiel. Pour plus d'informations, consultez la section [Création de AWS IoT ressources](#) dans le guide du AWS IoT Core développeur.
- La AWS IoT politique de l'appareil client doit autoriser `greengrass:Discover` autorisation. Pour de plus amples informations, veuillez consulter [AWS IoT Politique minimale pour les appareils clients](#).
- L'appareil client doit se trouver sur le même réseau que le périphérique principal.
- L'appareil client doit exécuter [Python 3](#).
- L'appareil client doit exécuter [Git](#).

Étape 1 : révision et mise à jour de la AWS IoT politique de base relative aux appareils

Pour prendre en charge les appareils clients, la AWS IoT politique d'un appareil principal doit autoriser les autorisations suivantes :

- `greengrass:PutCertificateAuthorities`
- `greengrass:VerifyClientDeviceIdentity`
- `greengrass:VerifyClientDeviceIoTCertificateAssociation`
- `greengrass:GetConnectivityInfo`
- `greengrass:UpdateConnectivityInfo`— (Facultatif) Cette autorisation est requise pour utiliser le [composant de détection IP](#), qui transmet les informations de connectivité réseau de l'appareil principal au service AWS IoT Greengrass cloud.

Pour plus d'informations sur ces autorisations et AWS IoT politiques pour les appareils principaux, consultez [AWS IoT politiques relatives aux opérations du plan de données](#) et [AWS IoT Politique minimale de prise en charge des appareils clients](#).

Dans cette section, vous passez en revue les AWS IoT politiques de votre appareil principal et ajoutez les autorisations requises manquantes. Si vous avez utilisé le [programme d'installation du logiciel AWS IoT Greengrass Core pour provisionner des ressources](#), votre appareil principal

dispose d'une AWS IoT politique qui autorise l'accès à toutes les AWS IoT Greengrass actions (`greengrass:*`). Dans ce cas, vous devez mettre à jour la AWS IoT politique uniquement si vous prévoyez de configurer le composant Shadow Manager avec lequel synchroniser les ombres du périphérique AWS IoT Core. Sinon, vous pouvez ignorer cette section.

Pour consulter et mettre à jour la AWS IoT politique d'un appareil principal

1. Dans le menu de navigation de la [AWS IoT Greengrass console](#), choisissez Core devices.
2. Sur la page des appareils principaux, choisissez le périphérique principal à mettre à jour.
3. Sur la page de détails de l'appareil principal, choisissez le lien vers l'objet de l'appareil principal. Ce lien ouvre la page des détails de l'objet dans la AWS IoT console.
4. Sur la page des détails de l'objet, sélectionnez Certificats.
5. Dans l'onglet Certificats, choisissez le certificat actif de l'objet.
6. Sur la page des détails du certificat, sélectionnez Politiques.
7. Dans l'onglet Politiques, choisissez la AWS IoT politique à revoir et à mettre à jour. Vous pouvez ajouter les autorisations requises à n'importe quelle politique associée au certificat actif de l'appareil principal.

 Note

Si vous avez utilisé le [programme d'installation du logiciel AWS IoT Greengrass Core pour provisionner des ressources](#), vous avez deux AWS IoT règles. Nous vous recommandons de choisir la politique nommée GreengrassV2IoTThingPolicy, si elle existe. Les appareils principaux que vous créez avec le programme d'installation rapide utilisent ce nom de politique par défaut. Si vous ajoutez des autorisations à cette politique, vous les accordez également aux autres appareils principaux qui utilisent cette politique.

8. Dans l'aperçu des politiques, choisissez Modifier la version active.
9. Passez en revue la politique relative aux autorisations requises et ajoutez les autorisations requises manquantes.
10. 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.
11. Choisissez Enregistrer en tant que nouvelle version.

Étape 2 : activer la prise en charge des appareils clients

Pour qu'un appareil client utilise la découverte du cloud pour se connecter à un appareil principal, vous devez associer les appareils. Lorsque vous associez un appareil client à un appareil principal, vous permettez à ce dernier de récupérer les adresses IP et les certificats du périphérique principal à utiliser pour se connecter.

Pour permettre aux appareils clients de se connecter en toute sécurité à un appareil principal et de communiquer avec les composants de Greengrass AWS IoT Core, vous déployez les composants Greengrass suivants sur le périphérique principal :

- [Authentification de l'appareil client](#) (`aws.greengrass.clientdevices.Auth`)

Déployez le composant d'authentification des appareils clients pour authentifier les appareils clients et autoriser les actions des appareils clients. Ce composant permet à vos AWS IoT objets de se connecter à un appareil principal.

Ce composant nécessite une certaine configuration pour pouvoir être utilisé. Vous devez spécifier les groupes de périphériques clients et les opérations que chaque groupe est autorisé à effectuer, telles que la connexion et la communication via MQTT. Pour plus d'informations, consultez la section [Configuration du composant d'authentification de l'appareil client](#).

- [Courtier MQTT 3.1.1 \(Moquette\)](#) (`aws.greengrass.clientdevices.mqtt.Moquette`)

Déployez le composant de courtier MQTT Moquette pour exécuter un courtier MQTT léger. Le broker Moquette MQTT est conforme à MQTT 3.1.1 et inclut un support local pour QoS 0, QoS 1, QoS 2, les messages conservés, les messages de dernière volonté et les abonnements persistants.

Vous n'êtes pas obligé de configurer ce composant pour l'utiliser. Cependant, vous pouvez configurer le port sur lequel ce composant fait fonctionner le broker MQTT. Par défaut, il utilise le port 8883.

- [Pont MQTT](#) (`aws.greengrass.clientdevices.mqtt.Bridge`)

(Facultatif) Déployez le composant pont MQTT pour relayer les messages entre les appareils clients (MQTT local), publication/abonnement locaux et MQTT. AWS IoT Core Configurez ce composant pour synchroniser les appareils clients avec les appareils clients AWS IoT Core et interagir avec eux à partir des composants Greengrass.

Ce composant nécessite une configuration pour être utilisé. Vous devez spécifier les mappages de rubriques dans lesquels ce composant relaie les messages. Pour plus d'informations, consultez la section [Configuration des composants du pont MQTT](#).

- [Détecteur IP](#) (`aws.greengrass.clientdevices.IPDetector`)

(Facultatif) Déployez le composant de détection IP pour signaler automatiquement les points de terminaison du broker MQTT du périphérique principal au service AWS IoT Greengrass cloud. Vous ne pouvez pas utiliser ce composant si votre configuration réseau est complexe, par exemple si un routeur transmet le port du broker MQTT au périphérique principal.

Vous n'êtes pas obligé de configurer ce composant pour l'utiliser.

Dans cette section, vous allez utiliser la AWS IoT Greengrass console pour associer des appareils clients et déployer des composants de périphériques clients sur un périphérique principal.

Pour activer la prise en charge des appareils clients

1. Accédez à la [console AWS IoT Greengrass](#).
2. Dans le menu de navigation de gauche, choisissez Core devices.
3. Sur la page des appareils principaux, choisissez le périphérique principal sur lequel vous souhaitez activer la prise en charge des appareils clients.
4. Sur la page des détails de l'appareil principal, choisissez l'onglet Appareils clients.
5. Dans l'onglet Appareils clients, choisissez Configurer la découverte du cloud.

La page Configurer la découverte du périphérique principal s'ouvre. Sur cette page, vous pouvez associer des appareils clients à un appareil principal et déployer des composants de périphériques clients. Cette page sélectionne l'appareil principal pour vous à l'étape 1 : Sélectionnez les appareils principaux cibles.

Note

Vous pouvez également utiliser cette page pour configurer la découverte des périphériques principaux pour un groupe d'objets. Si vous choisissez cette option, vous pouvez déployer les composants de l'appareil client sur tous les appareils principaux d'un groupe d'objets. Toutefois, si vous choisissez cette option, vous devrez associer manuellement les appareils clients à chaque périphérique principal ultérieurement après

avoir créé le déploiement. Dans ce didacticiel, vous allez configurer un périphérique monocœur.

6. À l'étape 2 : Associer les appareils clients, associez l' AWS IoT objet de l'appareil client au périphérique principal. Cela permet à l'appareil client d'utiliser la découverte du cloud pour récupérer les informations de connectivité et les certificats du périphérique principal. Procédez comme suit :
 - a. Choisissez Associer les appareils clients.
 - b. Dans le mode Associer les appareils clients aux appareils principaux, entrez le nom de l' AWS IoT objet à associer.
 - c. Choisissez Ajouter.
 - d. Choisissez Associer.
7. À l'étape 3 : Configuration et déploiement des composants Greengrass, déployez des composants pour permettre la prise en charge des appareils clients. Si le périphérique principal cible a déjà été déployé, cette page révisé ce déploiement. Dans le cas contraire, cette page crée un nouveau déploiement pour le périphérique principal. Procédez comme suit pour configurer et déployer les composants de l'appareil client :
 - a. Le périphérique principal doit exécuter [Greengrass nucleus](#) v2.6.0 ou version ultérieure pour terminer ce didacticiel. Si le périphérique principal exécute une version antérieure, procédez comme suit :
 - i. Cochez la case pour déployer le aws.greengrass.Nucleuscomposant.
 - ii. Pour le aws.greengrass.Nucleuscomposant, choisissez Modifier la configuration.
 - iii. Pour la version du composant, choisissez la version 2.6.0 ou ultérieure.
 - iv. Choisissez Confirmer.

 Note

Si vous mettez à niveau le noyau Greengrass à partir d'une version mineure antérieure et que le périphérique principal exécute des [composants AWS fournis](#) qui dépendent du noyau, vous devez également mettre à jour les composants AWS fournis vers des versions plus récentes. Vous pouvez configurer la version de ces composants lorsque vous examinerez le déploiement ultérieurement dans

ce didacticiel. Pour de plus amples informations, veuillez consulter [Mettre à jour le logiciel AWS IoT Greengrass principal \(OTA\)](#).

- b. Pour le `aws.greengrass.clientdevices.Authcomposant`, choisissez Modifier la configuration.
- c. Dans le mode Modifier la configuration pour le composant d'authentification du périphérique client, configurez une politique d'autorisation qui permet aux appareils clients de publier et de s'abonner au courtier MQTT sur le périphérique principal. Procédez comme suit :
 - i. Sous Configuration, dans le bloc de code Configuration pour fusionner, entrez la configuration suivante, qui contient une politique d'autorisation de l'appareil client. Chaque politique d'autorisation de groupe d'appareils spécifie un ensemble d'actions et les ressources sur lesquelles un appareil client peut effectuer ces actions.
 - Cette politique permet aux appareils clients dont le nom commence par `se MyClientDevice` de connecter et de communiquer sur tous les sujets MQTT. Remplacez *MyClientDevice** par le nom de l'AWS IoT objet à connecter en tant qu'appareil client. Vous pouvez également spécifier un nom avec le `*` caractère générique correspondant au nom de l'appareil client. Le `*` joker doit se trouver à la fin du nom.

Si vous devez vous connecter à un deuxième appareil client, remplacez-le *MyOtherClientDevice** par le nom de cet appareil client ou par un modèle générique correspondant au nom de cet appareil client. Sinon, vous pouvez supprimer ou conserver cette section de la règle de sélection qui permet aux appareils clients dont les noms correspondent de se `MyOtherClientDevice*` de connecter et de communiquer.
 - Cette politique utilise un OR opérateur pour autoriser également les appareils clients dont le nom commence par `MyOtherClientDevice` à se connecter et à communiquer sur tous les sujets MQTT. Vous pouvez supprimer cette clause dans la règle de sélection ou la modifier pour qu'elle corresponde aux appareils clients à connecter.
 - Cette politique permet aux appareils clients de publier et de s'abonner à tous les sujets MQTT. Pour suivre les meilleures pratiques de sécurité, limitez les `mqtt:subscribe` opérations `mqtt:publish` et à l'ensemble minimal de sujets utilisés par les appareils clients pour communiquer.

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyDeviceGroup": {
        "selectionRule": "thingName: MyClientDevice* OR
thingName: MyOtherClientDevice*",
        "policyName": "MyClientDevicePolicy"
      }
    },
    "policies": {
      "MyClientDevicePolicy": {
        "AllowConnect": {
          "statementDescription": "Allow client devices to connect.",
          "operations": [
            "mqtt:connect"
          ],
          "resources": [
            "*"
          ]
        },
        "AllowPublish": {
          "statementDescription": "Allow client devices to publish to all
topics.",
          "operations": [
            "mqtt:publish"
          ],
          "resources": [
            "*"
          ]
        },
        "AllowSubscribe": {
          "statementDescription": "Allow client devices to subscribe to all
topics.",
          "operations": [
            "mqtt:subscribe"
          ],
          "resources": [
            "*"
          ]
        }
      }
    }
  }
}
```

```
}  
}  
}
```

Pour plus d'informations, consultez la section [Configuration du composant d'authentification de l'appareil client](#).

- ii. Choisissez Confirmer.
- d. Pour le `aws.greengrass.clientdevices.mqtt.Bridgecomposant`, choisissez Modifier la configuration.
- e. Dans le mode Modifier la configuration pour le composant du pont MQTT, configurez un mappage de rubriques qui relaie les messages MQTT des appareils clients vers AWS IoT Core. Procédez comme suit :
 - i. Sous Configuration, dans le bloc de code Configuration pour fusionner, entrez la configuration suivante. Cette configuration indique de relayer les messages MQTT sur le filtre `clients/+/hello/world` thématique des appareils clients vers le service AWS IoT Core cloud. Par exemple, ce filtre de rubrique correspond à la `clients/MyClientDevice1/hello/world` rubrique.

```
{  
  "mqttTopicMapping": {  
    "HelloWorldIotCoreMapping": {  
      "topic": "clients/+/hello/world",  
      "source": "LocalMqtt",  
      "target": "IotCore"  
    }  
  }  
}
```

Pour plus d'informations, consultez la section [Configuration des composants du pont MQTT](#).

- ii. Choisissez Confirmer.
8. Choisissez Vérifier et déployer pour passer en revue le déploiement que cette page crée pour vous.
9. Si vous n'avez pas encore configuré le [rôle de service Greengrass](#) dans cette région, la console ouvre un modal pour configurer le rôle de service pour vous. Le composant d'authentification du périphérique client utilise ce rôle de service pour vérifier l'identité des appareils clients, et

le composant du détecteur d'adresses IP utilise ce rôle de service pour gérer les informations de connectivité du périphérique principal. Choisissez Grant permissions (Accorder des autorisations).

10. Sur la page Révision, choisissez Déployer pour démarrer le déploiement sur le périphérique principal.
11. Pour vérifier que le déploiement est réussi, vérifiez l'état du déploiement et consultez les journaux sur le périphérique principal. Pour vérifier l'état du déploiement sur le périphérique principal, vous pouvez choisir Target dans l'aperçu du déploiement. Pour plus d'informations, consultez les ressources suivantes :

- [Vérifier l'état du déploiement](#)
- [AWS IoT Greengrass Journaux de surveillance](#)

Étape 3 : Connecter les appareils clients

Les appareils clients peuvent utiliser le Kit SDK des appareils AWS IoT pour découvrir, se connecter et communiquer avec un appareil principal. L'appareil client doit être un AWS IoT objet. Pour plus d'informations, consultez la section [Création d'un objet](#) dans le Guide du AWS IoT Core développeur.

Dans cette section, vous allez installer la [Kit SDK des appareils AWS IoT version 2 pour Python](#) et exécuter l'exemple d'application Greengrass Discovery à partir du. Kit SDK des appareils AWS IoT

Note

Kit SDK des appareils AWS IoT Il est également disponible dans d'autres langages de programmation. Ce didacticiel utilise la Kit SDK des appareils AWS IoT version 2 pour Python, mais vous pouvez explorer l'autre version SDKs pour votre cas d'utilisation. Pour plus d'informations, consultez la section [AWS IoT Appareil SDKs](#) dans le guide du AWS IoT Core développeur.

Pour connecter un appareil client à un appareil principal

1. Téléchargez et installez la [Kit SDK des appareils AWS IoT version 2 pour Python](#) sur l' AWS IoT appareil à connecter en tant que périphérique client.

Sur l'appareil client, effectuez les opérations suivantes :

- a. Clonez le dépôt Kit SDK des appareils AWS IoT v2 pour Python pour le télécharger.

```
git clone https://github.com/aws/aws-iot-device-sdk-python-v2.git
```

- b. Installez la Kit SDK des appareils AWS IoT version 2 pour Python.

```
python3 -m pip install --user ./aws-iot-device-sdk-python-v2
```

2. Accédez au dossier d'échantillons dans la Kit SDK des appareils AWS IoT version 2 pour Python.

```
cd aws-iot-device-sdk-python-v2/samples/greengrass
```

3. Exécutez l'exemple d'application de découverte Greengrass. Cette application attend des arguments qui spécifient le nom de l'objet du périphérique client, le sujet MQTT et le message à utiliser, ainsi que les certificats qui authentifient et sécurisent la connexion. L'exemple suivant envoie un message Hello World à la clients/*MyClientDevice1*/hello/world rubrique.

Note

Cette rubrique correspond à la rubrique dans laquelle vous avez configuré le pont MQTT pour relayer des messages à une version AWS IoT Core antérieure.

- *MyClientDevice1* Remplacez-le par le nom de l'objet de l'appareil client.
- *~/certs/AmazonRootCA1.pem* Remplacez-le par le chemin d'accès au certificat racine de l'autorité de certification Amazon sur l'appareil client.
- Remplacez *~/certs/device.pem.crt* par le chemin d'accès au certificat de l'appareil sur l'appareil client.
- Remplacez *~/certs/private.pem.key* par le chemin d'accès au fichier de clé privée sur l'appareil client.
- Remplacez *us-east-1* par la AWS région dans laquelle fonctionnent votre appareil client et votre appareil principal.

```
python3 basic_discovery.py \  
--thing_name MyClientDevice1 \  

```

```
--topic 'clients/MyClientDevice1/hello/world' \\
--message 'Hello World!' \\
--ca_file ~/certs/AmazonRootCA1.pem \\
--cert ~/certs/device.pem.crt \\
--key ~/certs/private.pem.key \\
--region us-east-1 \\
--verbosity Warn
```

L'exemple d'application de découverte envoie le message 10 fois et se déconnecte. Il s'abonne également au même sujet où il publie des messages. Si le résultat indique que l'application a reçu des messages MQTT sur le sujet, le dispositif client peut communiquer avec succès avec le dispositif principal.

```
Performing greengrass discovery...
awsiot.greengrass_discovery.DiscoverResponse(gg_groups=[awsiot.greengrass_discovery.GGGroup
coreDevice-MyGreengrassCore',
  cores=[awsiot.greengrass_discovery.GGCore(thing_arn='arn:aws:iot:us-
east-1:123456789012:thing/MyGreengrassCore',
  connectivity=[awsiot.greengrass_discovery.ConnectivityInfo(id='203.0.113.0',
  host_address='203.0.113.0', metadata='', port=8883)])),
  certificate_authorities=['-----BEGIN CERTIFICATE-----\
MIICiT...EXAMPLE=\
-----END CERTIFICATE-----\
'])])
Trying core arn:aws:iot:us-east-1:123456789012:thing/MyGreengrassCore at host
203.0.113.0 port 8883
Connected!
Published topic clients/MyClientDevice1/hello/world: {"message": "Hello World!",
  "sequence": 0}

Publish received on topic clients/MyClientDevice1/hello/world
b'{"message": "Hello World!", "sequence": 0}'
Published topic clients/MyClientDevice1/hello/world: {"message": "Hello World!",
  "sequence": 1}

Publish received on topic clients/MyClientDevice1/hello/world
b'{"message": "Hello World!", "sequence": 1}'

...

Published topic clients/MyClientDevice1/hello/world: {"message": "Hello World!",
  "sequence": 9}
```

```
Publish received on topic clients/MyClientDevice1/hello/world  
b'{"message": "Hello World!", "sequence": 9}'
```

Si l'application génère plutôt une erreur, reportez-vous à la section [Résolution des problèmes liés à la découverte de Greengrass](#).

Vous pouvez également consulter les journaux Greengrass sur l'appareil principal pour vérifier si le périphérique client se connecte et envoie des messages avec succès. Pour de plus amples informations, veuillez consulter [AWS IoT Greengrass Journaux de surveillance](#).

4. Vérifiez que le pont MQTT relaie les messages du périphérique client vers AWS IoT Core. Vous pouvez utiliser le client de test MQTT dans la AWS IoT Core console pour vous abonner à un filtre de rubrique MQTT. Procédez comme suit :
 - a. Accédez à la [console AWS IoT](#).
 - b. Dans le menu de navigation de gauche, sous Test, choisissez le client de test MQTT.
 - c. Dans l'onglet S'abonner à un sujet, dans le champ Filtre de sujet, entrez pour vous abonner `clients/+ /hello/world` aux messages de l'appareil client à partir de l'appareil principal.
 - d. Choisissez Abonner.
 - e. Exécutez à nouveau l' `publish/subscribe` application sur l'appareil client.

Le client de test MQTT affiche les messages que vous envoyez depuis le périphérique client sur des sujets qui correspondent à ce filtre de sujets.

Étape 4 : développer un composant qui communique avec les appareils clients

Vous pouvez développer des composants Greengrass qui communiquent avec les appareils clients. Les composants utilisent [la communication interprocessus \(IPC\)](#) et l' [publish/subscribe interface locale](#) pour communiquer sur un périphérique principal. Pour interagir avec les appareils clients, configurez le composant du pont MQTT pour relayer les messages entre les appareils clients et l' `publish/subscribe` interface locale.

Dans cette section, vous allez mettre à jour le composant du pont MQTT pour relayer les messages des appareils clients vers l' `publish/subscribe` interface locale. Ensuite, vous développez un composant qui s'abonne à ces messages et les imprime lorsqu'il les reçoit.

Pour développer un composant qui communique avec les appareils clients

1. Révisez le déploiement sur le périphérique principal et configurez le composant du pont MQTT pour relayer les messages des appareils clients vers la publication/l'abonnement locaux. Procédez comme suit :

- a. Accédez à la [console AWS IoT Greengrass](#).
- b. Dans le menu de navigation de gauche, choisissez Core devices.
- c. Sur la page des appareils principaux, choisissez le périphérique principal que vous utilisez pour ce didacticiel.
- d. Sur la page des détails de l'appareil principal, choisissez l'onglet Appareils clients.
- e. Dans l'onglet Appareils clients, choisissez Configurer la découverte du cloud.

La page Configurer la découverte du périphérique principal s'ouvre. Sur cette page, vous pouvez modifier ou configurer les composants de l'appareil client déployés sur le périphérique principal.

- f. À l'étape 3, pour le `aws.greengrass.clientdevices.mqtt.Bridgecomposant`, choisissez Modifier la configuration.
- g. Dans le mode Modifier la configuration pour le composant du pont MQTT, configurez un mappage de rubriques qui relaie les messages MQTT des appareils clients vers l'interface locale. `publish/subscribe` Procédez comme suit :
 - i. Sous Configuration, dans le bloc de code Configuration pour fusionner, entrez la configuration suivante. Cette configuration indique de relayer les messages MQTT sur des sujets qui correspondent au filtre de `clients/+/hello/world` sujets depuis les appareils clients vers le service AWS IoT Core cloud et le courtier `publish/subscribe` Greengrass local.

```
{
  "mqttTopicMapping": {
    "HelloWorldIotCoreMapping": {
      "topic": "clients/+/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    },
    "HelloWorldPubsubMapping": {
      "topic": "clients/+/hello/world",
      "source": "LocalMqtt",
```



```
        "target": "Pubsub"  
      }  
    }  
  }
```

Pour plus d'informations, consultez la section [Configuration des composants du pont MQTT](#).

- ii. Choisissez Confirmer.
 - h. Choisissez Vérifier et déployer pour passer en revue le déploiement que cette page crée pour vous.
 - i. Sur la page Révision, choisissez Déployer pour démarrer le déploiement sur le périphérique principal.
 - j. Pour vérifier que le déploiement est réussi, vérifiez l'état du déploiement et consultez les journaux sur le périphérique principal. Pour vérifier l'état du déploiement sur le périphérique principal, vous pouvez choisir Target dans l'aperçu du déploiement. Pour plus d'informations, consultez les ressources suivantes :
 - [Vérifier l'état du déploiement](#)
 - [AWS IoT Greengrass Journaux de surveillance](#)
2. Développez et déployez un composant Greengrass qui s'abonne aux messages Hello World depuis les appareils clients. Procédez comme suit :
- a. Créez des dossiers pour les recettes et les artefacts sur l'appareil principal.

Linux or Unix

```
mkdir recipes  
mkdir -p artifacts/com.example.clientdevices.MyHelloWorldSubscriber/1.0.0
```

Windows Command Prompt (CMD)

```
mkdir recipes  
mkdir artifacts\com.example.clientdevices.MyHelloWorldSubscriber\1.0.0
```

PowerShell

```
mkdir recipes
```

```
mkdir artifacts\com.example.clientdevices.MyHelloWorldSubscriber\1.0.0
```

 Important

Vous devez utiliser le format suivant pour le chemin du dossier d'artefacts. Incluez le nom et la version du composant que vous spécifiez dans la recette.

```
artifacts/componentName/componentVersion/
```

- b. Utilisez un éditeur de texte pour créer une recette de composant avec le contenu suivant. Cette recette indique d'installer la Kit SDK des appareils AWS IoT version 2 pour Python et d'exécuter un script qui s'abonne au sujet et imprime des messages.

Par exemple, sur un système basé sur Linux, vous pouvez exécuter la commande suivante pour utiliser GNU nano pour créer le fichier.

```
nano recipes/com.example.clientdevices.MyHelloWorldSubscriber-1.0.0.json
```

Copiez la recette suivante dans le fichier.

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.clientdevices.MyHelloWorldSubscriber",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that subscribes to Hello World messages from client devices.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.pubsub": {
          "com.example.clientdevices.MyHelloWorldSubscriber:pubsub:1": {
            "policyDescription": "Allows access to subscribe to all topics.",
            "operations": [
              "aws.greengrass#SubscribeToTopic"
            ],
            "resources": [
              "*"
            ]
          }
        }
      }
    }
  }
}
```

```

    }
  }
}
},
"Manifests": [
  {
    "Platform": {
      "os": "linux"
    },
    "Lifecycle": {
      "install": "python3 -m pip install --user awsiotsdk",
      "Run": "python3 -u {artifacts:path}/hello_world_subscriber.py"
    }
  },
  {
    "Platform": {
      "os": "windows"
    },
    "Lifecycle": {
      "install": "py -3 -m pip install --user awsiotsdk",
      "Run": "py -3 -u {artifacts:path}/hello_world_subscriber.py"
    }
  }
]
}

```

- c. Utilisez un éditeur de texte pour créer un artefact de script Python `hello_world_subscriber.py` dont le nom est le suivant. Cette application utilise le service publish/subscribe IPC pour s'abonner au `clients/+ /hello/world` sujet et imprimer les messages qu'elle reçoit.

Par exemple, sur un système basé sur Linux, vous pouvez exécuter la commande suivante pour utiliser GNU nano pour créer le fichier.

```
nano artifacts/com.example.clientdevices.MyHelloWorldSubscriber/1.0.0/
hello_world_subscriber.py
```

Copiez le code Python suivant dans le fichier.

```
import sys
import time
```

```
import traceback

from awsiot.greengrasscoreipc.clientv2 import GreengrassCoreIPCClientV2

CLIENT_DEVICE_HELLO_WORLD_TOPIC = 'clients+/hello/world'
TIMEOUT = 10

def on_hello_world_message(event):
    try:
        message = str(event.binary_message.message, 'utf-8')
        print('Received new message: %s' % message)
    except:
        traceback.print_exc()

try:
    ipc_client = GreengrassCoreIPCClientV2()

    # SubscribeToTopic returns a tuple with the response and the operation.
    _, operation = ipc_client.subscribe_to_topic(
        topic=CLIENT_DEVICE_HELLO_WORLD_TOPIC,
        on_stream_event=on_hello_world_message)
    print('Successfully subscribed to topic: %s' %
          CLIENT_DEVICE_HELLO_WORLD_TOPIC)

    # Keep the main thread alive, or the process will exit.
    try:
        while True:
            time.sleep(10)
    except InterruptedError:
        print('Subscribe interrupted.')

    operation.close()
except Exception:
    print('Exception occurred when using IPC.', file=sys.stderr)
    traceback.print_exc()
    exit(1)
```

Note

Ce composant utilise le client IPC V2 dans le [Kit SDK des appareils AWS IoT V2 pour que Python](#) communique avec le logiciel AWS IoT Greengrass Core. Par

rapport au client IPC d'origine, le client IPC V2 réduit la quantité de code que vous devez écrire pour utiliser IPC dans des composants personnalisés.

- d. Utilisez la CLI Greengrass pour déployer le composant.

Linux or Unix

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

Windows Command Prompt (CMD)

```
C:\greengrass\v2\bin>greengrass-cli deployment create ^  
  --recipeDir recipes ^  
  --artifactDir artifacts ^  
  --merge "com.example.clientdevices.MyHelloWorldSubscriber=1.0.0"
```

PowerShell

```
C:\greengrass\v2\bin>greengrass-cli deployment create `  
  --recipeDir recipes `  
  --artifactDir artifacts `  
  --merge "com.example.clientdevices.MyHelloWorldSubscriber=1.0.0"
```

3. Consultez les journaux des composants pour vérifier que le composant s'installe correctement et s'abonne à la rubrique.

Linux or Unix

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

PowerShell

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

Vous pouvez laisser le flux du journal ouvert pour vérifier que le périphérique principal reçoit les messages.

4. Sur l'appareil client, exécutez à nouveau l'exemple d'application de découverte Greengrass pour envoyer des messages au périphérique principal.

```
python3 basic_discovery.py \\  
  --thing_name MyClientDevice1 \\  
  --topic 'clients/MyClientDevice1/hello/world' \\  
  --message 'Hello World!' \\  
  --ca_file ~/certs/AmazonRootCA1.pem \\  
  --cert ~/certs/device.pem.crt \\  
  --key ~/certs/private.pem.key \\  
  --region us-east-1 \\  
  --verbosity Warn
```

5. Consultez à nouveau les journaux du composant pour vérifier que le composant reçoit et imprime les messages provenant de l'appareil client.

Linux or Unix

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

PowerShell

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

Étape 5 : développer un composant qui interagit avec les ombres du périphérique client

Vous pouvez développer des composants Greengrass qui interagissent avec les ombres de l'[AWS IoT appareil](#) client. Un shadow est un document JSON qui stocke les informations d'état actuelles ou souhaitées pour un AWS IoT objet, tel qu'un appareil client. Les composants personnalisés peuvent accéder aux ombres des appareils clients pour gérer leur état, même lorsque le périphérique client n'est pas connecté à AWS IoT. Chaque AWS IoT objet possède une ombre sans nom, et vous pouvez également créer plusieurs ombres nommées pour chaque élément.

Dans cette section, vous déployez le [composant Shadow Manager](#) pour gérer les ombres sur le périphérique principal. Vous mettez également à jour le composant pont MQTT pour relayer les messages instantanés entre les appareils clients et le composant Shadow Manager. Ensuite, vous développez un composant qui met à jour les ombres des appareils clients, et vous exécutez un exemple d'application sur les appareils clients qui répond aux mises à jour instantanées du composant. Ce composant représente une application de gestion intelligente de l'éclairage, dans laquelle le périphérique principal gère l'état des couleurs des lampes intelligentes qui s'y connectent en tant que périphériques clients.

Pour développer un composant qui interagit avec les ombres du périphérique client

1. Révisez le déploiement sur le périphérique principal pour déployer le composant Shadow Manager et configurez le composant MQTT Bridge pour relayer les messages instantanés entre les appareils clients et les publications/abonnements locaux, où le Shadow Manager communique. Procédez comme suit :
 - a. Accédez à la [console AWS IoT Greengrass](#).
 - b. Dans le menu de navigation de gauche, choisissez Core devices.
 - c. Sur la page des appareils principaux, choisissez le périphérique principal que vous utilisez pour ce didacticiel.
 - d. Sur la page des détails de l'appareil principal, choisissez l'onglet Appareils clients.
 - e. Dans l'onglet Appareils clients, choisissez Configurer la découverte du cloud.

La page Configurer la découverte du périphérique principal s'ouvre. Sur cette page, vous pouvez modifier ou configurer les composants de l'appareil client déployés sur le périphérique principal.

- f. À l'étape 3, pour le `aws.greengrass.clientdevices.mqtt.Bridgecomposant`, choisissez Modifier la configuration.
- g. Dans le mode Modifier la configuration pour le composant du pont MQTT, configurez un mappage de sujets qui relaie les messages MQTT sur les [sujets](#) cachés entre les appareils clients et l'interface de publication/d'abonnement locale. Vous confirmez également que le déploiement spécifie une version de pont MQTT compatible. La prise en charge du shadow sur le périphérique client nécessite le pont MQTT v2.2.0 ou version ultérieure. Procédez comme suit :
 - i. Pour la version du composant, choisissez la version 2.2.0 ou ultérieure.

- ii. Sous Configuration, dans le bloc de code Configuration pour fusionner, entrez la configuration suivante. Cette configuration indique de relayer les messages MQTT sur des sujets cachés.

```
{
  "mqttTopicMapping": {
    "HelloWorldIotCoreMapping": {
      "topic": "clients/+/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    },
    "HelloWorldPubsubMapping": {
      "topic": "clients/+/hello/world",
      "source": "LocalMqtt",
      "target": "Pubsub"
    },
    "ShadowsLocalMqttToPubsub": {
      "topic": "$aws/things/+/shadow/#",
      "source": "LocalMqtt",
      "target": "Pubsub"
    },
    "ShadowsPubsubToLocalMqtt": {
      "topic": "$aws/things/+/shadow/#",
      "source": "Pubsub",
      "target": "LocalMqtt"
    }
  }
}
```

Pour plus d'informations, consultez la section [Configuration des composants du pont MQTT](#).

- iii. Choisissez Confirmer.
- h. À l'étape 3, sélectionnez le `aws.greengrass.ShadowManager` composant à déployer.
- i. Choisissez Vérifier et déployer pour passer en revue le déploiement que cette page crée pour vous.
- j. Sur la page Révision, choisissez Déployer pour démarrer le déploiement sur le périphérique principal.
- k. Pour vérifier que le déploiement est réussi, vérifiez l'état du déploiement et consultez les journaux sur le périphérique principal. Pour vérifier l'état du déploiement sur le

périphérique principal, vous pouvez choisir Target dans l'aperçu du déploiement. Pour plus d'informations, consultez les ressources suivantes :

- [Vérifier l'état du déploiement](#)
- [AWS IoT Greengrass Journaux de surveillance](#)

2. Développez et déployez un composant Greengrass qui gère les appareils clients Smart Light. Procédez comme suit :

- a. Créez un dossier contenant les artefacts du composant sur le périphérique principal.

Linux or Unix

```
mkdir -p artifacts/com.example.clientdevices.MySmartLightManager/1.0.0
```

Windows Command Prompt (CMD)

```
mkdir artifacts\com.example.clientdevices.MySmartLightManager\1.0.0
```

PowerShell

```
mkdir artifacts\com.example.clientdevices.MySmartLightManager\1.0.0
```

Important

Vous devez utiliser le format suivant pour le chemin du dossier d'artefacts. Incluez le nom et la version du composant que vous spécifiez dans la recette.

```
artifacts/componentName/componentVersion/
```

- b. Utilisez un éditeur de texte pour créer une recette de composant avec le contenu suivant. Cette recette indique d'installer la Kit SDK des appareils AWS IoT version 2 pour Python et d'exécuter un script qui interagit avec les ombres des appareils clients Smart Light pour gérer leurs couleurs.

Par exemple, sur un système basé sur Linux, vous pouvez exécuter la commande suivante pour utiliser GNU nano pour créer le fichier.

```
nano recipes/com.example.clientdevices.MySmartLightManager-1.0.0.json
```

Copiez la recette suivante dans le fichier.

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.clientdevices.MySmartLightManager",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that interacts with smart light client
  devices.",
  "ComponentPublisher": "Amazon",
  "ComponentDependencies": {
    "aws.greengrass.Nucleus": {
      "VersionRequirement": "^2.6.0"
    },
    "aws.greengrass.ShadowManager": {
      "VersionRequirement": "^2.2.0"
    },
    "aws.greengrass.clientdevices.mqtt.Bridge": {
      "VersionRequirement": "^2.2.0"
    }
  },
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "smartLightDeviceNames": [],
      "accessControl": {
        "aws.greengrass.ShadowManager": {
          "com.example.clientdevices.MySmartLightManager:shadow:1": {
            "policyDescription": "Allows access to client devices' unnamed
            shadows",
            "operations": [
              "aws.greengrass#GetThingShadow",
              "aws.greengrass#UpdateThingShadow"
            ],
            "resources": [
              "$aws/things/MyClientDevice*/shadow"
            ]
          }
        }
      },
      "aws.greengrass.ipc.pubsub": {
        "com.example.clientdevices.MySmartLightManager:pubsub:1": {
```

```

        "policyDescription": "Allows access to client devices' unnamed
shadow updates",
        "operations": [
            "aws.greengrass#SubscribeToTopic"
        ],
        "resources": [
            "$aws/things/+/shadow/update/accepted"
        ]
    }
}
},
"Manifests": [
    {
        "Platform": {
            "os": "linux"
        },
        "Lifecycle": {
            "install": "python3 -m pip install --user awsiot-sdk",
            "Run": "python3 -u {artifacts:path}/smart_light_manager.py"
        }
    },
    {
        "Platform": {
            "os": "windows"
        },
        "Lifecycle": {
            "install": "py -3 -m pip install --user awsiot-sdk",
            "Run": "py -3 -u {artifacts:path}/smart_light_manager.py"
        }
    }
]
}

```

- c. Utilisez un éditeur de texte pour créer un artefact de script Python `smart_light_manager.py` dont le nom est le suivant. Cette application utilise le service IPC fantôme pour obtenir et mettre à jour les ombres du périphérique client et le service publish/subscribe IPC local pour recevoir les mises à jour instantanées signalées.

Par exemple, sur un système basé sur Linux, vous pouvez exécuter la commande suivante pour utiliser GNU nano pour créer le fichier.

```
nano artifacts/com.example.clientdevices.MySmartLightManager/1.0.0/  
smart_light_manager.py
```

Copiez le code Python suivant dans le fichier.

```
import json  
import random  
import sys  
import time  
import traceback  
from uuid import uuid4  
  
from awsiot.greengrasscoreipc.clientv2 import GreengrassCoreIPCClientV2  
from awsiot.greengrasscoreipc.model import ResourceNotFoundError  
  
SHADOW_COLOR_PROPERTY = 'color'  
CONFIGURATION_CLIENT_DEVICE_NAMES = 'smartLightDeviceNames'  
COLORS = ['red', 'orange', 'yellow', 'green', 'blue', 'purple']  
SHADOW_UPDATE_TOPIC = '$aws/things/+/shadow/update/accepted'  
SET_COLOR_INTERVAL = 15  
  
class SmartLightDevice():  
    def __init__(self, client_device_name: str, reported_color: str = None):  
        self.name = client_device_name  
        self.reported_color = reported_color  
        self.desired_color = None  
  
class SmartLightDeviceManager():  
    def __init__(self, ipc_client: GreengrassCoreIPCClientV2):  
        self.ipc_client = ipc_client  
        self.devices = {}  
        self.client_tokens = set()  
        self.shadow_update_accepted_subscription_operation = None  
        self.client_device_names_configuration_subscription_operation = None  
        self.update_smart_light_device_list()  
  
    def update_smart_light_device_list(self):  
        # Update the device list from the component configuration.  
        response = self.ipc_client.get_configuration(  
            key_path=[CONFIGURATION_CLIENT_DEVICE_NAMES])
```

```

        # Identify the difference between the configuration and the currently
        tracked devices.
        current_device_names = self.devices.keys()
        updated_device_names =
        response.value[CONFIGURATION_CLIENT_DEVICE_NAMES]
        added_device_names = set(updated_device_names) -
        set(current_device_names)
        removed_device_names = set(current_device_names) -
        set(updated_device_names)

        # Stop tracking any smart light devices that are no longer in the
        configuration.
        for name in removed_device_names:
            print('Removing %s from smart light device manager' % name)
            self.devices.pop(name)

        # Start tracking any new smart light devices that are in the
        configuration.
        for name in added_device_names:
            print('Adding %s to smart light device manager' % name)
            device = SmartLightDevice(name)
            device.reported_color = self.get_device_reported_color(device)
            self.devices[name] = device
            print('Current color for %s is %s' % (name,
            device.reported_color))

    def get_device_reported_color(self, smart_light_device):
        try:
            response = self.ipc_client.get_thing_shadow(
                thing_name=smart_light_device.name, shadow_name='')
            shadow = json.loads(str(response.payload, 'utf-8'))
            if 'reported' in shadow['state']:
                return shadow['state']['reported'].get(SHADOW_COLOR_PROPERTY)
            return None
        except ResourceNotFoundError:
            return None

    def request_device_color_change(self, smart_light_device, color):
        # Generate and track a client token for the request.
        client_token = str(uuid4())
        self.client_tokens.add(client_token)
        # Create a shadow payload, which must be a blob.
        payload_json = {
            'state': {
                'desired': {
                    SHADOW_COLOR_PROPERTY: color

```

```

        }
    },
    'clientToken': client_token
}
payload = bytes(json.dumps(payload_json), 'utf-8')
self.ipc_client.update_thing_shadow(
    thing_name=smart_light_device.name, shadow_name='',
payload=payload)
smart_light_device.desired_color = color

def subscribe_to_shadow_update_accepted_events(self):
    if self.shadow_update_accepted_subscription_operation == None:
        # SubscribeToTopic returns a tuple with the response and the
operation.
        _, self.shadow_update_accepted_subscription_operation =
self.ipc_client.subscribe_to_topic(
            topic=SHADOW_UPDATE_TOPIC,
on_stream_event=self.on_shadow_update_accepted_event)
        print('Successfully subscribed to shadow update accepted topic')

def close_shadow_update_accepted_subscription(self):
    if self.shadow_update_accepted_subscription_operation is not None:
        self.shadow_update_accepted_subscription_operation.close()

def on_shadow_update_accepted_event(self, event):
    try:
        message = str(event.binary_message.message, 'utf-8')
        accepted_payload = json.loads(message)
        # Check for reported states from smart light devices and ignore
desired states from components.
        if 'reported' in accepted_payload['state']:
            # Process this update only if it uses a client token created by
this component.
            client_token = accepted_payload.get('clientToken')
            if client_token is not None and client_token in
self.client_tokens:
                self.client_tokens.remove(client_token)
                shadow_state = accepted_payload['state']['reported']
                if SHADOW_COLOR_PROPERTY in shadow_state:
                    reported_color = shadow_state[SHADOW_COLOR_PROPERTY]
                    topic = event.binary_message.context.topic
                    client_device_name = topic.split('/')[2]
                    if client_device_name in self.devices:

```

```

        # Set the reported color for the smart light
device.
        self.devices[client_device_name].reported_color =
reported_color

        print(
            'Received shadow update confirmation from
client device: %s' % client_device_name)
        else:
            print("Shadow update doesn't specify color")
    except:
        traceback.print_exc()

    def subscribe_to_client_device_name_configuration_updates(self):
        if self.client_device_names_configuration_subscription_operation ==
None:
            # SubscribeToConfigurationUpdate returns a tuple with the response
and the operation.
            _, self.client_device_names_configuration_subscription_operation =
self.ipc_client.subscribe_to_configuration_update(
                key_path=[CONFIGURATION_CLIENT_DEVICE_NAMES],
on_stream_event=self.on_client_device_names_configuration_update_event)
            print(
                'Successfully subscribed to configuration updates for smart
light device names')

    def close_client_device_names_configuration_subscription(self):
        if self.client_device_names_configuration_subscription_operation is not
None:
            self.client_device_names_configuration_subscription_operation.close()

    def on_client_device_names_configuration_update_event(self, event):
        try:
            if CONFIGURATION_CLIENT_DEVICE_NAMES in
event.configuration_update_event.key_path:
                print('Received configuration update for list of client
devices')
                self.update_smart_light_device_list()
        except:
            traceback.print_exc()

    def choose_random_color():
        return random.choice(COLORS)

```

```

def main():
    try:
        # Create an IPC client and a smart light device manager.
        ipc_client = GreengrassCoreIPCClientV2()
        smart_light_manager = SmartLightDeviceManager(ipc_client)
        smart_light_manager.subscribe_to_shadow_update_accepted_events()

    smart_light_manager.subscribe_to_client_device_name_configuration_updates()
    try:
        # Keep the main thread alive, or the process will exit.
        while True:
            # Set each smart light device to a random color at a regular
            interval.
            for device_name in smart_light_manager.devices:
                device = smart_light_manager.devices[device_name]
                desired_color = choose_random_color()
                print('Chose random color (%s) for %s' %
                      (desired_color, device_name))
                if desired_color == device.desired_color:
                    print('Desired color for %s is already %s' %
                          (device_name, desired_color))
                elif desired_color == device.reported_color:
                    print('Reported color for %s is already %s' %
                          (device_name, desired_color))
                else:
                    smart_light_manager.request_device_color_change(
                        device, desired_color)
                    print('Requested color change for %s to %s' %
                          (device_name, desired_color))
                    time.sleep(SET_COLOR_INTERVAL)
            except InterruptedError:
                print('Application interrupted')
                smart_light_manager.close_shadow_update_accepted_subscription()

    smart_light_manager.close_client_device_names_configuration_subscription()
    except Exception:
        print('Exception occurred', file=sys.stderr)
        traceback.print_exc()
        exit(1)

if __name__ == '__main__':

```



```
main()
```

Cette application Python effectue les opérations suivantes :

- Lit la configuration du composant pour obtenir la liste des appareils clients Smart Light à gérer.
 - S'abonne aux notifications de mise à jour de configuration à l'aide de l'opération [SubscribeToConfigurationUpdate](#) IPC. Le logiciel AWS IoT Greengrass Core envoie des notifications chaque fois que la configuration du composant change. Lorsque le composant reçoit une notification de mise à jour de configuration, il met à jour la liste des appareils clients Smart Light qu'il gère.
 - Obtient l'ombre de chaque appareil client Smart Light pour obtenir son état de couleur initial.
 - Règle la couleur de chaque appareil client Smart Light sur une couleur aléatoire toutes les 15 secondes. Le composant met à jour l'ombre de l'objet de l'appareil client pour en changer la couleur. Cette opération envoie un événement shadow delta au périphérique client via MQTT.
 - S'abonne à la mise à jour instantanée des messages acceptés sur l'interface de publication/d'abonnement locale à l'aide de l'[SubscribeToTopic](#) opération IPC. Ce composant reçoit ces messages pour suivre la couleur de chaque dispositif client Smart Light. Lorsqu'un appareil client Smart Light reçoit une mise à jour parallèle, il envoie un message MQTT pour confirmer qu'il a reçu la mise à jour. Le pont MQTT transmet ce message à l'interface locale publish/subscribe .
- d. Utilisez la CLI Greengrass pour déployer le composant. Lorsque vous déployez ce composant, vous spécifiez la liste des appareils clients `smartLightDeviceNames` dont il gère les ombres. *MyClientDevice1* Remplacez-le par le nom de l'objet de l'appareil client.

Linux or Unix

```
sudo /greengrass/v2/bin/greengrass-cli deployment create \  
  --recipeDir recipes \  
  --artifactDir artifacts \  
  --merge "com.example.clientdevices.MySmartLightManager=1.0.0" \  
  --update-config '{  
    "com.example.clientdevices.MySmartLightManager": {  
      "MERGE": {  
        "smartLightDeviceNames": [  
          "MyClientDevice1"  
        ]  
      }  
    }  
  }'
```

```
    ]
  }
}
}'
```

Windows Command Prompt (CMD)

```
C:\greengrass\v2\bin\greengrass-cli deployment create ^
--recipeDir recipes ^
--artifactDir artifacts ^
--merge "com.example.clientdevices.MySmartLightManager=1.0.0" ^
--update-config '{"com.example.clientdevices.MySmartLightManager":
{"MERGE":{"smartLightDeviceNames":["MyClientDevice1"]}}}'
```

PowerShell

```
C:\greengrass\v2\bin\greengrass-cli deployment create `
--recipeDir recipes `
--artifactDir artifacts `
--merge "com.example.clientdevices.MySmartLightManager=1.0.0" `
--update-config '{
  "com.example.clientdevices.MySmartLightManager": {
    "MERGE": {
      "smartLightDeviceNames": [
        "MyClientDevice1"
      ]
    }
  }
}'
```

3. Consultez les journaux des composants pour vérifier que le composant s'installe et s'exécute correctement.

Linux or Unix

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

PowerShell

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

Le composant envoie des demandes pour changer la couleur du dispositif client Smart Light. Le gestionnaire d'ombres reçoit la demande et définit l'`desiredÉtat` de l'ombre. Cependant, le périphérique client Smart Light ne fonctionne pas encore, donc l'`reportedÉtat` de l'ombre ne change pas. Les journaux du composant incluent les messages suivants.

```
2022-07-07T03:49:24.908Z [INFO] (Copier)
com.example.clientdevices.MySmartLightManager: stdout. Chose random color (blue)
for MyClientDevice1.
{scriptName=services.com.example.clientdevices.MySmartLightManager.lifecycle.Run,
serviceName=com.example.clientdevices.MySmartLightManager, currentState=RUNNING}
2022-07-07T03:49:24.912Z [INFO] (Copier)
com.example.clientdevices.MySmartLightManager: stdout.
Requested color change for MyClientDevice1 to blue.
{scriptName=services.com.example.clientdevices.MySmartLightManager.lifecycle.Run,
serviceName=com.example.clientdevices.MySmartLightManager, currentState=RUNNING}
```

Vous pouvez laisser le flux du journal ouvert pour voir quand le composant imprime des messages.

4. Téléchargez et exécutez un exemple d'application qui utilise Greengrass Discovery et qui s'abonne aux mises à jour instantanées de l'appareil. Sur l'appareil client, effectuez les opérations suivantes :
 - a. Utilisez un éditeur de texte pour créer un script Python `basic_discovery_shadow.py` dont le nom est le suivant. Cette application utilise Greengrass Discovery et les ombres pour synchroniser une propriété entre le périphérique client et le périphérique principal.

Par exemple, sur un système basé sur Linux, vous pouvez exécuter la commande suivante pour utiliser GNU nano pour créer le fichier.

```
nano basic_discovery_shadow.py
```

Copiez le code Python suivant dans le fichier.

```

# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0.

from awscrt import io
from awscrt import mqtt
from awsiot import iotshadow
from awsiot.greengrass_discovery import DiscoveryClient
from awsiot import mqtt_connection_builder
from concurrent.futures import Future
import sys
import threading
import traceback
from uuid import uuid4

# ----- ARGUMENT PARSING
# -----
import argparse

def parse_sample_input():
    parser = argparse.ArgumentParser(
        description="AWS IoT Greengrass Discovery Shadow",
        formatter_class=argparse.ArgumentDefaultsHelpFormatter,
    )

    # Connection / TLS
    parser.add_argument("--cert", required=True, dest="input_cert",
                        help="Path to the certificate file to use during mTLS connection establishment")
    parser.add_argument("--key", required=True, dest="input_key",
                        help="Path to the private key file to use during mTLS connection establishment")
    parser.add_argument("--ca_file", dest="input_ca", help="Path to optional CA bundle (PEM)")

    # Shadow
    parser.add_argument("--thing_name", required=True, dest="input_thing_name",
                        help="The name assigned to your IoT Thing.")
    parser.add_argument("--shadow_property", dest="input_shadow_property",
                        default="color",
                        help="The name of the shadow property you want to change (optional, default='color')")
    parser.add_argument("--region", dest="input_region", help="The region to connect through.", required=True)

```

```

    parser.add_argument("--print_discover_resp_only",
                        dest="input_print_discover_resp_only", default=False)

    return parser.parse_args()

args = parse_sample_input()

# ----- ARGUMENT PARSING END
# -----

# Using globals to simplify sample code
is_sample_done = threading.Event()
mqtt_connection = None
shadow_thing_name = args.input_thing_name
shadow_property = args.input_shadow_property

SHADOW_VALUE_DEFAULT = "off"

class LockedData:
    def __init__(self):
        self.lock = threading.Lock()
        self.shadow_value = None
        self.disconnect_called = False
        self.request_tokens = set()

locked_data = LockedData()

def on_connection_interrupted(connection, error, **kwargs):
    print('connection interrupted with error {}'.format(error))

def on_connection_resumed(connection, return_code, session_present, **kwargs):
    print('connection resumed with return code {}, session present
    {}'.format(return_code, session_present))

# Try IoT endpoints until we find one that works
def try_iot_endpoints():
    for gg_group in discover_response.gg_groups:
        for gg_core in gg_group.cores:
            for connectivity_info in gg_core.connectivity:
                try:

```

```

        print('Trying core {} at host {} port
        {}'.format(gg_core.thing_arn, connectivity_info.host_address,
        connectivity_info.port))
        mqtt_connection = mqtt_connection_builder.mtls_from_path(
            endpoint=connectivity_info.host_address,
            port=connectivity_info.port,
            cert_filepath=args.input_cert,
            pri_key_filepath=args.input_key,

        ca_bytes=gg_group.certificate_authorities[0].encode('utf-8'),
            on_connection_interrupted=on_connection_interrupted,
            on_connection_resumed=on_connection_resumed,
            client_id=args.input_thing_name,
            clean_session=False,
            keep_alive_secs=30)

        connect_future = mqtt_connection.connect()
        connect_future.result()
        print('Connected!')
        return mqtt_connection

    except Exception as e:
        print('Connection failed with exception {}'.format(e))
        continue

    exit('All connection attempts failed')

# Function for gracefully quitting this sample
def exit(msg_or_exception):
    if isinstance(msg_or_exception, Exception):
        print("Exiting sample due to exception.")
        traceback.print_exception(msg_or_exception.__class__, msg_or_exception,
        sys.exc_info()[2])
    else:
        print("Exiting sample:", msg_or_exception)

    with locked_data.lock:
        if not locked_data.disconnect_called:
            print("Disconnecting...")
            locked_data.disconnect_called = True
            future = mqtt_connection.disconnect()
            future.add_done_callback(on_disconnected)

def on_disconnected(disconnect_future):

```

```

    # type: (Future) -> None
    print("Disconnected.")

    # Signal that sample is finished
    is_sample_done.set()

def on_get_shadow_accepted(response):
    # type: (iotshadow.GetShadowResponse) -> None
    try:
        with locked_data.lock:
            # check that this is a response to a request from this session
            try:
                locked_data.request_tokens.remove(response.client_token)
            except KeyError:
                return

            print("Finished getting initial shadow state.")
            if locked_data.shadow_value is not None:
                print(" Ignoring initial query because a delta event has
already been received.")
                return

            if response.state:
                if response.state.delta:
                    value = response.state.delta.get(shadow_property)
                    if value:
                        print(" Shadow contains delta value '{}'.format(value))
                        change_shadow_value(value)
                        return

                if response.state.reported:
                    value = response.state.reported.get(shadow_property)
                    if value:
                        print(" Shadow contains reported value
'{}'.format(value))

set_local_value_due_to_initial_query(response.state.reported[shadow_property])
                return

            print(" Shadow document lacks '{}' property. Setting
defaults...".format(shadow_property))
            change_shadow_value(SHADOW_VALUE_DEFAULT)
            return

```

```

except Exception as e:
    exit(e)

def on_get_shadow_rejected(error):
    # type: (iotshadow.ErrorResponse) -> None
    try:
        # check that this is a response to a request from this session
        with locked_data.lock:
            try:
                locked_data.request_tokens.remove(error.client_token)
            except KeyError:
                return

        if error.code == 404:
            print("Thing has no shadow document. Creating with defaults...")
            change_shadow_value(SHADOW_VALUE_DEFAULT)
        else:
            exit("Get request was rejected. code:{} message:'{}'".format(
                error.code, error.message))

    except Exception as e:
        exit(e)

def on_shadow_delta_updated(delta):
    # type: (iotshadow.ShadowDeltaUpdatedEvent) -> None
    try:
        print("Received shadow delta event.")
        if delta.state and (shadow_property in delta.state):
            value = delta.state[shadow_property]
            if value is None:
                print("  Delta reports that '{}' was deleted. Resetting
defaults...".format(shadow_property))
                change_shadow_value(SHADOW_VALUE_DEFAULT)
                return
            else:
                print("  Delta reports that desired value is '{}'. Changing
local value...".format(value))
                if (delta.client_token is not None):
                    print ("  ClientToken is: " + delta.client_token)
                    change_shadow_value(value, delta.client_token)
                else:
                    print("  Delta did not report a change in
'{}'".format(shadow_property))

```



```
except Exception as e:
    exit(e)

def on_publish_update_shadow(future):
    #type: (Future) -> None
    try:
        future.result()
        print("Update request published.")
    except Exception as e:
        print("Failed to publish update request.")
        exit(e)

def on_update_shadow_accepted(response):
    # type: (iotshadow.UpdateShadowResponse) -> None
    try:
        # check that this is a response to a request from this session
        with locked_data.lock:
            try:
                locked_data.request_tokens.remove(response.client_token)
            except KeyError:
                return

        try:
            if response.state.reported != None:
                if shadow_property in response.state.reported:
                    print("Finished updating reported shadow value to
'{}'.format(response.state.reported[shadow_property])) # type: ignore
                else:
                    print ("Could not find shadow property with name:
'{}'.format(shadow_property)) # type: ignore
                else:
                    print("Shadow states cleared.") # when the shadow states are
cleared, reported and desired are set to None
            except:
                exit("Updated shadow is missing the target property")

    except Exception as e:
        exit(e)

def on_update_shadow_rejected(error):
    # type: (iotshadow.ErrorResponse) -> None
    try:
        # check that this is a response to a request from this session
        with locked_data.lock:
```

```

        try:
            locked_data.request_tokens.remove(error.client_token)
        except KeyError:
            return

        exit("Update request was rejected. code:{} message:'{}'".format(
            error.code, error.message))

    except Exception as e:
        exit(e)

def set_local_value_due_to_initial_query(reported_value):
    with locked_data.lock:
        locked_data.shadow_value = reported_value

def change_shadow_value(value, token=None):
    with locked_data.lock:
        if locked_data.shadow_value == value:
            print("Local value is already '{}'.format(value))
            return

        print("Changed local shadow value to '{}'.format(value))
        locked_data.shadow_value = value

        print("Updating reported shadow value to '{}...'".format(value))

        reuse_token = token is not None
        # use a unique token so we can correlate this "request" message to
        # any "response" messages received on the /accepted and /rejected
        topics
        if not reuse_token:
            token = str(uuid4())

        # if the value is "clear shadow" then send a UpdateShadowRequest with
        None
        # for both reported and desired to clear the shadow document
        completely.
        if value == "clear_shadow":
            tmp_state = iotshadow.ShadowState(reported=None, desired=None,
            reported_is_nullable=True, desired_is_nullable=True)
            request = iotshadow.UpdateShadowRequest(
                thing_name=shadow_thing_name,
                state=tmp_state,
                client_token=token,

```

```

        )
        # Otherwise, send a normal update request
    else:
        # if the value is "none" then set it to a Python none object to
        # clear the individual shadow property
        if value == "none":
            value = None

        request = iotshadow.UpdateShadowRequest(
            thing_name=shadow_thing_name,
            state=iotshadow.ShadowState(
                reported={ shadow_property: value }
            ),
            client_token=token,
        )

        future = shadow_client.publish_update_shadow(request,
            mqtt.QoS.AT_LEAST_ONCE)

        if not reuse_token:
            locked_data.request_tokens.add(token)

        future.add_done_callback(on_publish_update_shadow)

if __name__ == '__main__':
    tls_options =
    io.TlsContextOptions.create_client_with_mtls_from_path(args.input_cert,args.input_key)
    if args.input_ca:
        tls_options.override_default_trust_store_from_path(None,
args.input_ca)
    tls_context = io.ClientTlsContext(tls_options)

    socket_options = io.SocketOptions()

    print('Performing greengrass discovery...')
    discovery_client =
    DiscoveryClient(io.ClientBootstrap.get_or_create_static_default(),
socket_options, tls_context, args.input_region)
    resp_future = discovery_client.discover(args.input_thing_name)
    discover_response = resp_future.result()

    print(discover_response)
    if args.print_discover_resp_only:

```

```
        exit(0)

mqtt_connection = try_iot_endpoints()
shadow_client = iotshadow.IotShadowClient(mqtt_connection)

try:
    # Subscribe to necessary topics.
    # Note that is **is** important to wait for "accepted/rejected"
subscriptions
    # to succeed before publishing the corresponding "request".
    print("Subscribing to Update responses...")
    update_accepted_subscribed_future, _ =
shadow_client.subscribe_to_update_shadow_accepted(

request=iotshadow.UpdateShadowSubscriptionRequest(thing_name=shadow_thing_name),
        qos=mqtt.QoS.AT_LEAST_ONCE,
        callback=on_update_shadow_accepted)

    update_rejected_subscribed_future, _ =
shadow_client.subscribe_to_update_shadow_rejected(

request=iotshadow.UpdateShadowSubscriptionRequest(thing_name=shadow_thing_name),
        qos=mqtt.QoS.AT_LEAST_ONCE,
        callback=on_update_shadow_rejected)

    # Wait for subscriptions to succeed
    update_accepted_subscribed_future.result()
    update_rejected_subscribed_future.result()

    print("Subscribing to Get responses...")
    get_accepted_subscribed_future, _ =
shadow_client.subscribe_to_get_shadow_accepted(

request=iotshadow.GetShadowSubscriptionRequest(thing_name=shadow_thing_name),
        qos=mqtt.QoS.AT_LEAST_ONCE,
        callback=on_get_shadow_accepted)

    get_rejected_subscribed_future, _ =
shadow_client.subscribe_to_get_shadow_rejected(

request=iotshadow.GetShadowSubscriptionRequest(thing_name=shadow_thing_name),
        qos=mqtt.QoS.AT_LEAST_ONCE,
        callback=on_get_shadow_rejected)
```

```

        # Wait for subscriptions to succeed
        get_accepted_subscribed_future.result()
        get_rejected_subscribed_future.result()

        print("Subscribing to Delta events...")
        delta_subscribed_future, _ =
shadow_client.subscribe_to_shadow_delta_updated_events(

request=iotshadow.ShadowDeltaUpdatedSubscriptionRequest(thing_name=shadow_thing_name),
                qos=mqtt.QoS.AT_LEAST_ONCE,
                callback=on_shadow_delta_updated)

        # Wait for subscription to succeed
        delta_subscribed_future.result()

        # The rest of the sample runs asynchronously.

        # Issue request for shadow's current state.
        # The response will be received by the on_get_accepted() callback
        print("Requesting current shadow state...")

        with locked_data.lock:
            # use a unique token so we can correlate this "request" message to
            # any "response" messages received on the /accepted and /rejected
topics
            token = str(uuid4())

            publish_get_future = shadow_client.publish_get_shadow(

request=iotshadow.GetShadowRequest(thing_name=shadow_thing_name,
client_token=token),
                qos=mqtt.QoS.AT_LEAST_ONCE)

            locked_data.request_tokens.add(token)

        # Ensure that publish succeeds
        publish_get_future.result()

    except Exception as e:
        exit(e)

    # Wait for the sample to finish (user types 'quit', or an error occurs)
    is_sample_done.wait()

```

Cette application Python effectue les opérations suivantes :

- Utilise Greengrass Discovery pour découvrir l'appareil principal et s'y connecter.
- Demande le document fantôme à l'appareil principal pour obtenir l'état initial de la propriété.
- S'abonne aux événements Shadow Delta, que le périphérique principal envoie lorsque la `desired` valeur de la propriété diffère de sa `reported` valeur. Lorsque l'application reçoit un événement Shadow Delta, elle modifie la valeur de la propriété et envoie une mise à jour au périphérique principal pour définir la nouvelle valeur comme sa `reported` valeur.

Cette application combine la découverte de Greengrass et des échantillons d'ombres de la Kit SDK des appareils AWS IoT v2.

- b. Exécutez l'exemple d'application. Cette application attend des arguments qui spécifient le nom de l'objet du périphérique client, la propriété fantôme à utiliser et les certificats qui authentifient et sécurisent la connexion.
- `MyClientDevice1` Remplacez-le par le nom de l'objet de l'appareil client.
 - `~/certs/AmazonRootCA1.pem` Remplacez-le par le chemin d'accès au certificat racine de l'autorité de certification Amazon sur l'appareil client.
 - Remplacez `~/certs/device.pem.crt` par le chemin d'accès au certificat de l'appareil sur l'appareil client.
 - Remplacez `~/certs/private.pem.key` par le chemin d'accès au fichier de clé privée sur l'appareil client.
 - Remplacez `us-east-1` par la AWS région dans laquelle fonctionnent votre appareil client et votre appareil principal.

```
python3 basic_discovery_shadow.py \  
  --thing_name MyClientDevice1 \  
  --shadow_property color \  
  --ca_file ~/certs/AmazonRootCA1.pem \  
  --cert ~/certs/device.pem.crt \  
  --key ~/certs/private.pem.key \  
  --region us-east-1 \  
  --verbosity Warn
```

L'exemple d'application s'abonne aux sujets cachés et attend de recevoir les événements Shadow Delta de l'appareil principal. Si le résultat indique que l'application reçoit des événements Shadow Delta et y répond, le dispositif client peut interagir avec succès avec son ombre sur le dispositif principal.

```
Performing greengrass discovery...
awsiot.greengrass_discovery.DiscoverResponse(gg_groups=[awsiot.greengrass_discovery.GG
coreDevice-MyGreengrassCore',
  cores=[awsiot.greengrass_discovery.GGCore(thing_arn='arn:aws:iot:us-
east-1:123456789012:thing/MyGreengrassCore',
  connectivity=[awsiot.greengrass_discovery.ConnectivityInfo(id='203.0.113.0',
  host_address='203.0.113.0', metadata='', port=8883)])),
  certificate_authorities=['-----BEGIN CERTIFICATE-----
\nMIICiT...EXAMPLE=\n-----END CERTIFICATE-----\n']]))
Trying core arn:aws:iot:us-east-1:123456789012:thing/MyGreengrassCore at host
203.0.113.0 port 8883
Connected!
Subscribing to Update responses...
Subscribing to Get responses...
Subscribing to Delta events...
Requesting current shadow state...
Received shadow delta event.
  Delta reports that desired value is 'purple'. Changing local value...
  ClientToken is: 3dce4d3f-e336-41ac-aa4f-7882725f0033
Changed local shadow value to 'purple'.
Updating reported shadow value to 'purple'...
Update request published.
```

Si l'application génère plutôt une erreur, reportez-vous à la section [Résolution des problèmes liés à la découverte de Greengrass](#).

Vous pouvez également consulter les journaux Greengrass sur l'appareil principal pour vérifier si le périphérique client se connecte et envoie des messages avec succès. Pour de plus amples informations, veuillez consulter [AWS IoT Greengrass Journaux de surveillance](#).

5. Consultez à nouveau les journaux des composants pour vérifier que le composant reçoit des confirmations de mise à jour parallèle de la part du périphérique client Smart Light.

Linux or Unix

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

PowerShell

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

Le composant enregistre des messages pour confirmer que le périphérique client Smart Light a changé de couleur.

```
2022-07-07T03:49:24.908Z [INFO] (Copier)  
com.example.clientdevices.MySmartLightManager: stdout. Chose random color (blue)  
for MyClientDevice1.  
{scriptName=services.com.example.clientdevices.MySmartLightManager.lifecycle.Run,  
serviceName=com.example.clientdevices.MySmartLightManager, currentState=RUNNING}  
2022-07-07T03:49:24.912Z [INFO] (Copier)  
com.example.clientdevices.MySmartLightManager: stdout.  
Requested color change for MyClientDevice1 to blue.  
{scriptName=services.com.example.clientdevices.MySmartLightManager.lifecycle.Run,  
serviceName=com.example.clientdevices.MySmartLightManager, currentState=RUNNING}  
2022-07-07T03:49:24.959Z [INFO] (Copier)  
com.example.clientdevices.MySmartLightManager: stdout. Received  
shadow update confirmation from client device: MyClientDevice1.  
{scriptName=services.com.example.clientdevices.MySmartLightManager.lifecycle.Run,  
serviceName=com.example.clientdevices.MySmartLightManager, currentState=RUNNING}
```

Note

L'ombre du dispositif client est synchronisée entre le dispositif principal et le dispositif client. Cependant, le périphérique principal ne synchronise pas l'ombre de l'appareil client avec AWS IoT Core. Vous pouvez synchroniser une ombre avec AWS IoT Core pour afficher ou modifier l'état de tous les appareils de votre parc, par exemple. Pour plus d'informations sur la façon de configurer le composant Shadow Manager avec lequel synchroniser les ombres AWS IoT Core, consultez [Synchronisez les ombres de l'appareil local avec AWS IoT Core](#).

Vous avez terminé ce didacticiel. Le périphérique client se connecte au périphérique principal, envoie des messages MQTT aux composants Greengrass AWS IoT Core et reçoit des mises à jour instantanées en provenance du périphérique principal. Pour plus d'informations sur les sujets abordés dans ce didacticiel, consultez les rubriques suivantes :

- [Associer des appareils clients](#)
- [Gérez les principaux points de terminaison des appareils](#)
- [Tester les communications entre appareils clients](#)
- [API de découverte Greengrass RESTful](#)
- [Relayer les messages MQTT entre les appareils clients et AWS IoT Core](#)
- [Interagir avec les appareils clients dans les composants](#)
- [Interagissez avec les ombres de l'appareil](#)
- [Interagissez avec les ombres des appareils clients et synchronisez-les](#)

Tutoriel : Sécurisez Greengrass Nucleus avec le module Trusted Platform (TPM)

Note

Le mécanisme décrit dans ce didacticiel n'est pris en charge qu'avec [Installation AWS IoT Greengrass du logiciel Core avec provisionnement manuel des ressources](#).

Ces didacticiels contiennent des instructions sur l'utilisation de la TPM2 puce en tant que module de sécurité matériel (HSM) pour créer une clé privée et un CSR. Ce qui est habituel [Créez le certificat d'objet](#).

Ces didacticiels vous montrent comment améliorer la sécurité des appareils en configurant le logiciel AWS IoT Greengrass principal (Greengrass Nucleus) avec un module TPM (Trusted Platform Module) à l'aide de l'interface PKCS #11. Cette intégration du TPM garantit que les clés privées et les certificats utilisés pour l'identité de l'appareil et la connexion à celui-ci AWS IoT Core sont stockés en toute sécurité dans un matériel inviolable, empêchant ainsi toute extraction à des fins d'usurpation d'identité ou d'autres activités malveillantes.

Lorsque vous aurez terminé cette intégration, votre appareil principal Greengrass utilisera des clés privées protégées par TPM pour son identité et sa communication avec les services. AWS IoT

Pour plus d'informations sur la sécurité des appareils Greengrass, consultez. [Sécurité dans AWS IoT Greengrass](#)

Prérequis

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

- Un appareil compatible Linux doté du matériel TPM 2.0 ou du microprogramme TPM 2.0.
- Les instructions de ce didacticiel sont définies pour Ubuntu 24.04 LTS.
 - Toute distribution Linux capable de prendre en charge le [Linux TPM2 Software Stack](#) peut prendre en charge ce mécanisme.
- Une machine de développement AWS CLI installée et configurée avec les autorisations nécessaires pour :
 - Création et gestion AWS IoT des ressources
 - Création et gestion des rôles et politiques IAM
- Java Runtime Environment (JRE) version 8 ou ultérieure installée sur votre appareil.
- Les packages logiciels suivants sont installés sur votre appareil :
 - curl
 - jq
- Privilèges root ou sudo sur l'appareil.

Étape 1 : Installation TPM2 des outils et des dépendances

Au cours de cette étape, vous devez installer les outils TPM2 logiciels et les bibliothèques nécessaires.

1. Mettez à jour votre gestionnaire de packages et installez les TPM2 outils et les dépendances en exécutant la commande suivante.

```
sudo apt-get update && sudo apt-get install tpm2-tools \  
tpm2-abrmd \  
tpm2-tss-engine-tools \  
gnutls-bin \  
libtpm2-pkcs11-1 \  
libtpm2-pkcs11-tools \  

```

```
libtpm2-pkcs11-1-dev \  
python3-tpm2-pkcs11-tools \  
libengine-pkcs11-openssl \  
libtss2-tcti-tabrmd0
```

2. Installez les packages du fournisseur TPM2 OpenSSL sur Ubuntu 24.04 qui utilisent le moteur OpenSSL 3.

```
sudo apt-get install tpm2-openssl
```

Étape 2 : Initialisation du magasin PKCS #11 et création d'un emplacement

1. Créez un répertoire pour stocker les données.

```
sudo mkdir -p /etc/tpm2_pkcs11
```

2. Définissez l'emplacement de stockage en tant que variable d'environnement. Pour plus d'informations sur la hiérarchie des magasins, consultez [Initialisation](#).

```
export TPM2_PKCS11_STORE=/etc/tpm2_store
```

3. Initialisez le TPM2 jeton avec l'objet principal.

```
sudo tpm2_ptool init
```

Les options disponibles sont les suivantes :

hierarchy-auth HIERARCHY_AUTH

Le mot de passe d'autorisation pour ajouter un objet principal à la hiérarchie.

primary-auth PRIMARY_AUTH

Valeur d'autorisation pour l'objet clé primaire existant.

La valeur par défaut est une valeur d'authentification vide.

`primary-handle [PRIMARY_HANDLE]`

Utilisez un objet clé primaire existant.

Par défaut : `0x81000001`

`transient-parents`

Utilisez un objet principal transitoire d'un modèle donné.

Valeurs : `tpm2-tools-default`, `tpm2-tools-ecc-default`, `tss2-engine-key`

`path PATH`

Emplacement du répertoire du magasin. S'il est spécifié, le répertoire doit exister. S'il n'est pas spécifié, il effectue une recherche en examinant la variable d'environnement `TPM2_PKCS11_STORE`. Si cette variable d'environnement n'est pas définie, elle sera examinée `/etc/tpm2_pkcs11`. Si ce répertoire n'est pas trouvé ou ne peut pas être créé, il s'agit par défaut du répertoire de travail actuel.

Étape 3 : Création d'un jeton et d'une clé

1. Créez un jeton PKCS #11.

```
sudo tpm2_ptool addtoken --pid=1 --userpin=USERPIN --sopin=SOPIN --label=greengrass
```

Les options disponibles sont les suivantes :

`--pid PID`

L'identifiant d'objet principal à associer à ce jeton.

`--sopin SOPIN`

Le code PIN de l'administrateur. Cette épingle est utilisée pour la restauration d'objets.

`--userpin USERPIN`

Le code PIN de l'utilisateur. Ce code PIN est utilisé pour l'authentification lors de l'utilisation d'objets.

`--label LABEL`

Une étiquette unique permettant d'identifier le profil utilisé doit être unique.

`--hierarchy-auth HIERARCHY_AUTH`

2. Créez un objet clé ECC.

```
sudo tpm2_ptool addkey --algorithm=ecc256 --label=greengrass --userpin=***** --key-label=greenkey
```

`--label LABEL`

L'étiquette des jetons permet également d'importer la clé.

`--key-label KEY_LABEL`

L'étiquette de la clé importée. La valeur par défaut est un entier.

`--id ID`

L'identifiant de la clé. La valeur par défaut est un hexadécimal aléatoire de 8 octets.

`--attr-always-authenticate`

Définit l'attribut CKA_ALWAYS_AUTHENTICATE sur CK_TRUE.

`--hierarchy-auth HIERARCHY_AUTH`

Le hierarchyauth, obligatoire pour les objets transitoires.

`--sopin SOPIN`

Le code PIN de l'administrateur.

`--userpin USERPIN`

Le code PIN de l'utilisateur.

--algorithm

{rsa1024, rsa2048, rsa3072, rsa4096, aes128, aes256, ecc224, ecc256, ecc384, ecc521,
hmac : sha1, hmac : sha256, hmac : sha384, hmac : sha512}

Type de clé.

3. Exportez l'objet TPM2 -TSS depuis le jeton pour capturer les données d'authentification.

```
yaml_ecc0=$(sudo tpm2_ptool export -label="greengrass" -key-label="greenkey" -  
userpin="*****")
```

Exemple de sortie :

```
> echo $yaml_ecc0  
object-auth: 706c1cad8a5238871b30149705255926  
primary-object:  
  auth: ''  
  hierarchy: owner  
  is_transient: false
```

Note

Vous trouverez également un fichier nommé « greenkey.pem » qui est la clé TSS2 privée dans le répertoire où vous avez exécuté cette commande. Utilisez-le pour générer le CSR avec le fournisseur openssl tpm2. Le fichier TSS2 PRIVATE KEY est protégé par le TPM et ne peut pas être utilisé sur un autre ordinateur. Pour plus d'informations sur TSS2 les clés avec OpenSSL, [consultez Stockage de la clé privée](#) ou publique.

4. Capturez les données d'authentification pour la clé privée TSS.

```
auth_ecc0=$(echo "$yaml_ecc0" | grep "object-auth" | cut -d' ' -f2-)
```

Étape 4 : générer une demande de signature de certificat (CSR)

Au cours de cette étape, vous allez utiliser la clé privée TPM2 protégée pour générer un CSR.

1. Générez un CSR à l'aide du TPM2 fournisseur.

```
sudo openssl req -new -provider tpm2 -provider base -key greenkey.pem -passin "pass:$auth_ecc0" -out "$H"$HOSTNAME".csr"
```

Lorsque vous y êtes invité, fournissez les informations requises pour votre CSR, notamment :

- Nom du pays (code à 2 lettres)
- Nom de l'état ou de la province
- Nom de la localité
- Nom de l'organisation
- Nom de l'unité d'organisation
- Nom commun
- Adresse e-mail

Vous pouvez également fournir un fichier de configuration OpenSSL pour une génération sans assistance. Pour plus d'informations, consultez la [documentation OpenSSL](#).

2. Si vous ne générez pas le CSR sur la même machine, copiez le CSR généré sur un ordinateur sur lequel les AWS informations d'identification sont configurées.

Étape 5 : Création du certificat d'objet

Créez un certificat d' AWS IoT objet. Pour plus d'informations sur la création d'un certificat d'objet, consultez [Créez le certificat d'objet](#).

Étape 6 : Importer un certificat d'objet dans le TPM

1. Copiez le certificat d'objet sur l'appareil.
2. Ajoutez le certificat d'objet au jeton Greengrass.

```
sudo tpm2_ptool addcert --label=greengrass --key-label=greenkey device.pem.crt
```

Les options disponibles sont les suivantes :

```
--help
```

Afficher ce message d'aide et quitter.

```
--label LABEL
```

L'étiquette de profil à supprimer.

```
--key-label KEY_LABEL
```

L'étiquette de clé privée associée.

```
--key-id KEY_ID
```

L'identifiant de clé privée associé est en hexadécimal.

cert

Le certificat PEM x509 à ajouter.

Étape 7 : Capture de l'URL de l'objet PKCS #11

Nous utiliserons le `gnutls-bin` package `p11tool` fourni pour obtenir l'URL et l'objet URLs du jeton PKCS #11.

- ## 1. Capturez l'URL du jeton Greengrass.

```
TOKEN=sudo p11tool --list-token-urls | grep "token=greengrass"
```

2. Obtenez l'objet du URLs jeton Greengrass. Utilisez le même code PIN que celui utilisé à l'étape 3.

```
sudo p11tool --login --list-all "${TOKEN}"
```

Exemple de sortie :

```
Token 'greengrass' with URL
'pkcs11:model=SLB9672%00%00%00%00%00%00%00%00;manufacturer=Infineon;serial=0000000000000000
requires user PIN
Enter PIN:
WARNING: Needed CKA_VALUE but didn't find encrypted blob
Object 0:
```



```

URL:
pkcs11:model=SLB9672%00%00%00%00%00%00%00%00;manufacturer=Infineon;serial=0000000000000000
%39%32%37%65%61%64%61%39%31%32%35%31%61%35%37%31;object=greenkey;type=private
Type: Private key (EC/ECDSA-SECP256R1)
Label: greenkey
Flags: CKA_PRIVATE; CKA_NEVER_EXTRACTABLE; CKA_SENSITIVE;
ID: 39:32:37:65:61:64:61:39:31:32:35:31:61:35:37:31

Object 1:
URL:
pkcs11:model=SLB9672%00%00%00%00%00%00%00%00;manufacturer=Infineon;serial=0000000000000000
%39%32%37%65%61%64%61%39%31%32%35%31%61%35%37%31;object=greenkey;type=public
Type: Public key (EC/ECDSA-SECP256R1)
Label: greenkey
ID: 39:32:37:65:61:64:61:39:31:32:35:31:61:35:37:31

Object 2:
URL:
pkcs11:model=SLB9672%00%00%00%00%00%00%00%00;manufacturer=Infineon;serial=0000000000000000
%39%32%37%65%61%64%61%39%31%32%35%31%61%35%37%31;object=greenkey;type=cert
Type: X.509 Certificate (EC/ECDSA-SECP256R1)
Expires: Fri Dec 31 18:59:59 2049
Label: greenkey
ID: 39:32:37:65:61:64:61:39:31:32:35:31:61:35:37:31

```

- ### 3. Capturez l'URL de l'objet pour la clé privée et le certificat.

Étape 8 : Configuration et installation de Greengrass avec assistance TPM2

1. Configurez le certificat d'objet. Pour plus d'informations, voir [Configurer le certificat d'objet](#).
2. Suivez les instructions pour installer le logiciel AWS IoT Greengrass principal avec la clé privée et le certificat dans un port HSM. [Installation du logiciel AWS IoT Greengrass Core](#) Suivez ensuite les étapes suivantes pour configurer votre installation afin de tirer parti TPM2 de l'interface PKCS #11.
3. Vérifiez que le composant du fournisseur PKCS #11 a été téléchargé et enregistré dans l'emplacement de votre programme d'installation de Greengrass.
4. Utilisez un éditeur de texte pour créer un fichier de configuration nommé `config.yaml` à fournir au programme d'installation. Par exemple, sur un système basé sur Linux, vous pouvez exécuter la commande suivante pour utiliser GNU nano pour créer le fichier.

```
nano GreengrassInstaller/config.yaml
```

5. Copiez le contenu YAML suivant dans le fichier. Ce fichier de configuration partiel spécifie les paramètres système, les paramètres du noyau Greengrass et les paramètres du fournisseur PKCS #11.

```
---
system:
  certificateFilePath: "pkcs11:model=SW%20%20%20TPM
%00%00%00%00%00%00%00%00;manufacturer=IBM;serial=0000000000000000;token=greengrass;id=
%34%35;object=greenkey;type=cert"
  privateKeyPath: "pkcs11:model=SW%20%20%20TPM
%00%00%00%00%00%00%00%00;manufacturer=IBM;serial=0000000000000000;token=greengrass;id=
%34%35;object=greenkey;type=private"
  rootCaPath: "/greengrass/v2/AmazonRootCA1.pem"
  rootpath: "/greengrass/v2"
  thingName: "myThing"
services:
  aws.greengrass.Nucleus:
    componentType: "NUCLEUS"
    version: "2.14.0"
    configuration:
      awsRegion: "us-east-1"
      iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
      iotDataEndpoint: "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
      iotCredEndpoint: "device-credentials-prefix.credentials.iot.us-
west-2.amazonaws.com"
  aws.greengrass.crypto.Pkcs11Provider:
    configuration:
      name: "tpm2_pkcs11"
      library: "/usr/lib/x86_64-linux-gnu/pkcs11/libtpm2_pkcs11.so"
      slot: 1
      userPin: "123456"
```

6. Modifiez le fichier avec les paramètres spécifiques à votre installation.
- Mettez à jour `certificateFilePath` et `privateKeyPath` avec la mise à jour `certificateFilePath` et `privateKeyPath` avec l'URL PKCS #11 capturée à l'étape 7.
 - Mettez à jour votre `iotDataEndpoint` annonce en `iotCredEndpoint` fonction de votre AWS IoT point de terminaison.
 - Dans la `aws.greengrass.crypto.Pkcs11Provider` configuration, mettez à jour la bibliothèque en fonction de votre plateforme.

 Note

L'exemple présenté concerne X86_64. Le chemin du fichier sera similaire pour l'ARM64 appareil.

7. Effectuez les étapes d'installation de Greengrass dans. [Installation du logiciel AWS IoT Greengrass Core](#)

Étape 9 : vérifier l'installation

Au cours de cette étape, vous allez vérifier que Greengrass fonctionne correctement grâce TPM2 à l'intégration.

1. Vérifiez l'état du service Greengrass.

```
sudo systemctl status greengrass.service
```

2. Consultez les journaux de Greengrass pour vous assurer qu'il n'y a aucune erreur.

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

3. Vérifiez que votre appareil apparaît comme étant connecté dans la [AWS IoT console](#).
- Connectez-vous à la [console AWS IoT Greengrass](#).
 - Sous Gérer, développez les appareils Greengrass et choisissez les appareils Core.
 - Vérifiez que votre appareil est connecté. L'état de l'appareil indiquera HEALTHY s'il est connecté. Pour de plus amples informations, veuillez consulter [Vérifiez l'état de l'appareil Greengrass Core](#).

Résolution de problème

Si vous rencontrez des problèmes lors de la configuration ou du fonctionnement de votre appareil Greengrass TPM2 compatible, essayez les étapes de dépannage suivantes.

- Consultez le fichier journal principal de Greengrass.

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

- Vérifiez la configuration du fournisseur PKCS #11.

```
sudo cat /greengrass/v2/config/effectiveConfig.yaml
```

- Assurez-vous que le TPM2 service fonctionne.

```
sudo systemctl status tpm2-abrmd.service
```

- Vérifiez que la TPM2 clé est accessible.

```
sudo pkcs11-tool -module /usr/lib/x86_64-linux-gnu/pkcs11/libtpm2_pkcs11.so -l -p  
123456 -list-objects
```

- Si votre système d'exploitation est configuré avec un chiffrement complet du disque avec des clés racine de TPM2 stockage telles que Clevis ou systemd-cryptenroll, vérifiez que vous n'utilisez pas le même identifiant persistant que celui utilisé par ces outils. L'utilisation du même identifiant persistant peut affecter le mécanisme de chiffrement de votre disque. Pour vérifier tous les descripteurs persistants créés et utilisés, exécutez la commande suivante

```
sudo tpm2_getcap handles-persistent
```

Étapes suivantes

Maintenant que vous avez intégré avec succès votre appareil principal Greengrass à TPM2, vous pouvez :

- Déployez des composants sur votre appareil Greengrass sécurisé. Pour de plus amples informations, consultez [Déployer AWS IoT Greengrass des composants sur des appareils](#).
- Configurez des appareils Greengrass supplémentaires grâce TPM2 à l'intégration.

Pour plus d'informations sur la sécurité des appareils Greengrass, consultez. [Sécurité dans AWS IoT Greengrass](#)

Tutoriel : Sécurisez AWS IoT Greengrass Nucleus Lite avec le module Trusted Platform (TPM)

Ce didacticiel explique comment activer et configurer le support du Trusted Platform Module (TPM) pour AWS IoT Greengrass Nucleus Lite. Le TPM fournit une racine de confiance basée sur le matériel pour un stockage sécurisé des clés. Cette fonctionnalité de sécurité protège les opérations cryptographiques et les informations d'identification sensibles, améliorant ainsi la sécurité et l'intégrité de l'appareil.

Lorsque vous aurez terminé cette intégration, votre appareil AWS IoT Greengrass principal utilisera des clés privées protégées par le TPM pour son identité et sa communication avec AWS IoT les services.

Pour plus d'informations sur la sécurité AWS IoT Greengrass des appareils, consultez [Sécurité dans AWS IoT Greengrass](#).

Important

Ce mécanisme n'est pris en charge que lors de l'installation de AWS IoT Greengrass Nucleus Lite avec un provisionnement manuel des ressources.

Prérequis

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

- [Un appareil compatible Linux avec du matériel TPM 2.0 ou NitroTPM](#)
- Une machine de développement sur laquelle Greengrass Nucleus Lite est installé. Pour plus d'informations, voir [Installer le logiciel AWS IoT Greengrass principal \(console\)](#).
- Les instructions de ce didacticiel sont définies pour Ubuntu 24.04 LTS.
- Toute distribution Linux capable de prendre en charge le [Linux TPM2 Software Stack](#) peut prendre en charge ce mécanisme.
- Une machine de développement [AWS CLI](#) installée et configurée avec les autorisations nécessaires pour :
 - Création et gestion AWS IoT des ressources
 - Création et gestion des rôles et des politiques IAM
- Privilèges root ou sudo sur l'appareil.

Ce didacticiel contient des instructions sur la façon d'utiliser la TPM2 puce en tant que module de sécurité matériel (HSM) pour créer une clé privée et un CSR qui sont utilisés pour créer le certificat d'AWS IoT objet.

Étape 1 : configurer une instance NitroTPM

1. Configurez une instance NitroTPM. Pour plus d'informations, consultez [NitroTPM](#).
2. Lancez l'instance finale à l'aide de l'AMI personnalisée créée à l'étape précédente.

Important

Lorsque vous vous connectez via SSH, utilisez l'utilisateur `ubuntu` au lieu de `root`.

3. Vérifiez que le périphérique TPM est présent et fonctionne en exécutant la commande suivante :

```
ls -la /dev/tpm*
```

Vous devriez voir `/dev/tpm0` et `/dev/tpmrm0` appareils.

Étape 2 : Installation et configuration des outils TPM

1. Installez les packages requis en exécutant la commande suivante :

```
sudo apt update
sudo apt install tpm2-openssl tpm2-tools tpm2-abrmd libtss2-tcti-tabrmd0
```

2. Vérifiez les autorisations du dispositif TPM en exécutant la commande suivante :

```
ls -l /dev/tpm0      # Should be owned by tss:root with permissions 0660
ls -l /dev/tpmrm0   # Should be owned by tss:tss with permissions 0660
```

Étape 3 : Configuration du fournisseur OpenSSL TPM2

1. Modifiez le fichier de configuration OpenSSL :

```
sudo vi /etc/ssl/openssl.cnf
```

2. Ajoutez la configuration suivante :

```
[openssl_init]
providers = provider_sect

[provider_sect]
default = default_sect
tpm2 = tpm2_sect

[default_sect]
activate = 1

[tpm2_sect]
identity = tpm2
module = /usr/local/lib64/tpm2.so
activate = 1
```

3. Ajustez le chemin du module si nécessaire. Vous pouvez trouver le bon chemin en utilisant :

```
find /usr -name "tpm2.so"
```

Étape 4 : générer des clés TPM persistantes

1. Créez une clé primaire en exécutant la commande suivante :

```
sudo tpm2_createprimary -C o -c primary.ctx
```

2. Créez un objet clé ECC en exécutant la commande suivante :

```
sudo tpm2_create -C primary.ctx -g sha256 -G ecc256 -r device.priv -u device.pub
```

3. Chargez la clé en exécutant la commande suivante :

```
sudo tpm2_load -C primary.ctx -r device.priv -u device.pub -c device.ctx
```

4. Rendez la clé persistante en exécutant la commande suivante :

```
sudo tpm2_evictcontrol -C o -c device.ctx 0x81000002
```

Cela crée une clé persistante avec le handle (like 0x81000002).

Étape 5 : générer une demande de signature de certificat (CSR)

Au cours de cette étape, vous allez utiliser la clé privée TPM2 protégée pour générer une demande de signature de certificat (CSR).

1. Générez un CSR à l'aide de la clé TPM :

```
openssl req -new -provider tpm2 -key "handle:0x81000002" \  
  -out device.csr \  
  -subj "/CN=TPMThing"
```

2. 0x81000002 Remplacez-le par la valeur de handle que vous avez choisie et TPMThing par le nom d'objet de votre choix.

Étape 6 : Création du certificat à partir du CSR

1. Sur votre ordinateur de développement, créez un dossier dans lequel vous avez téléchargé le certificat AWS IoT correspondant.

```
mkdir greengrass-v2-certs
```

2. Utilisez le fichier CSR pour créer et télécharger le certificat correspondant à l' AWS IoT objet sur votre ordinateur de développement.

```
aws iot create-certificate-from-csr \  
  --set-as-active \  
  --certificate-signing-request file://path_to_device.csr \  
  --certificate-pem-outfile greengrass-v2-certs/device.pem.crt
```

Si la demande aboutit, la réponse ressemble à l'exemple suivant :

```
{  
  "certificateArn": "arn:aws:iot:us-west-2:123456789012:cert/  
aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4",  
  "certificateId":  
  "aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4",  
  "certificatePem": "-----BEGIN CERTIFICATE-----  
MIICiTCCAFICQD6m7oRw0uX0jANBgkqhkiG9w  
0BAQUFADCBIDELMAkGA1UEBhMCVVMxCzAJBgNVBAGTA1dBMRAwDgYDVQQHEwdTZ  
WF0dGx1MQ8wDQYDVQQKEwZBbWF6b24xFDASBgNVBA5TC0lBTSBDb25zb2x1MRIw
```



```

EAYDVQQDEw1UZsXN0Q21sYWMxHzAdBgkqhkiG9w0BCQEWEG5vb251QGFTYXpvi5
jb20wHhcNMTEwNDI1MjA0NTIxWhcnMTIwNDI0MjA0NTIxWjCBiDELMAkGA1UEBh
MCVVMxCzAJBgNVBAGTAldBMRAwDgYDVQQHEwdTZWF0dGx1MQ8wDQYDVQQKEwZBb
WF6b24xFDASBgNVBASTC0lBTsBDb25zb2x1MRIwEAYDVQQDEw1UZsXN0Q21sYWMx
HzAdBgkqhkiG9w0BCQEWEG5vb251QGFTYXpvi5jb20wgZ8wDQYJKoZIhvcNAQEE
BBQADgY0AMIGJAoGBAMaK0dn+a4GmWIWJ21uUSfwfEvySWtC2XADZ4nB+BLygVI
k60CpiwsZ3G93vUEI03IyNoH/f0wYK8m9TrDHudUZg3qX4waLG5M43q7Wgc/MbQ
ITx0USQv7c7ugFFDzQGBzZswY6786m86gpEIbb30hjZnzcqvQAaRHhd1QWIMm2nr
AgMBAAEwDQYJKoZIhvcNAQEFBQADgYEAtCu4nUhVVxYUntneD9+h8Mg9q6q+auN
KyExzyLwax1Aoo7TJHidbtS4J5iNmZgXL0FkbFFBjvSfpJI1J00zbhNYS5f6Guo
EDmFJl0ZxBHjJnyp3780D8uTs7fLvJx79LjSTbNYiytVbZPQUQ5Yaxu2jXnimvw
3rrszlaEXAMPLE=
-----END CERTIFICATE-----"
}

```

Étape 7 : Configuration de Greengrass Nucleus Lite avec le support TPM

Pour activer la prise en charge du TPM dans Greengrass Nucleus Lite, apportez les modifications suivantes :

1. Configurez les autorisations utilisateur en ajoutant l'ggcoreutilisateur au tss groupe pour l'accès au TPM :

```
sudo usermod -a -G tss ggcore
```

2. Mettez à jour le répertoire des informations d'identification en effectuant les étapes suivantes :
 - Supprimez le fichier de clé privée du répertoire des informations d'identification.
 - Comme nous utilisons des clés TPM persistantes, aucun fichier de clé privée n'a besoin d'être copié.
 - Copiez le nouveau device.pem.crt dans ce répertoire d'informations d'identification.
3. Modifiez config.yaml avec la configuration spécifique au TPM suivante :

```

system:
  privateKeyPath: "handle:0x81000002" # Use your chosen handle
  certificateFilePath: "" # Replace with the path of device.pem.crt
  ...

```

4. Redémarrez votre Greengrass Nucleus Lite en exécutant la commande suivante :

```
systemctl restart greengrass-lite.target
```

Résolution de problème

Si vous rencontrez des problèmes lors de la configuration ou du fonctionnement de votre AWS IoT Greengrass appareil TPM2 compatible, essayez les étapes de résolution des problèmes suivantes :

Le périphérique TPM est introuvable (lors de l'utilisation de NitroTPM)

Si `/dev/tpm0` ce n'est pas le cas, procédez comme suit :

1. Vérifiez que vous utilisez un type d'instance compatible avec NitroTPM.
2. Assurez-vous que l'AMI a été créée avec `--tpm-support v2.0`.
3. Vérifiez que l'instance a été lancée à partir de l'AMI personnalisée.

Erreurs de type « Autorisation refusée »

Si vous rencontrez des erreurs d'accès au TPM, procédez comme suit :

1. Vérifiez que l'utilisateur fait partie du `tss` groupe : `groups $USER`
2. Vérifiez les autorisations de l'appareil TPM à l'aide de la commande suivante :

```
ls -l /dev/tpm*
```

3. Vérifiez que vous vous êtes déconnecté et que vous vous êtes reconnecté après avoir été ajouté au `tss` groupe.

Problèmes avec le fournisseur OpenSSL

Si le fournisseur TPM n'est pas trouvé, procédez comme suit :

1. Vérifiez `tpm2.so` le chemin dans `le/etc/ssl/openssl.cnf`.
2. Vérifiez l'installation du fournisseur en exécutant la commande suivante :

```
openssl list -providers
```

3. Vérifiez que le `tpm2-openssl` package est correctement installé.

Étapes suivantes

Maintenant que vous avez intégré avec succès votre appareil AWS IoT Greengrass principal à TPM2, vous pouvez :

- Déployez des composants sur votre AWS IoT Greengrass appareil sécurisé
- Configurez des AWS IoT Greengrass appareils supplémentaires grâce à TPM2 l'intégration

Pour plus d'informations sur la sécurité AWS IoT Greengrass des appareils, consultez [Sécurité dans AWS IoT Greengrass](#).

Tutoriel : Démarrez avec SageMaker AI Edge Manager

Important

SageMaker AI Edge Manager a été abandonné le 26 avril 2024. Pour plus d'informations sur la poursuite du déploiement de vos modèles sur des appareils Edge, consultez [SageMaker AI Edge Manager end of life](#).

Amazon SageMaker AI Edge Manager est un agent logiciel qui s'exécute sur des appareils périphériques. SageMaker AI Edge Manager assure la gestion des modèles pour les appareils de pointe afin que vous puissiez emballer et utiliser les modèles compilés par Amazon SageMaker AI Neo directement sur les appareils principaux de Greengrass. En utilisant SageMaker AI Edge Manager, vous pouvez également échantillonner les données d'entrée et de sortie du modèle à partir de vos principaux appareils, et envoyer ces données à des AWS Cloud fins de surveillance et d'analyse. Pour plus d'informations sur le fonctionnement d' SageMaker AI Edge Manager sur les appareils principaux de Greengrass, consultez. [Utiliser Amazon SageMaker AI Edge Manager sur les appareils principaux de Greengrass](#)

Ce didacticiel vous explique comment commencer à utiliser SageMaker AI Edge Manager avec des exemples de composants AWS fournis sur un périphérique principal existant. Ces exemples de composants utilisent le composant SageMaker AI Edge Manager comme dépendance pour déployer l'agent Edge Manager et effectuer des inférences à l'aide de modèles préentraînés compilés à l'aide d' SageMaker AI Neo. Pour plus d'informations sur l'agent SageMaker AI Edge Manager, consultez [SageMaker AI Edge Manager](#) dans le manuel Amazon SageMaker AI Developer Guide.

Pour configurer et utiliser l'agent SageMaker AI Edge Manager sur un appareil principal Greengrass existant, AWS fournit un exemple de code que vous pouvez utiliser pour créer les exemples d'inférence et de composants de modèle suivants.

- Classification des images
 - `com.greengrass.SageMakerEdgeManager.ImageClassification`
 - `com.greengrass.SageMakerEdgeManager.ImageClassification.Model`
- Détection d'objets
 - `com.greengrass.SageMakerEdgeManager.ObjectDetection`
 - `com.greengrass.SageMakerEdgeManager.ObjectDetection.Model`

Ce didacticiel explique comment déployer les exemples de composants et l'agent SageMaker AI Edge Manager.

Rubriques

- [Conditions préalables](#)
- [Configurez votre appareil principal Greengrass dans SageMaker AI Edge Manager](#)
- [Création des exemples de composants](#)
- [Exécuter un exemple d'inférence de classification d'images](#)

Conditions préalables

Pour suivre ce didacticiel, vous devez remplir les conditions préalables suivantes :

- Un appareil Greengrass principal fonctionnant sous Amazon Linux 2, une plate-forme Linux basée sur Debian (x86_64 ou Armv8) ou Windows (x86_64). Si vous n'en avez pas, veuillez consulter [Tutoriel : Débuter avec AWS IoT Greengrass V2](#).
- [Python](#) 3.6 ou version ultérieure, y compris `pip` pour votre version de Python, installé sur votre appareil principal.
- L'environnement d'exécution GLX de l'API OpenGL `libgl1-mesa-glx ()` est installé sur votre appareil principal.
- Un utilisateur Gestion des identités et des accès AWS (IAM) doté d'autorisations d'administrateur.
- Un ordinateur de développement de type Windows, Mac ou Unix connecté à Internet qui répond aux exigences suivantes :

- [Python](#) 3.6 ou version ultérieure installé.
- AWS CLI installé et configuré avec vos informations d'identification d'administrateur IAM. Pour plus d'informations, voir [Installation AWS CLI](#) et [configuration du AWS CLI](#).
- Les compartiments S3 suivants ont été créés à l'identique Compte AWS et en même temps Région AWS que votre appareil principal Greengrass :
 - Un compartiment S3 pour stocker les artefacts inclus dans l'inférence d'échantillon et les composants du modèle. Ce didacticiel utilise `amzn-s3-demo-bucket1` pour faire référence à ce compartiment.
 - Un compartiment S3 que vous associez à votre parc d'appareils SageMaker AI Edge. SageMaker AI Edge Manager nécessite un compartiment S3 pour créer le parc d'appareils Edge et pour stocker des exemples de données provenant de l'exécution d'inférences sur votre appareil. Ce didacticiel utilise `amzn-s3-demo-bucket2` pour faire référence à ce compartiment.

Pour plus d'informations sur la création de compartiments S3, consultez [Getting started with Amazon S3](#).

- Le [rôle d'appareil Greengrass](#) est configuré comme suit :
 - Une relation de confiance qui permet `credentials.iot.amazonaws.com` et permet `sagemaker.amazonaws.com` d'assumer le rôle, comme le montre l'exemple de politique IAM suivant.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "credentials.iot.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    },
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "sagemaker.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

```
}
```

- La politique gérée par [AmazonSageMakerEdgeDeviceFleetPolicy](#) IAM.
- La politique gérée par [AmazonSageMakerFullAccess](#) IAM.
- `s3:GetObject` pour le compartiment S3 qui contient les artefacts de vos composants, comme illustré dans l'exemple de politique IAM suivant.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "s3:GetObject"
      ],
      "Resource": [
        "arn:aws:s3:::amzn-s3-demo-bucket1/*"
      ],
      "Effect": "Allow"
    }
  ]
}
```

Configurez votre appareil principal Greengrass dans SageMaker AI Edge Manager

Les flottes d'appareils Edge dans SageMaker AI Edge Manager sont des ensembles d'appareils regroupés de manière logique. Pour utiliser SageMaker AI Edge Manager avec AWS IoT Greengrass, vous devez créer un parc d'appareils Edge utilisant le même alias de AWS IoT rôle que l'appareil principal Greengrass sur lequel vous déployez l'agent SageMaker AI Edge Manager. Ensuite, vous devez enregistrer l'appareil principal dans le cadre de cette flotte.

Rubriques

- [Créez un parc d'appareils de pointe](#)
- [Enregistrez votre appareil Greengrass Core](#)

Créez un parc d'appareils de pointe

Pour créer un parc d'appareils Edge (console)

1. Dans la [console Amazon SageMaker AI](#), choisissez Edge Manager, puis choisissez Edge devices fleets.
2. Sur la page Flottes d'appareils, choisissez Créer un parc d'appareils.
3. Sous Propriétés du parc d'appareils, procédez comme suit :
 - Dans Nom du parc d'appareils, entrez le nom de votre parc d'appareils.
 - Pour le rôle IAM, entrez le nom de ressource Amazon (ARN) de l'alias de AWS IoT rôle que vous avez spécifié lors de la configuration de votre appareil principal Greengrass.
 - Désactivez le bouton Créer un alias de rôle IAM.
4. Choisissez Suivant.
5. Sous Configuration de sortie, pour l'URI du compartiment S3, entrez l'URI du compartiment S3 que vous souhaitez associer au parc d'appareils.
6. Sélectionnez Soumettre.

Enregistrez votre appareil Greengrass Core

Pour enregistrer votre appareil Greengrass Core en tant qu'appareil périphérique (console)

1. Dans la [console Amazon SageMaker AI](#), choisissez Edge Manager, puis choisissez Edge devices.
2. Sur la page Appareils, choisissez Enregistrer les appareils.
3. Sous Propriétés de l'appareil, pour Nom du parc d'appareils, entrez le nom du parc d'appareils que vous avez créé, puis choisissez Suivant.
4. Choisissez Suivant.
5. Sous Source de l'appareil, dans Nom de l'appareil, entrez le nom de l' AWS IoT objet de votre appareil Greengrass principal.
6. Sélectionnez Soumettre.

Création des exemples de composants

Pour vous aider à commencer à utiliser le composant SageMaker AI Edge Manager, AWS fournit un script Python GitHub qui crée l'exemple d'inférence et les composants du modèle et les télécharge sur le site AWS Cloud pour vous. Effectuez les étapes suivantes sur un ordinateur de développement.

Pour créer les exemples de composants

1. Téléchargez le référentiel [d'exemples de AWS IoT Greengrass composants](#) sur GitHub votre ordinateur de développement.
2. Accédez au `/machine-learning/sagemaker-edge-manager` dossier téléchargé.

```
cd download-directory/machine-learning/sagemaker-edge-manager
```

3. Exécutez la commande suivante pour créer et télécharger les exemples de composants dans le AWS Cloud.

```
python3 create_components.py -r region -b amzn-s3-demo-bucket
```

Remplacez *region* par l' Région AWS endroit où vous avez créé votre périphérique principal Greengrass, et remplacez `amzn-s3-demo-bucket1` par le nom du compartiment S3 dans lequel seront stockés les artefacts de vos composants.

Note

Par défaut, le script crée des exemples de composants pour la classification des images et l'inférence de détection d'objets. Pour créer des composants uniquement pour un type d'inférence spécifique, spécifiez l' `-i ImageClassification / ObjectDetection` argument.

Les exemples d'inférence et les composants du modèle à utiliser avec SageMaker AI Edge Manager sont désormais créés dans votre Compte AWS. Pour afficher les exemples de composants dans la [AWS IoT Greengrass console](#), choisissez Composants, puis sous Mes composants, recherchez les composants suivants :

- `com.greengrass.SageMakerEdgeManager.ImageClassification`

- `com.greengrass.SageMakerEdgeManager.ImageClassification.Model`
- `com.greengrass.SageMakerEdgeManager.ObjectDetection`
- `com.greengrass.SageMakerEdgeManager.ObjectDetection.Model`

Exécuter un exemple d'inférence de classification d'images

Pour exécuter l'inférence de classification d'images à l'aide des exemples de composants AWS fournis et de l'agent SageMaker AI Edge Manager, vous devez déployer ces composants sur votre appareil principal. Le déploiement de ces composants télécharge un modèle Resnet-50 préentraîné compilé par SageMaker AI Neo et installe l'agent SageMaker AI Edge Manager sur votre appareil. L'agent SageMaker AI Edge Manager charge le modèle et publie les résultats d'inférence sur le `gg/sageMakerEdgeManager/image-classification` sujet. Pour consulter ces résultats d'inférence, utilisez le client AWS IoT MQTT de la AWS IoT console pour vous abonner à cette rubrique.

Rubriques

- [Abonnez-vous à la rubrique des notifications](#)
- [Déployer les exemples de composants](#)
- [Afficher les résultats d'inférence](#)

Abonnez-vous à la rubrique des notifications

Au cours de cette étape, vous configurez le client AWS IoT MQTT dans la AWS IoT console pour qu'il regarde les messages MQTT publiés par le composant d'inférence d'exemple. Par défaut, le composant publie les résultats d'inférence sur le `gg/sageMakerEdgeManager/image-classification` sujet. Abonnez-vous à cette rubrique avant de déployer le composant sur votre appareil principal Greengrass pour voir les résultats de l'inférence lorsque le composant s'exécute pour la première fois.

Pour vous abonner à la rubrique de notifications par défaut

1. Dans le menu de navigation de la [AWS IoT console](#), choisissez Test, client de test MQTT.
2. Sous S'abonner à un sujet, dans le champ Nom du sujet, entrez **`gg/sageMakerEdgeManager/image-classification`**.
3. Choisissez Abonner.

Déployer les exemples de composants

Au cours de cette étape, vous configurez et déployez les composants suivants sur votre appareil principal :

- `aws.greengrass.SageMakerEdgeManager`
- `com.greengrass.SageMakerEdgeManager.ImageClassification`
- `com.greengrass.SageMakerEdgeManager.ImageClassification.Model`

Pour déployer vos composants (console)

1. Dans le menu de navigation de la [AWS IoT Greengrass console](#), choisissez Déploiements, puis choisissez le déploiement que vous souhaitez modifier pour votre appareil cible.
2. Sur la page de déploiement, choisissez Revise, puis Revise le déploiement.
3. Sur la page Spécifier la cible, choisissez Next.
4. Sur la page Sélectionner les composants, procédez comme suit :
 - a. Sous Mes composants, sélectionnez les composants suivants :
 - `com.greengrass.SageMakerEdgeManager.ImageClassification`
 - `com.greengrass.SageMakerEdgeManager.ImageClassification.Model`
 - b. Sous Composants publics, désactivez le bouton Afficher uniquement les composants sélectionnés, puis sélectionnez le `aws.greengrass.SageMakerEdgeManager` composant.
 - c. Choisissez Suivant.
5. Sur la page Configurer les composants, sélectionnez le `aws.greengrass.SageMakerEdgeManager` composant et procédez comme suit.
 - a. Choisissez Configure component (Configurer un composant).
 - b. Sous Configuration update (Mise à jour de la configuration), dans Configuration to merge (Configuration à fusionner), saisissez la configuration suivante.

```
{
  "DeviceFleetName": "device-fleet-name",
  "BucketName": "amzn-s3-demo-bucket"
}
```

Remplacez-le *device-fleet-name* par le nom du parc d'appareils Edge que vous avez créé et remplacez par *amzn-s3-demo-bucket* le nom du compartiment S3 associé à votre parc d'appareils.

- c. Choisissez Confirm (Confirmer), puis Next (Suivant).
6. Sur la page Configure advanced settings (Configurer les paramètres avancés), conservez les paramètres de configuration par défaut et choisissez Next (Suivant).
7. Sur la page de révision, choisissez Déployer

Pour déployer vos composants (AWS CLI)

1. Sur votre ordinateur de développement, créez un `deployment.json` fichier pour définir la configuration de déploiement de vos composants SageMaker AI Edge Manager. Ce fichier doit ressembler à l'exemple suivant.

```
{
  "targetArn": "targetArn",
  "components": {
    "aws.greengrass.SageMakerEdgeManager": {
      "componentVersion": "1.0.x",
      "configurationUpdate": {
        "merge": "{ \"DeviceFleetName\": \"device-fleet-name\", \"BucketName\": \"amzn-s3-demo-bucket2\" }"
      }
    },
    "com.greengrass.SageMakerEdgeManager.ImageClassification": {
      "componentVersion": "1.0.x",
      "configurationUpdate": {
      }
    },
    "com.greengrass.SageMakerEdgeManager.ImageClassification.Model": {
      "componentVersion": "1.0.x",
      "configurationUpdate": {
      }
    }
  }
}
```

- Dans le champ `targetArn`, remplacez *targetArn* par l'Amazon Resource Name (ARN) de l'objet ou du groupe d'objets à cibler pour le déploiement, au format suivant :

- Objet : `arn:aws:iot:region:account-id:thing/thingName`
- Groupe d'objets : `arn:aws:iot:region:account-id:thinggroup/thingGroupName`
- Dans le merge champ, remplacez `device-fleet-name` par le nom du parc d'appareils Edge que vous avez créé. Remplacez-le ensuite `amzn-s3-demo-bucket2` par le nom du compartiment S3 associé à votre parc d'appareils.
- Remplacez les versions de composant de chaque composant par la dernière version disponible.

2. Exécutez la commande suivante pour déployer les composants sur le périphérique :

```
aws greengrassv2 create-deployment \  
  --cli-input-json file://path/to/deployment.json
```

L'exécution du déploiement peut prendre plusieurs minutes. À l'étape suivante, vérifiez le journal des composants pour vous assurer que le déploiement s'est terminé avec succès et afficher les résultats des inférences.

Afficher les résultats d'inférence

Après avoir déployé les composants, vous pouvez consulter les résultats de l'inférence dans le journal des composants sur votre périphérique principal Greengrass et dans AWS IoT le client MQTT de la console. AWS IoT Pour vous abonner à la rubrique sur laquelle le composant publie les résultats d'inférence, consultez [Abonnez-vous à la rubrique des notifications](#).

- AWS IoT Client MQTT : pour afficher les résultats publiés par le composant d'inférence dans la [rubrique des notifications par défaut](#), procédez comme suit :
 1. Dans le menu de navigation de la [AWS IoT console](#), choisissez Test, client de test MQTT.
 2. Sous Abonnements, sélectionnez **gg/sageMakerEdgeManager/image-classification**.
- Journal des composants : pour afficher les résultats de l'inférence dans le journal des composants, exécutez la commande suivante sur votre appareil principal Greengrass.

```
sudo tail -f /greengrass/v2/logs/  
com.greengrass.SageMakerEdgeManager.ImageClassification.log
```

Si vous ne pouvez pas voir les résultats d'inférence dans le journal des composants ou dans le client MQTT, cela signifie que le déploiement a échoué ou n'a pas atteint le périphérique principal. Cela peut se produire si votre appareil principal n'est pas connecté à Internet ou ne dispose pas des autorisations nécessaires pour exécuter le composant. Exécutez la commande suivante sur votre appareil principal pour afficher le fichier journal du logiciel AWS IoT Greengrass Core. Ce fichier inclut les journaux du service de déploiement de l'appareil principal Greengrass.

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

Pour de plus amples informations, veuillez consulter [Résolution des problèmes liés à l'inférence par apprentissage automatique](#).

Tutoriel : Effectuer une inférence de classification d'images d'échantillons à l'aide TensorFlow de Lite

Ce didacticiel explique comment utiliser le composant d'inférence de [classification d'images TensorFlow Lite](#) pour effectuer une inférence de classification d'images par exemple sur un appareil Greengrass Core. Ce composant inclut les dépendances suivantes :

- TensorFlow Composant de magasin de modèles de classification d'images Lite
- TensorFlow Composant d'exécution Lite

Lorsque vous déployez ce composant, il télécharge un modèle MobileNet v1 préentraîné et installe le runtime [TensorFlow Lite](#) et ses dépendances. Ce composant publie les résultats d'inférence sur le `ml/tflite/image-classification` sujet. Pour consulter ces résultats d'inférence, utilisez le client AWS IoT MQTT de la AWS IoT console pour vous abonner à cette rubrique.

Dans ce didacticiel, vous déployez le composant d'inférence d'échantillons pour effectuer une classification d'image sur l'exemple d'image fourni par AWS IoT Greengrass. Après avoir terminé ce didacticiel, vous pouvez le terminer [Tutoriel : effectuer une inférence de classification d'images d'échantillons sur des images provenant d'un appareil photo à l'aide TensorFlow de Lite](#), qui vous montre comment modifier le composant d'inférence d'échantillons pour effectuer une classification d'images sur des images provenant d'une caméra localement sur un périphérique principal Greengrass.

Pour plus d'informations sur l'apprentissage automatique sur les appareils Greengrass, consultez. [Exécuter l'inférence de Machine Learning](#)

Rubriques

- [Prérequis](#)
- [Étape 1 : Abonnez-vous à la rubrique de notifications par défaut](#)
- [Étape 2 : Déploiement du composant de classification d'images TensorFlow Lite](#)
- [Étape 3 : Afficher les résultats de l'inférence](#)
- [Étapes suivantes](#)

Prérequis

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

- Un appareil principal de Linux Greengrass. Si vous n'en avez pas, veuillez consulter [Tutoriel : Débuter avec AWS IoT Greengrass V2](#). Le périphérique principal doit répondre aux exigences suivantes :
 - Sur les appareils principaux de Greengrass exécutant Amazon Linux 2 ou Ubuntu 18.04, la version 2.27 ou ultérieure de la [bibliothèque GNU C](#) (glibc) est installée sur l'appareil.
 - Sur les appareils ARMv7L, tels que Raspberry Pi, les dépendances pour OpenCV-Python sont installées sur l'appareil. Exécutez la commande suivante pour installer les dépendances.

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev  
libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

- Les appareils Raspberry Pi qui exécutent le système d'exploitation Raspberry Pi Bullseye doivent répondre aux exigences suivantes :
 - NumPy 1.22.4 ou version ultérieure installée sur l'appareil. Raspberry Pi OS Bullseye inclut une version antérieure de NumPy. Vous pouvez donc exécuter la commande suivante pour effectuer la mise à niveau NumPy sur l'appareil.

```
pip3 install --upgrade numpy
```

- L'ancienne pile de caméras activée sur l'appareil. Raspberry Pi OS Bullseye inclut une nouvelle pile de caméras activée par défaut et non compatible. Vous devez donc activer la pile de caméras existante.

Pour activer l'ancienne pile de caméras

1. Exécutez la commande suivante pour ouvrir l'outil de configuration du Raspberry Pi.

```
sudo raspi-config
```

2. Sélectionnez Options d'interface.
3. Sélectionnez Legacy camera pour activer l'ancienne pile de caméras.
4. Redémarrez l'appareil Raspberry Pi.

Étape 1 : Abonnez-vous à la rubrique de notifications par défaut

Au cours de cette étape, vous configurez le client AWS IoT MQTT dans la AWS IoT console pour qu'il regarde les messages MQTT publiés par le composant de classification d'images TensorFlow Lite. Par défaut, le composant publie les résultats d'inférence sur le `ml/tflite/image-classification` sujet. Abonnez-vous à cette rubrique avant de déployer le composant sur votre appareil principal Greengrass pour voir les résultats de l'inférence lorsque le composant s'exécute pour la première fois.

Pour vous abonner à la rubrique de notifications par défaut

1. Dans le menu de navigation de la [AWS IoT console](#), choisissez Test, client de test MQTT.
2. Sous S'abonner à un sujet, dans le champ Nom du sujet, entrez **`ml/tflite/image-classification`**.
3. Choisissez Souscrire.

Étape 2 : Déploiement du composant de classification d'images TensorFlow Lite

Au cours de cette étape, vous déployez le composant de classification d'images TensorFlow Lite sur votre appareil principal :

Pour déployer le composant de classification d'images TensorFlow Lite (console)

1. Dans le menu de navigation de la [AWS IoT Greengrass console](#), sélectionnez Composants.
2. Sur la page Components (Composants), sous l'onglet Public components (Composants publics), choisissez `aws.greengrass.TensorFlowLiteImageClassification`.
3. Dans la page `aws.greengrass.TensorFlowLiteImageClassification`, choisissez Déployer.

4. Dans Ajouter au déploiement, sélectionnez l'une des options suivantes :
 - a. Pour fusionner ce composant avec un déploiement existant sur votre dispositif cible, choisissez Add to existing deployment (Ajouter à un déploiement existant), puis sélectionnez le déploiement à réviser.
 - b. Pour créer un nouveau déploiement sur votre dispositif cible, choisissez Create new deployment (Créer un déploiement). S'il existe un déploiement sur votre dispositif et que vous choisissez cette étape, le déploiement existant sera remplacé.
5. Sur la page Specify target (Spécifier une cible), procédez comme suit :
 - a. Sous Deployment information (Informations sur le déploiement), saisissez ou modifiez le nom convivial de votre déploiement.
 - b. Sous Deployment targets (Cibles de déploiement), sélectionnez une cible pour votre déploiement, puis choisissez Next (Suivant). Vous ne pouvez pas modifier la cible de déploiement si vous réviser un déploiement existant.
6. Sur la page Sélectionner les composants, sous Composants publics, vérifiez que le `aws.greengrass.TensorFlowLiteImageClassification` composant est sélectionné, puis choisissez Next.
7. Sur la page Configurer les composants, conservez les paramètres de configuration par défaut et choisissez Next.
8. Sur la page Configure advanced settings (Configurer les paramètres avancés), conservez les paramètres de configuration par défaut et choisissez Next (Suivant).
9. Sur la page de révision, choisissez Déployer

Pour déployer le composant de classification d'images TensorFlow Lite (AWS CLI)

1. Créez un `deployment.json` fichier pour définir la configuration de déploiement du composant de classification d'images TensorFlow Lite. Ce fichier doit ressembler à ce qui suit :

```
{
  "targetArn": "targetArn",
  "components": {
    "aws.greengrass.TensorFlowLiteImageClassification": {
      "componentVersion": 2.1.0,
      "configurationUpdate": {
      }
    }
  }
}
```



```
}
}
```

- Dans le champ `targetArn`, remplacez *targetArn* par l'Amazon Resource Name (ARN) de l'objet ou du groupe d'objets à cibler pour le déploiement, au format suivant :
 - Objet : `arn:aws:iot:region:account-id:thing/thingName`
 - Groupe d'objets : `arn:aws:iot:region:account-id:thinggroup/thingGroupName`
 - Ce didacticiel utilise la version 2.1.0 du composant. Dans l'objet `aws.greengrass.TensorFlowLiteObjectDetection` composant, remplacez-le **2.1.0** pour utiliser une version différente du composant de détection d'objets TensorFlow Lite.
2. Exécutez la commande suivante pour déployer le composant de classification d'images TensorFlow Lite sur le périphérique :

```
aws greengrassv2 create-deployment \
  --cli-input-json file://path/to/deployment.json
```

L'exécution du déploiement peut prendre plusieurs minutes. À l'étape suivante, vérifiez le journal des composants pour vous assurer que le déploiement s'est terminé avec succès et afficher les résultats des inférences.

Étape 3 : Afficher les résultats de l'inférence

Après avoir déployé le composant, vous pouvez consulter les résultats de l'inférence dans le journal du composant sur votre périphérique principal Greengrass et dans AWS IoT le client MQTT de la console. AWS IoT Pour vous abonner à la rubrique sur laquelle le composant publie les résultats d'inférence, consultez [Étape 1 : Abonnez-vous à la rubrique de notifications par défaut](#).

- AWS IoT Client MQTT : pour afficher les résultats publiés par le composant d'inférence dans la [rubrique des notifications par défaut](#), procédez comme suit :
 1. Dans le menu de navigation de la [AWS IoT console](#), choisissez Test, client de test MQTT.
 2. Sous Abonnements, sélectionnez **ml/tflite/image-classification**.

Vous devriez voir des messages similaires à ceux de l'exemple suivant.

```
{
  "timestamp": "2021-01-01 00:00:00.000000",
```

```

"inference-type": "image-classification",
"inference-description": "Top 5 predictions with score 0.3 or above ",
"inference-results": [
  {
    "Label": "cougar, puma, catamount, mountain lion, painter, panther, Felis concolor",
    "Score": "0.5882352941176471"
  },
  {
    "Label": "Persian cat",
    "Score": "0.5882352941176471"
  },
  {
    "Label": "tiger cat",
    "Score": "0.5882352941176471"
  },
  {
    "Label": "dalmatian, coach dog, carriage dog",
    "Score": "0.5607843137254902"
  },
  {
    "Label": "malamute, malemute, Alaskan malamute",
    "Score": "0.5450980392156862"
  }
]
}

```

- Journal des composants : pour afficher les résultats de l'inférence dans le journal des composants, exécutez la commande suivante sur votre appareil principal Greengrass.

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

Vous devriez obtenir des résultats similaires à ceux de l'exemple suivant.

```

2021-01-01 00:00:00.000000 [INFO] (Copier)
aws.greengrass.TensorFlowLiteImageClassification: stdout. Publishing results to the
IoT core....
{scriptName=services.aws.greengrass.TensorFlowLiteImageClassification.lifecycle.Run.script,
serviceName=aws.greengrass.TensorFlowLiteImageClassification, currentState=RUNNING}

2021-01-01 00:00:00.000000 [INFO] (Copier)
aws.greengrass.TensorFlowLiteImageClassification: stdout. {"timestamp":
"2021-01-01 00:00:00.000000", "inference-type": "image-classification", "inference-

```

```
description": "Top 5 predictions with score 0.3 or above ", "inference-results":  
[{"Label": "cougar, puma, catamount, mountain lion, painter, panther, Felis  
concolor", "Score": "0.5882352941176471"}, {"Label": "Persian cat", "Score":  
"0.5882352941176471"}, {"Label": "tiger cat", "Score": "0.5882352941176471"},  
{"Label": "dalmatian, coach dog, carriage dog", "Score": "0.5607843137254902"},  
{"Label": "malamute, malemute, Alaskan malamute", "Score": "0.5450980392156862"}]].  
{scriptName=services.aws.greengrass.TensorFlowLiteImageClassification.lifecycle.Run.script,  
serviceName=aws.greengrass.TensorFlowLiteImageClassification, currentState=RUNNING}
```

Si vous ne pouvez pas voir les résultats d'inférence dans le journal des composants ou dans le client MQTT, cela signifie que le déploiement a échoué ou n'a pas atteint le périphérique principal. Cela peut se produire si votre appareil principal n'est pas connecté à Internet ou ne dispose pas des autorisations nécessaires pour exécuter le composant. Exécutez la commande suivante sur votre appareil principal pour afficher le fichier journal du logiciel AWS IoT Greengrass Core. Ce fichier inclut les journaux du service de déploiement de l'appareil principal Greengrass.

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

Pour de plus amples informations, veuillez consulter [Résolution des problèmes liés à l'inférence par apprentissage automatique](#).

Étapes suivantes

Si vous disposez d'un périphérique principal Greengrass doté d'une interface de caméra compatible, vous pouvez terminer [Tutoriel : effectuer une inférence de classification d'images d'échantillons sur des images provenant d'un appareil photo à l'aide TensorFlow de Lite](#), qui vous montre comment modifier le composant d'inférence d'échantillons pour effectuer une classification d'images sur des images provenant d'une caméra.

Pour explorer plus en détail la configuration de l'exemple de composant d'inférence de [classification d'images TensorFlow Lite](#), essayez ce qui suit :

- Modifiez le paramètre `InferenceInterval` de configuration pour modifier la fréquence d'exécution du code d'inférence.
- Modifiez les paramètres `ImageDirectory` de configuration `ImageName` et dans la configuration du composant d'inférence pour spécifier une image personnalisée à utiliser pour l'inférence.

Pour plus d'informations sur la personnalisation de la configuration des composants publics ou la création de composants d'apprentissage automatique personnalisés, consultez [Personnalisez vos composants d'apprentissage automatique](#).

Tutoriel : effectuer une inférence de classification d'images d'échantillons sur des images provenant d'un appareil photo à l'aide TensorFlow de Lite

Ce didacticiel explique comment utiliser le composant d'inférence de [classification d'images TensorFlow Lite](#) pour effectuer une inférence de classification d'images par exemple sur des images provenant d'une caméra en local sur un périphérique principal Greengrass. Ce composant inclut les dépendances suivantes :

- TensorFlow Composant de magasin de modèles de classification d'images Lite
- TensorFlow Composant d'exécution Lite

Note

Ce didacticiel accède au module caméra pour les appareils [Raspberry Pi](#) ou [NVIDIA Jetson Nano](#), mais AWS IoT Greengrass prend en charge d'autres appareils sur les plateformes ARMv7L, Armv8 ou x86_64. Pour configurer une caméra pour un autre appareil, consultez la documentation correspondante à votre appareil.

Pour plus d'informations sur l'apprentissage automatique sur les appareils Greengrass, consultez. [Exécuter l'inférence de Machine Learning](#)

Rubriques

- [Prérequis](#)
- [Étape 1 : configurer le module de caméra sur votre appareil](#)
- [Étape 2 : vérifier votre abonnement à la rubrique de notifications par défaut](#)
- [Étape 3 : modifier la configuration du composant de classification d'images TensorFlow Lite et le déployer](#)
- [Étape 4 : Afficher les résultats de l'inférence](#)
- [Étapes suivantes](#)

Prérequis

Pour terminer ce didacticiel, vous devez d'abord le terminer [Tutoriel : Effectuer une inférence de classification d'images d'échantillons à l'aide TensorFlow de Lite](#).

Vous avez également besoin des éléments suivants :

- Un appareil central Linux Greengrass doté d'une interface de caméra. Ce didacticiel permet d'accéder au module caméra sur l'un des appareils compatibles suivants :
 - [Raspberry Pi](#) exécutant le système d'[exploitation Raspberry Pi](#) (auparavant appelé Raspbian)
 - [NVIDIA Jetson Nano](#)

Pour plus d'informations sur la configuration d'un appareil Greengrass Core, consultez. [Tutoriel : Débuter avec AWS IoT Greengrass V2](#)

Le périphérique principal doit répondre aux exigences suivantes :

- Sur les appareils principaux de Greengrass exécutant Amazon Linux 2 ou Ubuntu 18.04, la version 2.27 ou ultérieure de la [bibliothèque GNU C](#) (glibc) est installée sur l'appareil.
- Sur les appareils ARMv7L, tels que le Raspberry Pi, les dépendances pour OpenCV-Python sont installées sur l'appareil. Exécutez la commande suivante pour installer les dépendances.

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev  
libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

- Les appareils Raspberry Pi qui exécutent le système d'exploitation Raspberry Pi Bullseye doivent répondre aux exigences suivantes :
 - NumPy 1.22.4 ou version ultérieure installée sur l'appareil. Raspberry Pi OS Bullseye inclut une version antérieure de NumPy. Vous pouvez donc exécuter la commande suivante pour effectuer la mise à niveau NumPy sur l'appareil.

```
pip3 install --upgrade numpy
```

- L'ancienne pile de caméras activée sur l'appareil. Raspberry Pi OS Bullseye inclut une nouvelle pile de caméras qui est activée par défaut et qui n'est pas compatible. Vous devez donc activer la pile de caméras existante.

Pour activer l'ancienne pile de caméras

1. Exécutez la commande suivante pour ouvrir l'outil de configuration du Raspberry Pi.

```
sudo raspi-config
```

2. Sélectionnez Options d'interface.
 3. Sélectionnez Legacy camera pour activer l'ancienne pile de caméras.
 4. Redémarrez l'appareil Raspberry Pi.
- Pour les appareils Raspberry Pi ou NVIDIA Jetson Nano, [module de caméra Raspberry Pi V2 - 8 mégapixels, 1080p](#). Pour savoir comment configurer la caméra, consultez la section [Connexion de la caméra](#) dans la documentation du Raspberry Pi.

Étape 1 : configurer le module de caméra sur votre appareil

Au cours de cette étape, vous allez installer et activer le module caméra de votre appareil. Exécutez les commandes suivantes sur l'appareil.

Raspberry Pi (Armv7l)

1. Installez l'`picamera` interface du module de caméra. Exécutez la commande suivante pour installer le module caméra et les autres bibliothèques Python requises pour ce didacticiel.

```
sudo apt-get install -y python3-picamera
```

2. Vérifiez que Picamera a été correctement installé.

```
sudo -u ggc_user bash -c 'python3 -c "import picamera"'
```

Si la sortie ne contient pas d'erreurs, la validation a abouti.

Note

Si le fichier exécutable Python installé sur votre appareil l'est `python3.7`, utilisez-le `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.

```
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. Exécutez 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.

7. Exécutez la commande suivante pour créer un lien symbolique permettant au composant d'inférence d'accéder à votre caméra depuis l'environnement virtuel créé par le composant d'exécution.

```
sudo ln -s /usr/lib/python3/dist-packages/picamera "MLRootPath/  
greengrass_ml_tf-lite_venv/lib/python3.7/site-packages"
```

La valeur par défaut *MLRootPath* pour ce didacticiel est `/greengrass/v2/work/variant.TensorFlowLite/greengrass_ml`. Le `greengrass_ml_tf-lite_venv` dossier situé à cet emplacement est créé lorsque vous déployez le composant d'inférence pour la première fois dans [Tutoriel : Effectuer une inférence de classification d'images d'échantillons à l'aide TensorFlow de Lite](#).

Jetson Nano (Armv8)

1. Exécutez la commande suivante pour tester la configuration de la caméra.

```
gst-launch-1.0 nvarguscamerasrc num-buffers=1 ! "video/x-raw(memory:NVMM),  
width=1920, height=1080, format=NV12, framerate=30/1" ! nvjpegenc ! filesink  
location=test.jpg
```

Cela capture et enregistre une image nommée `test.jpg` dans votre répertoire actuel.

2. (Facultatif) Redémarrez l'appareil. Si vous rencontrez des problèmes lors de l'exécution de la `gst-launch` commande à l'étape précédente, le redémarrage de votre appareil peut les résoudre.

```
sudo reboot
```

Note

Pour les appareils Armv8 (AArch64), tels qu'un Jetson Nano, il n'est pas nécessaire de créer un lien symbolique pour permettre au composant d'inférence d'accéder à la caméra depuis l'environnement virtuel créé par le composant d'exécution.

Étape 2 : vérifier votre abonnement à la rubrique de notifications par défaut

Dans [Tutoriel : Effectuer une inférence de classification d'images d'échantillons à l'aide TensorFlow de Lite](#), vous avez configuré le client AWS IoT MQTT est configuré dans la AWS IoT console pour regarder les messages MQTT publiés par le composant de classification d'images TensorFlow Lite sur le `ml/tflite/image-classification` sujet. Dans la AWS IoT console, vérifiez que cet abonnement existe. Si ce n'est pas le cas, suivez les étapes décrites [Étape 1 : Abonnez-vous à la rubrique de notifications par défaut](#) pour vous abonner à cette rubrique avant de déployer le composant sur votre appareil principal Greengrass.

Étape 3 : modifier la configuration du composant de classification d'images TensorFlow Lite et le déployer

Au cours de cette étape, vous configurez et déployez le composant de classification d'images TensorFlow Lite sur votre appareil principal :

Pour configurer et déployer le composant de classification d'images TensorFlow Lite (console)

1. Dans le menu de navigation de la [AWS IoT Greengrass console](#), sélectionnez Composants.
2. Sur la page Components (Composants), sous l'onglet Public components (Composants publics), choisissez `aws.greengrass.TensorFlowLiteImageClassification`.
3. Dans la page `aws.greengrass.TensorFlowLiteImageClassification` page, choisissez Déployer.
4. Dans Ajouter au déploiement, sélectionnez l'une des options suivantes :

- a. Pour fusionner ce composant avec un déploiement existant sur votre dispositif cible, choisissez Add to existing deployment (Ajouter à un déploiement existant), puis sélectionnez le déploiement à réviser.
 - b. Pour créer un nouveau déploiement sur votre dispositif cible, choisissez Create new deployment (Créer un déploiement). S'il existe un déploiement sur votre dispositif et que vous choisissez cette étape, le déploiement existant sera remplacé.
5. Sur la page Specify target (Spécifier une cible), procédez comme suit :
 - a. Sous Deployment information (Informations sur le déploiement), saisissez ou modifiez le nom convivial de votre déploiement.
 - b. Sous Deployment targets (Cibles de déploiement), sélectionnez une cible pour votre déploiement, puis choisissez Next (Suivant). Vous ne pouvez pas modifier la cible de déploiement si vous réviser un déploiement existant.
6. Sur la page Sélectionner les composants, sous Composants publics, vérifiez que le `aws.greengrass.TensorFlowLiteImageClassification` composant est sélectionné, puis choisissez Next.
7. Sur la page Configurer les composants, procédez comme suit :
 - a. Sélectionnez le composant d'inférence, puis sélectionnez Configurer le composant.
 - b. Sous Mise à jour de configuration, entrez la mise à jour de configuration suivante dans la zone Configuration à fusionner.

```
{  
  "InferenceInterval": "60",  
  "UseCamera": "true"  
}
```

Avec cette mise à jour de configuration, le composant accède au module de caméra de votre appareil et effectue des inférences sur les images prises par la caméra. Le code d'inférence s'exécute toutes les 60 secondes.

- c. Choisissez Confirm (Confirmer), puis Next (Suivant).
8. Sur la page Configure advanced settings (Configurer les paramètres avancés), conservez les paramètres de configuration par défaut et choisissez Next (Suivant).
9. Sur la page de révision, choisissez Déployer

Pour configurer et déployer le composant de classification d'images TensorFlow Lite (AWS CLI)

1. Créez un `deployment.json` fichier pour définir la configuration de déploiement du composant de classification d'images TensorFlow Lite. Ce fichier doit ressembler à ce qui suit :

```
{
  "targetArn": "targetArn",
  "components": {
    "aws.greengrass.TensorFlowLiteImageClassification": {
      "componentVersion": 2.1.0,
      "configurationUpdate": {
        "InferenceInterval": "60",
        "UseCamera": "true"
      }
    }
  }
}
```

- Dans le champ `targetArn`, remplacez *targetArn* par l'Amazon Resource Name (ARN) de l'objet ou du groupe d'objets à cibler pour le déploiement, au format suivant :
 - Objet : `arn:aws:iot:region:account-id:thing/thingName`
 - Groupe d'objets : `arn:aws:iot:region:account-id:thinggroup/thingGroupName`
- Ce didacticiel utilise la version 2.1.0 du composant. Dans l'objet `aws.greengrass.TensorFlowLiteImageClassification` composant, remplacez-le *2.1.0* pour utiliser une version différente du composant de classification d'images TensorFlow Lite.

Avec cette mise à jour de configuration, le composant accède au module de caméra de votre appareil et effectue des inférences sur les images prises par la caméra. Le code d'inférence s'exécute toutes les 60 secondes. Remplacez les valeurs suivantes

2. Exécutez la commande suivante pour déployer le composant de classification d'images TensorFlow Lite sur le périphérique :

```
aws greengrassv2 create-deployment \
  --cli-input-json file://path/to/deployment.json
```

L'exécution du déploiement peut prendre plusieurs minutes. À l'étape suivante, vérifiez le journal des composants pour vous assurer que le déploiement s'est terminé avec succès et affichez les résultats des inférences.

Étape 4 : Afficher les résultats de l'inférence

Après avoir déployé le composant, vous pouvez consulter les résultats de l'inférence dans le journal du composant sur votre périphérique principal Greengrass et dans AWS IoT le client MQTT de la console. AWS IoT Pour vous abonner à la rubrique sur laquelle le composant publie les résultats d'inférence, consultez [Étape 2 : vérifier votre abonnement à la rubrique de notifications par défaut](#).

- AWS IoT Client MQTT : pour afficher les résultats publiés par le composant d'inférence dans la [rubrique des notifications par défaut](#), procédez comme suit :
 1. Dans le menu de navigation de la [AWS IoT console](#), choisissez Test, client de test MQTT.
 2. Sous Abonnements, sélectionnez **ml/tflite/image-classification**.
- Journal des composants : pour afficher les résultats de l'inférence dans le journal des composants, exécutez la commande suivante sur votre appareil principal Greengrass.

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

Si vous ne pouvez pas voir les résultats d'inférence dans le journal des composants ou dans le client MQTT, cela signifie que le déploiement a échoué ou n'a pas atteint le périphérique principal. Cela peut se produire si votre appareil principal n'est pas connecté à Internet ou ne dispose pas des autorisations requises pour exécuter le composant. Exécutez la commande suivante sur votre appareil principal pour afficher le fichier journal du logiciel AWS IoT Greengrass Core. Ce fichier inclut les journaux du service de déploiement de l'appareil principal Greengrass.

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

Pour de plus amples informations, veuillez consulter [Résolution des problèmes liés à l'inférence par apprentissage automatique](#).

Étapes suivantes

Ce didacticiel explique comment utiliser le composant de classification d'images TensorFlow Lite, avec des options de configuration personnalisées pour effectuer une classification d'images d'échantillons sur des images prises par un appareil photo.

Pour plus d'informations sur la personnalisation de la configuration des composants publics ou la création de composants d'apprentissage automatique personnalisés, consultez [Personnalisez vos composants d'apprentissage automatique](#).

Composants

AWS IoT Greengrass les composants sont des modules logiciels que vous déployez sur les appareils principaux de Greengrass. Les composants peuvent représenter des applications, des programmes d'installation d'exécution, des bibliothèques ou tout autre code que vous exécuteriez sur un appareil. Vous pouvez définir des composants qui dépendent d'autres composants. Par exemple, vous pouvez définir un composant qui installe Python, puis définir ce composant comme une dépendance de vos composants qui exécutent des applications Python. Lorsque vous déployez vos composants sur vos flottes d'appareils, Greengrass déploie uniquement les modules logiciels dont vos appareils ont besoin.

Rubriques

- [AWS-composants fournis](#)
- [Composants pris en charge par l'éditeur](#)
- [Composantes communautaires](#)
- [AWS IoT Greengrass outils de développement](#)
- [Développer des AWS IoT Greengrass composants](#)
- [Déployer AWS IoT Greengrass des composants sur des appareils](#)

AWS-composants fournis

AWS IoT Greengrass fournit et gère des composants prédéfinis que vous pouvez déployer sur vos appareils. Ces composants incluent des fonctionnalités (telles que le gestionnaire de flux), des connecteurs AWS IoT Greengrass V1 (tels que CloudWatch les métriques) et des outils de développement locaux (tels que la AWS IoT Greengrass CLI). Vous pouvez [déployer ces composants](#) sur vos appareils pour leur fonctionnalité autonome, ou vous pouvez les utiliser comme dépendances dans vos composants [Greengrass personnalisés](#).

Note

Plusieurs composants AWS fournis dépendent de versions mineures spécifiques du noyau Greengrass. En raison de cette dépendance, vous devez mettre à jour ces composants lorsque vous mettez à jour le noyau Greengrass vers une nouvelle version mineure. Pour plus d'informations sur les versions spécifiques du noyau dont dépend chaque composant,

consultez la rubrique correspondante sur les composants. Pour plus d'informations sur la mise à jour du noyau, consultez [Mettre à jour le logiciel AWS IoT Greengrass principal \(OTA\)](#).

Lorsqu'un composant possède un type de composant à la fois générique et Lambda, la version actuelle du composant est le type générique et une version précédente du composant est le type Lambda.

Composant	Description	Type de composant	Systèmes d'exploitation pris en charge	Open source	Compatible avec Nucleus Lite
Noyau de Greengrass	Le noyau du logiciel AWS IoT Greengrass Core. Utilisez ce composant pour configurer et mettre à jour le logiciel sur vos appareils principaux.	Noyau	Linux, Windows	Oui	Non
Greengrass Nucleus Lite	Un noyau léger pour les appareils aux ressources limitées, optimisé pour les	Nucleus Lite	Linux	Oui	Non

Composant	Description	Type de composant	Systèmes d'exploitation pris en charge	Open source	Compatibilité avec Nucleus Lite
	appareils de pointe à faible coût et les applications à volume élevé				
Authentification de l'appareil client	Permet aux appareils IoT locaux, appelés appareils clients, de se connecter à l'appareil principal.	Plugin	Linux, Windows	Oui	Non
CloudWatch métriques	Publie des statistiques personnalisées sur Amazon CloudWatch.	Générique, Lambda	Linux, Windows	Oui	Oui

Composant	Description	Type de composant	Systèmes d'exploitation pris en charge	Open source	Compatible avec Nucleus Lite
AWS IoT Device Defender	Informe les administrateurs des modifications de l'état de l'appareil principal Greengrass afin d'identifier les comportements inhabituels.	Générique, Lambda	Linux, Windows	Oui	Non

Composant	Description	Type de composant	Systèmes d'exploitation pris en charge	Open source	Compatible avec Nucleus Lite
Spouleur à disque	Active une option de stockage permanent pour les messages envoyés depuis les appareils principaux de Greengrass vers AWS IoT Core. Ce composant stockera ces messages sortants sur le disque.	Plugin	Linux, Windows	Oui	Non

Composant	Description	Type de composant	Systèmes d'exploitation pris en charge	Open source	Compatible avec Nucleus Lite
Gestionnaire d'applications Docker	Permet AWS IoT Greengrass de télécharger des images Docker depuis Docker Hub et Amazon Elastic Container Registry (Amazon ECR).	Générique	Linux, Windows	Non	Non

Composant	Description	Type de composant	Systèmes d'exploitation pris en charge	Open source	Compatible avec Nucleus Lite
Connecteur Edge pour Kinesis Video Streams	Lit les flux vidéo des caméras locales, publie les flux sur Kinesis Video Streams et affiche les flux dans les tableaux de bord Grafana avec AWS IoT TwinMaker	Générique	Linux	Non	Non

Composant	Description	Type de composant	Systèmes d'exploitation pris en charge	Open source	Compatible avec Nucleus Lite
Greengrass CLI	Fournit une interface de ligne de commande que vous pouvez utiliser pour créer des déploiements locaux et interagir avec le périphérique principal Greengrass et ses composants.	Plugin	Linux, Windows	Oui	Non

Composant	Description	Type de composant	Systèmes d'exploitation pris en charge	Open source	Compatibilité avec Nucleus Lite
Détecteur IP	Transmet les informations de connectivité au AWS IoT Greengrass courtier MQTT afin que les appareils clients puissent découvrir comment se connecter.	Plugin	Linux, Windows	Oui	Non
Firehose	Publie des données via les flux de livraison Amazon Data Firehose vers des destinations situées dans le AWS Cloud	Lambda	Linux	Non	Non

Composant	Description	Type de composant	Systèmes d'exploitation pris en charge	Open source	Compatibilité avec Nucleus Lite
Lanceur Lambda	Gère la configuration des processus et de l'environnement pour les fonctions Lambda.	Générique	Linux	Non	Non
Gestionnaire Lambda	Gère la communication interprocessus et le dimensionnement pour les fonctions Lambda.	Plugin	Linux	Non	Non
Environnements d'exécution (runtimes) Lambda	Fournit des artefacts pour chaque environnement d'exécution Lambda.	Générique	Linux	Non	Non

Composant	Description	Type de composant	Systèmes d'exploitation pris en charge	Open source	Compatibilité avec Nucleus Lite
Routeur d'abonnement Legacy	Gère les abonnements aux fonctions Lambda qui s'exécutent sur AWS IoT Greengrass la version 1.	Générique	Linux	Non	Non
Console de débogage locale	Fournit une console locale que vous pouvez utiliser pour déboguer et gérer le périphérique principal Greengrass et ses composants.	Plugin	Linux, Windows	Oui	Non
Gestionnaire de journaux	Collecte et télécharge les journaux sur l'appareil principal de Greengrass.	Plugin	Linux, Windows	Oui	Non

Composant	Description	Type de composant	Systèmes d'exploitation pris en charge	Open source	Compatibilité avec Nucleus Lite
Composants d'apprentissage automatique	Fournit des modèles d'apprentissage automatique et des exemples de code d'inférence que vous pouvez utiliser pour effectuer des inférences d'apprentissage automatique sur les appareils principaux de Greengrass.	Consultez Composants d'apprentissage automatique .			Non
Adaptateur de protocole Modbus-RTU	Interroge les informations des appareils Modbus RTU locaux.	Lambda	Linux	Non	Non

Composant	Description	Type de composant	Systèmes d'exploitation pris en charge	Open source	Compatibilité avec Nucleus Lite
Émetteur de télémétrie Nucleus	Publie les données de télémétrie relatives à l'état du système recueillies depuis le noyau vers un sujet local ou un sujet AWS IoT Core MQTT.	Plugin	Linux, Windows	Oui	Non
Pont MQTT	Relaie les messages MQTT entre les appareils clients, AWS IoT Greengrass publie/abonnement locaux, et. AWS IoT Core	Plugin	Linux, Windows	Oui	Non

Composant	Description	Type de composant	Systèmes d'exploitation pris en charge	Open source	Compatibilité avec Nucleus Lite
Courtier MQTT 3.1.1 (Moquette)	Exécute un broker MQTT 3.1.1 qui gère les messages entre les appareils clients et le périphérique principal.	Plugin	Linux, Windows	Oui	Non
Courtier MQTT 5 (EMQX)	Exécute un broker MQTT 5 qui gère les messages entre les appareils clients et le périphérique principal.	Générique	Linux, Windows	Non	Non

Composant	Description	Type de composant	Systèmes d'exploitation pris en charge	Open source	Compatible avec Nucleus Lite
Fournisseur PKCS #11	Permet aux composants Greengrass d'accéder à une clé privée et à un certificat que vous stockez en toute sécurité dans un module de sécurité matériel (HSM).	Plugin	Linux	Oui	Non

Composant	Description	Type de composant	Systèmes d'exploitation pris en charge	Open source	Compatibilité avec Nucleus Lite
Directeur des services secrets	Déploie les secrets à partir de AWS Secrets Manager secrets afin que vous puissiez utiliser en toute sécurité des informations d'identification, telles que des mots de passe, dans des composants personnalisés de l'appareil principal de Greengrass.	Plugin	Linux, Windows	Oui	Non

Composant	Description	Type de composant	Systèmes d'exploitation pris en charge	Open source	Compatible avec Nucleus Lite
Tunneling sécurisé	Permet des connexions tunnelisées AWS IoT sécurisées que vous pouvez utiliser pour établir des communications bidirectionnelles avec les principaux appareils de Greengrass qui se trouvent derrière des pare-feux restreints.	Générique	Linux	Non	Oui

Composant	Description	Type de composant	Systèmes d'exploitation pris en charge	Open source	Compatibilité avec Nucleus Lite
Shadow Manager	Permet l'interaction avec les ombres sur le périphérique principal. Il gère le stockage des documents cachés ainsi que la synchronisation des états d'ombre locaux avec le service AWS IoT Device Shadow.	Plugin	Linux, Windows	Oui	Non
Amazon SNS	Publie des messages sur les rubriques Amazon SNS.	Lambda	Linux	Non	Non

Composant	Description	Type de composant	Systèmes d'exploitation pris en charge	Open source	Compatibilité avec Nucleus Lite
Gestionnaire de flux	Diffuse de gros volumes de données provenant de sources locales vers le AWS Cloud.	Générique	Linux, Windows	Non	Oui
Redirecteur de journal du système	Téléchargez les journaux systemd-journal dans le AWS Cloud	Générique	Linux	Oui	Oui
Agent Systems Manager	Gérez le périphérique principal avec AWS Systems qui vous permet de patcher des appareils, d'exécuter des commandes, etc.	Générique	Linux	Oui	Non

Composant	Description	Type de composant	Systèmes d'exploitation pris en charge	Open source	Compatible avec Nucleus Lite
Service d'échange de jetons	Fournit des AWS informations d'identification que vous pouvez utiliser pour interagir avec AWS les services.	Générique	Linux, Windows	Non	Non
Collecteur IoT SiteWise OPC UA	Collecte des données à partir des serveurs OPC-UA.	Générique	Linux, Windows	Non	Non
Simulateur de source de données IoT SiteWise OPC UA	Exécute un serveur OPC-UA local qui génère des exemples de données.	Générique	Linux, Windows	Non	Non
SiteWise Éditeur IoT	Publie des données dans le AWS cloud.	Générique	Linux, Windows	Non	Non

Composant	Description	Type de composant	Systèmes d'exploitation pris en charge	Open source	Compatible avec Nucleus Lite
SiteWise Processeur IoT	Traite les données sur les appareils principaux de Greengrass.	Générique	Linux, Windows	Non	Non

Noyau de Greengrass

Le composant Greengrass nucleus (`aws.greengrass.Nucleus`) est un composant obligatoire et le minimum requis pour exécuter le logiciel AWS IoT Greengrass Core sur un appareil. Vous pouvez configurer ce composant pour personnaliser et mettre à jour votre logiciel AWS IoT Greengrass Core à distance. Déployez ce composant pour configurer des paramètres tels que le proxy, le rôle de l'appareil et AWS IoT la configuration des objets sur vos appareils principaux.

Note

À partir de la version 2.14.0 de Greengrass, une version optimisée de l'environnement d'exécution des périphériques Nucleus est disponible pour les appareils à faible encombrement mémoire. Consultez [Greengrass nucleus lite](#) pour plus d'informations sur sa configuration et son utilisation.

Important

Lorsque la version du composant du noyau change, ou lorsque vous modifiez certains paramètres de configuration, le logiciel AWS IoT Greengrass Core, qui inclut le noyau et tous les autres composants de votre appareil, redémarre pour appliquer les modifications. Lorsque vous déployez un composant, AWS IoT Greengrass installe les dernières versions prises en charge de toutes les dépendances de ce composant. De ce fait, les nouvelles versions de correctif des composants publics AWS fournis peuvent être automatiquement

déployées sur vos appareils principaux si vous ajoutez de nouveaux appareils à un groupe d'objets ou si vous mettez à jour le déploiement qui cible ces appareils. Certaines mises à jour automatiques, telles que la mise à jour du noyau, peuvent provoquer le redémarrage inattendu de vos appareils.

Pour éviter les mises à jour involontaires d'un composant en cours d'exécution sur votre appareil, nous vous recommandons d'inclure directement votre version préférée de ce composant lorsque vous [créez un déploiement](#). Pour plus d'informations sur le comportement de mise à jour du logiciel AWS IoT Greengrass Core, consultez [Mettre à jour le logiciel AWS IoT Greengrass principal \(OTA\)](#).

Rubriques

- [Versions](#)
- [Exigences relatives aux dispositifs](#)
- [Plateformes prises en charge](#)
- [Système d'exploitation](#)
- [Prérequis](#)
- [Dépendances](#)
- [Téléchargement et installation](#)
- [Configuration](#)
- [Fichier journal local](#)
- [Journal des modifications](#)

Versions

Les versions de ce composant sont les suivantes :

- 2,16.x
- 2,15.x
- 2,14.x
- 2,13.x
- 2,12. x
- 2,11.x
- 2.10.x

- 2,9. x
- 2,8 x
- 2.7.x
- 2.6.x
- 2,5.x
- 2.4.x
- 2.3.x
- 2.2.x
- 2.1.x
- 2.0.x

Exigences relatives aux dispositifs

Note

Vous pouvez utiliser AWS IoT Device Tester for AWS IoT Greengrass pour vérifier que votre appareil peut exécuter le logiciel AWS IoT Greengrass Core et communiquer avec le AWS Cloud. Pour de plus amples informations, veuillez consulter [Utilisation AWS IoT Device Tester pour la AWS IoT Greengrass V2](#).

Linux

- L'utilisation d'un ventilateur [Région AWS](#) qui supporte AWS IoT Greengrass V2. Pour obtenir la liste des régions prises en charge, consultez [Points de terminaison et quotas AWS IoT Greengrass V2](#) dans le Références générales AWS.
- 256 Mo d'espace disque minimum disponible pour le logiciel AWS IoT Greengrass Core. Cette exigence n'inclut pas les composants déployés sur le périphérique principal.
- 96 Mo de RAM au minimum sont alloués au logiciel AWS IoT Greengrass Core. Cette exigence n'inclut pas les composants qui s'exécutent sur le périphérique principal. Pour de plus amples informations, veuillez consulter [Contrôlez l'allocation de mémoire grâce aux options JVM](#).
- Java Runtime Environment (JRE) version 8 ou supérieure. Java doit être disponible sur la variable d'environnement [PATH](#) de l'appareil. Pour utiliser Java pour développer des composants personnalisés, vous devez installer un kit de développement Java (JDK). Nous

vous recommandons d'utiliser les versions de support à long terme d'[Amazon Corretto](#) ou d'OpenJDK. La version 8 ou supérieure est requise.

- [Bibliothèque GNU C](#) (glibc) version 2.25 ou supérieure.
- Vous devez exécuter le logiciel AWS IoT Greengrass Core en tant qu'utilisateur root. Utilisez `sudo`, par exemple.
- L'utilisateur root qui exécute le logiciel AWS IoT Greengrass Core, par exemple `root`, doit être autorisé à fonctionner `sudo` avec n'importe quel utilisateur et n'importe quel groupe. Le `/etc/sudoers` fichier doit autoriser cet utilisateur à s'exécuter `sudo` en tant qu'autre groupe. L'autorisation accordée à l'utilisateur `/etc/sudoers` doit ressembler à l'exemple suivant.

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

- Le périphérique principal doit être capable d'effectuer des demandes sortantes vers un ensemble de points de terminaison et de ports. Pour de plus amples informations, veuillez consulter [Autoriser le trafic des appareils via un proxy ou un pare-feu](#).
- Le `/tmp` répertoire doit être monté avec des `exec` autorisations.
- Toutes les commandes shell suivantes :
 - `ps -ax -o pid,ppid`
 - `sudo`
 - `sh`
 - `kill`
 - `cp`
 - `chmod`
 - `rm`
 - `ln`
 - `echo`
 - `exit`
 - `id`
 - `uname`
 - `grep`
- Votre appareil peut également nécessiter les commandes shell facultatives suivantes :
 - (Facultatif) `systemctl`. Cette commande est utilisée pour configurer le logiciel AWS IoT Greengrass Core en tant que service système.

- (Facultatif) `useraddgroupadd`, `etusermod`. Ces commandes sont utilisées pour configurer l'utilisateur et le `ggc_user` groupe `ggc_group` système du système.
- (Facultatif) `mkfifo`. Cette commande est utilisée pour exécuter les fonctions Lambda en tant que composants.
- Pour configurer les limites de ressources système pour les processus des composants, votre appareil doit exécuter le noyau Linux version 2.6.24 ou ultérieure.
- Pour exécuter les fonctions Lambda, votre appareil doit répondre à des exigences supplémentaires. Pour de plus amples informations, veuillez consulter [Exigences relatives à la fonction Lambda](#).

Windows

- L'utilisation d'un ventilateur [Région AWS](#) qui supporte AWS IoT Greengrass V2. Pour obtenir la liste des régions prises en charge, consultez [Points de terminaison et quotas AWS IoT Greengrass V2](#) dans le Références générales AWS.
- 256 Mo d'espace disque minimum disponible pour le logiciel AWS IoT Greengrass Core. Cette exigence n'inclut pas les composants déployés sur le périphérique principal.
- 160 Mo de RAM au minimum sont alloués au logiciel AWS IoT Greengrass Core. Cette exigence n'inclut pas les composants qui s'exécutent sur le périphérique principal. Pour de plus amples informations, veuillez consulter [Contrôlez l'allocation de mémoire grâce aux options JVM](#).
- Java Runtime Environment (JRE) version 8 ou supérieure. Java doit être disponible sur la variable système [PATH](#) de l'appareil. Pour utiliser Java pour développer des composants personnalisés, vous devez installer un kit de développement Java (JDK). Nous vous recommandons d'utiliser les versions de support à long terme d'[Amazon Corretto](#) ou d'OpenJDK. La version 8 ou supérieure est requise.

Note

Pour utiliser la version 2.5.0 du noyau [Greengrass](#), vous devez utiliser une version 64 bits du Java Runtime Environment (JRE). La version 2.5.1 de Greengrass Nucleus est compatible avec 32 bits et 64 bits. JREs

- L'utilisateur qui installe le logiciel AWS IoT Greengrass Core doit être un administrateur.

- Vous devez installer le logiciel AWS IoT Greengrass Core en tant que service système. Spécifiez `--setup-system-service true` le moment où vous installez le logiciel.
- Chaque utilisateur qui exécute des processus de composants doit exister dans le LocalSystem compte, et le nom et le mot de passe de l'utilisateur doivent figurer dans l'instance Credential Manager du LocalSystem compte. Vous pouvez configurer cet utilisateur en suivant les instructions d'[installation du logiciel AWS IoT Greengrass Core](#).
- Le périphérique principal doit être capable d'effectuer des demandes sortantes vers un ensemble de points de terminaison et de ports. Pour de plus amples informations, veuillez consulter [Autoriser le trafic des appareils via un proxy ou un pare-feu](#).

Plateformes prises en charge

AWS IoT Greengrass prend officiellement en charge les appareils exécutant les plateformes suivantes. Les appareils dont les plateformes ne figurent pas dans cette liste peuvent fonctionner, mais les AWS IoT Greengrass tests ne sont effectués que sur les plateformes spécifiées.

Linux

Architectures :

- Armv7l
- Bras (8) AArch64
- x86_64

Windows

Architectures :

- x86_64

Versions :

- Windows 10
- Windows 11
- Windows Server 2019
- Windows Server 2022

 Note

Certaines AWS IoT Greengrass fonctionnalités ne sont actuellement pas prises en charge sur les appareils Windows. Pour plus d'informations, consultez [Compatibilité des fonctionnalités de Greengrass](#) et [Considérations relatives aux fonctionnalités](#).

Considérations relatives aux fonctionnalités

Certaines AWS IoT Greengrass fonctionnalités ne sont actuellement pas prises en charge sur les appareils Windows. Passez en revue les différences entre les fonctionnalités pour vérifier si un appareil Windows répond à vos exigences. Pour de plus amples informations, veuillez consulter [Compatibilité des fonctionnalités de Greengrass](#).

[Pour créer un système d'exploitation personnalisé basé sur Linux, vous pouvez utiliser la BitBake recette AWS IoT Greengrass du projet. meta-aws](#) Le meta-aws projet fournit des recettes que vous pouvez utiliser pour développer des fonctionnalités logicielles de AWS pointe dans des systèmes [Linux embarqués](#) conçus avec [OpenEmbedded](#) des frameworks de construction Yocto Project. Le projet [Yocto est un projet](#) de collaboration open source qui vous aide à créer des systèmes Linux personnalisés pour les applications embarquées, quelle que soit l'architecture matérielle. La BitBake recette pour AWS IoT Greengrass installer, configurer et exécuter automatiquement le logiciel AWS IoT Greengrass Core sur votre appareil.

Les plateformes Linux peuvent également fonctionner AWS IoT Greengrass dans un conteneur Docker. Pour de plus amples informations, veuillez consulter [Exécuter le logiciel AWS IoT Greengrass Core dans un conteneur Docker](#).

Système d'exploitation

Ce composant peut être installé sur les appareils principaux qui exécutent les systèmes d'exploitation suivants :

- Linux
- Windows

Pour de plus amples informations, veuillez consulter [Plateformes prises en charge](#).

Prérequis

Les appareils doivent répondre à certaines exigences pour installer et exécuter le noyau Greengrass et le logiciel AWS IoT Greengrass Core. Pour de plus amples informations, veuillez consulter [Exigences relatives aux dispositifs](#).

Le composant Greengrass nucleus est compatible pour fonctionner dans un VPC. Pour déployer ce composant dans un VPC, les éléments suivants sont requis.

- Le composant Greengrass nucleus doit disposer d'une connectivité AWS IoT data, d' AWS IoT informations d'identification et d'Amazon S3.

Dépendances

Le noyau Greengrass n'inclut aucune dépendance entre les composants. Cependant, plusieurs composants AWS fournis incluent le noyau en tant que dépendance. Pour de plus amples informations, veuillez consulter [AWS-composants fournis](#).

Pour plus d'informations sur les dépendances des composants, consultez la [référence de la recette des composants](#).

Téléchargement et installation

Vous pouvez télécharger un programme d'installation qui configure le composant Greengrass nucleus sur votre appareil. Ce programme d'installation configure votre appareil en tant qu'appareil principal de Greengrass. Vous pouvez effectuer deux types d'installations : une installation rapide qui crée les AWS ressources nécessaires pour vous, ou une installation manuelle dans laquelle vous créez les AWS ressources vous-même. Pour de plus amples informations, veuillez consulter [Installation du logiciel AWS IoT Greengrass de base](#).

Vous pouvez également suivre un tutoriel pour installer le noyau Greengrass et découvrir le développement des composants Greengrass. Pour de plus amples informations, veuillez consulter [Tutoriel : Débuter avec AWS IoT Greengrass V2](#).

Configuration

Ce composant fournit les paramètres de configuration suivants que vous pouvez personnaliser lorsque vous déployez le composant. Certains paramètres nécessitent le redémarrage du logiciel AWS IoT Greengrass Core pour être pris en compte. Pour plus d'informations sur les raisons et les

modalités de configuration de ce composant, consultez [Configuration du logiciel AWS IoT Greengrass de base](#).

`iotRoleAlias`

Alias de AWS IoT rôle qui pointe vers un rôle IAM d'échange de jetons. Le fournisseur AWS IoT d'informations d'identification assume ce rôle pour permettre au dispositif principal de Greengrass d'interagir avec AWS les services. Pour de plus amples informations, veuillez consulter [Autoriser les appareils principaux à interagir avec les AWS services](#).

Lorsque vous exécutez le logiciel AWS IoT Greengrass Core avec `--provision true` cette option, le logiciel fournit un alias de rôle et définit sa valeur dans le composant du noyau.

`interpolateComponentConfiguration`

(Facultatif) Vous pouvez activer le noyau Greengrass pour interpoler les [variables de recette des composants dans les configurations des composants](#) et [fusionner](#) les mises à jour de configuration. Nous vous recommandons de définir cette option sur `true` afin que le périphérique principal puisse exécuter les composants Greengrass qui utilisent des variables de recette dans leurs configurations.

Cette fonctionnalité est disponible pour les versions 2.6.0 et ultérieures de ce composant.

Valeur par défaut : `false`

`networkProxy`

(Facultatif) Le proxy réseau à utiliser pour toutes les connexions. Pour de plus amples informations, veuillez consulter [Connexion au port 443 ou via un proxy réseau](#).

Important

Lorsque vous déployez une modification de ce paramètre de configuration, le logiciel AWS IoT Greengrass Core redémarre pour que la modification prenne effet.

Cet objet contient les informations suivantes :

`noProxyAddresses`

(Facultatif) Liste séparée par des virgules d'adresses IP ou de noms d'hôtes exemptés du proxy.

proxy

Le proxy auquel se connecter. Cet objet contient les informations suivantes :

url

URL du serveur proxy au format `scheme://userinfo@host:port`.

- `scheme`— Le schéma, qui doit être `http` ou `https`.

Important

Les appareils principaux de Greengrass doivent exécuter [Greengrass noyau v2.5.0](#) ou version ultérieure pour utiliser les proxys HTTPS.

Si vous configurez un proxy HTTPS, vous devez ajouter le certificat CA du serveur proxy au certificat CA racine Amazon de l'appareil principal. Pour de plus amples informations, veuillez consulter [Permettre au périphérique principal de faire confiance à un proxy HTTPS](#).

- `userinfo`— (Facultatif) Les informations relatives au nom d'utilisateur et au mot de passe. Si vous spécifiez ces informations dans `url`, le périphérique principal de Greengrass ignore les `username` et `password`.
- `host`— Le nom d'hôte ou l'adresse IP du serveur proxy.
- `port`— (Facultatif) Le numéro de port. Si vous ne spécifiez pas le port, le périphérique principal de Greengrass utilise les valeurs par défaut suivantes :
 - `http`— 80
 - `https`— 443

username

(Facultatif) Le nom d'utilisateur qui authentifie le serveur proxy.

password

(Facultatif) Le mot de passe qui authentifie le serveur proxy.

mqtt

(Facultatif) La configuration MQTT pour le périphérique principal de Greengrass. Pour de plus amples informations, veuillez consulter [Connexion au port 443 ou via un proxy réseau](#).

 Important

Lorsque vous déployez une modification de ce paramètre de configuration, le logiciel AWS IoT Greengrass Core redémarre pour que la modification prenne effet.

Cet objet contient les informations suivantes :

port

(Facultatif) Le port à utiliser pour les connexions MQTT.

Valeur par défaut : 8883

keepAliveTimeoutMs

(Facultatif) Durée en millisecondes entre chaque PING message envoyé par le client pour maintenir la connexion MQTT active. Cette valeur doit être supérieure à `pingTimeoutMs`.

Par défaut : 60000 (60 secondes)

pingTimeoutMs

(Facultatif) Durée en millisecondes pendant laquelle le client attend de recevoir un PINGACK message du serveur. Si le délai d'attente dépasse le délai imparti, le périphérique principal ferme et rouvre la connexion MQTT. Cette valeur doit être inférieure à `keepAliveTimeoutMs`.

Par défaut : 30000 (30 secondes)

operationTimeoutMs

(Facultatif) Durée en millisecondes pendant laquelle le client attend la fin des opérations MQTT (telles que CONNECT ou). PUBLISH Cette option ne s'applique pas aux messages MQTT PING ou Keep Alive.

Par défaut : 30000 (30 secondes)

maxInFlightPublishes

(Facultatif) Le nombre maximum de messages MQTT QoS 1 non reconnus qui peuvent être envoyés en même temps.

Cette fonctionnalité est disponible pour les versions 2.1.0 et ultérieures de ce composant.

Valeur par défaut : 5

Plage valide : valeur maximale de 100

`maxMessageSizeInBytes`

(Facultatif) Taille maximale d'un message MQTT. Si un message dépasse cette taille, le noyau de Greengrass rejette le message avec une erreur.

Cette fonctionnalité est disponible pour les versions 2.1.0 et ultérieures de ce composant.

Par défaut : 131072 (128 Ko)

Plage valide : valeur maximale de 2621440 (2,5 Mo)

`maxPublishRetry`

(Facultatif) Le nombre maximum de fois que vous pouvez réessayer un message qui ne parvient pas à être publié. Vous pouvez spécifier de -1 réessayer un nombre illimité de fois.

Cette fonctionnalité est disponible pour les versions 2.1.0 et ultérieures de ce composant.

Valeur par défaut : 100

`spooler`

(Facultatif) Configuration du spouleur MQTT pour le périphérique principal Greengrass. Cet objet contient les informations suivantes :

`storageType`

Type de stockage pour le stockage des messages. Si `storageType` est défini sur `Disk`, il `pluginName` peut être configuré. Vous pouvez spécifier `Memory` ou `Disk`.

[Cette fonctionnalité est disponible pour les versions 2.11.0 et ultérieures du composant Greengrass nucleus.](#)

 Important

Si le spouleur MQTT `storageType` est réglé sur `Disk` et que vous souhaitez rétrograder Greengrass Nucleus de la version 2.11.x à une version antérieure, vous devez rétablir la configuration sur `Memory`. La seule configuration prise en charge dans les `storageType` versions 2.10.x et antérieures de Greengrass nucleus est `Memory`. Le non-respect de ces consignes peut entraîner la rupture du spouleur.

Cela empêcherait votre appareil principal Greengrass d'envoyer des messages MQTT au. AWS Cloud

Valeur par défaut : Memory

pluginName

(Facultatif) Le nom du composant du plugin. Ce composant ne sera utilisé que s'il storageType est défini surDisk. Cette option utilise par défaut le Greengrass fourni par [spouleur à disque](#) Greengrass aws.greengrass.DiskSpooler et l'utilisera.

[Cette fonctionnalité est disponible pour les versions 2.11.0 et ultérieures du composant Greengrass nucleus.](#)

Valeur par défaut : "aws.greengrass.DiskSpooler"

maxSizeInBytes

(Facultatif) Taille maximale du cache dans lequel le périphérique principal stocke les messages MQTT non traités en mémoire. Si le cache est plein, les nouveaux messages sont rejetés.

Par défaut : 2621440 (2,5 Mo)

keepQos0WhenOffline

(Facultatif) Vous pouvez mettre en attente les messages MQTT QoS 0 que le périphérique principal reçoit lorsqu'il est hors ligne. Si vous définissez cette option surtrue, le périphérique principal met en attente les messages QoS 0 qu'il ne peut pas envoyer lorsqu'il est hors ligne. Si vous définissez cette option surfalse, le périphérique principal supprime ces messages. Le périphérique principal met toujours en attente les messages QoS 1, sauf si la bobine est pleine.

Valeur par défaut : false

version

(Facultatif) La version de MQTT. Vous pouvez spécifier mqtt3 ou mqtt5.

[Cette fonctionnalité est disponible pour les versions 2.10.0 et ultérieures du composant Greengrass nucleus.](#)

Valeur par défaut : mqtt5

`receiveMaximum`

(Facultatif) Le nombre maximum de paquets QoS1 non reconnus que le broker peut envoyer.

[Cette fonctionnalité est disponible pour les versions 2.10.0 et ultérieures du composant Greengrass nucleus.](#)

Valeur par défaut : 100

`sessionExpirySeconds`

(Facultatif) Durée en secondes que vous pouvez demander à IoT Core pour qu'une session dure. La durée par défaut est la durée maximale prise en charge par AWS IoT Core.

[Cette fonctionnalité est disponible pour les versions 2.10.0 et ultérieures du composant Greengrass nucleus.](#)

Valeur par défaut : 604800 (7 days)

`minimumReconnectDelaySeconds`

(Facultatif) Une option pour le comportement de reconnexion. Durée minimale, en secondes, nécessaire à la reconnexion de MQTT.

[Cette fonctionnalité est disponible pour les versions 2.10.0 et ultérieures du composant Greengrass nucleus.](#)

Valeur par défaut : 1

`maximumReconnectDelaySeconds`

(Facultatif) Une option pour le comportement de reconnexion. Durée maximale en secondes pendant laquelle MQTT se reconnecte.

[Cette fonctionnalité est disponible pour les versions 2.10.0 et ultérieures du composant Greengrass nucleus.](#)

Valeur par défaut : 120

`minimumConnectedTimeBeforeRetryResetSeconds`

(Facultatif) Une option pour le comportement de reconnexion. Durée en secondes pendant laquelle une connexion doit être active avant que le délai de nouvelle tentative ne soit rétabli au minimum.

Cette fonctionnalité est disponible pour les versions 2.10.0 et ultérieures du composant Greengrass nucleus.

Valeur par défaut : 30

jvmOptions

(Facultatif) Les options JVM à utiliser pour exécuter le logiciel AWS IoT Greengrass Core. Pour plus d'informations sur les options JVM recommandées pour exécuter le logiciel AWS IoT Greengrass Core, consultez [Contrôlez l'allocation de mémoire grâce aux options JVM](#).

Important

Lorsque vous déployez une modification de ce paramètre de configuration, le logiciel AWS IoT Greengrass Core redémarre pour que la modification prenne effet.

iotDataEndpoint

Le point AWS IoT de terminaison de données pour votre Compte AWS.

Lorsque vous exécutez le logiciel AWS IoT Greengrass Core avec `--provision true` cette option, le logiciel extrait vos données et informations d'identification des points de terminaison AWS IoT et les place dans le composant Nucleus.

iotCredEndpoint

Le point de terminaison des informations d' AWS IoT identification de votre Compte AWS.

Lorsque vous exécutez le logiciel AWS IoT Greengrass Core avec `--provision true` cette option, le logiciel extrait vos données et informations d'identification des points de terminaison AWS IoT et les place dans le composant Nucleus.

greengrassDataPlaneEndpoint

Cette fonctionnalité est disponible dans les versions 2.7.0 et ultérieures de ce composant.

Pour de plus amples informations, veuillez consulter [Utiliser un certificat d'appareil signé par une autorité de certification privée](#).

greengrassDataPlanePort

Cette fonctionnalité est disponible dans les versions 2.0.4 et ultérieures de ce composant.

(Facultatif) Port à utiliser pour les connexions au plan de données. Pour de plus amples informations, veuillez consulter [Connexion au port 443 ou via un proxy réseau](#).

 Important

Vous devez spécifier un port sur lequel l'appareil peut établir des connexions sortantes. Si vous spécifiez un port bloqué, l'appareil ne pourra pas se connecter pour AWS IoT Greengrass recevoir des déploiements.

Sélectionnez l'une des options suivantes :

- 443
- 8443

Valeur par défaut : 8443

`awsRegion`

Le Région AWS à utiliser.

`runWithDefault`

L'utilisateur du système à utiliser pour exécuter les composants.

 Important

Lorsque vous déployez une modification de ce paramètre de configuration, le logiciel AWS IoT Greengrass Core redémarre pour que la modification prenne effet.

Cet objet contient les informations suivantes :

`posixUser`

Le nom ou l'ID de l'utilisateur du système et, éventuellement, du groupe système que le périphérique principal utilise pour exécuter les composants génériques et Lambda. Spécifiez l'utilisateur et le groupe en les séparant par deux points (:) au format suivant : `user:group`. Le groupe est facultatif. Si vous ne spécifiez aucun groupe, le logiciel AWS IoT Greengrass Core utilise le groupe principal pour l'utilisateur. Par exemple, vous définissez `ggc_user` ou `ggc_user:ggc_group`. Pour de plus amples informations, veuillez consulter [Configurer l'utilisateur qui exécute les composants](#).

Lorsque vous exécutez le programme d'installation du logiciel AWS IoT Greengrass Core avec l'option `--component-default-user ggc_user:ggc_group`, le logiciel définit ce paramètre dans le composant Nucleus.

`windowsUser`

Cette fonctionnalité est disponible dans les versions 2.5.0 et ultérieures de ce composant.

Nom de l'utilisateur Windows à utiliser pour exécuter ce composant sur les appareils principaux de Windows. L'utilisateur doit exister sur chaque appareil principal de Windows, et son nom et son mot de passe doivent être stockés dans l'instance Credentials Manager du LocalSystem compte. Pour de plus amples informations, veuillez consulter [Configurer l'utilisateur qui exécute les composants](#).

Lorsque vous exécutez le programme d'installation du logiciel AWS IoT Greengrass Core avec l'option `--component-default-user ggc_user`, le logiciel définit ce paramètre dans le composant Nucleus.

`systemResourceLimits`

Cette fonctionnalité est disponible dans les versions 2.4.0 et ultérieures de ce composant. AWS IoT Greengrass ne prend actuellement pas en charge cette fonctionnalité sur les appareils Windows principaux.

Les limites de ressources système à appliquer par défaut aux processus de composants Lambda génériques et non conteneurisés. Vous pouvez annuler les limites de ressources système pour des composants individuels lorsque vous créez un déploiement. Pour de plus amples informations, veuillez consulter [Configuration des limites de ressources système pour les composants](#).

Cet objet contient les informations suivantes :

`cpus`

Temps processeur maximal que les processus de chaque composant peuvent utiliser sur le périphérique principal. Le temps processeur total d'un appareil principal est équivalent au nombre de cœurs processeurs de l'appareil. Par exemple, sur un périphérique principal doté de 4 cœurs de processeur, vous pouvez définir cette valeur 2 pour limiter les processus de chaque composant à 50 % d'utilisation de chaque cœur de processeur. Sur un appareil doté d'un cœur de processeur, vous pouvez définir cette valeur 0.25 pour limiter les processus de chaque composant à 25 % d'utilisation du processeur. Si vous

définissez cette valeur sur un nombre supérieur au nombre de cœurs de processeur, le logiciel AWS IoT Greengrass Core ne limite pas l'utilisation du processeur par les composants.

`memory`

Quantité maximale de RAM (en kilo-octets) que les processus de chaque composant peuvent utiliser sur le périphérique principal.

`s3EndpointType`

(Facultatif) Type de point de terminaison S3. Ce paramètre ne prendra effet que pour la région USA Est (Virginie du Nordus-east-1) (). La définition de ce paramètre à partir d'une autre région sera ignorée. Sélectionnez l'une des options suivantes :

- **REGIONAL**— Le client S3 et l'URL présignée utilisent le point de terminaison régional.
- **GLOBAL**— Le client S3 et l'URL présignée utilisent l'ancien point de terminaison.
- **DUALSTACK**— L'URL présignée S3 utilise le point de terminaison dualstack.

Valeur par défaut : `GLOBAL`

`fipsMode`

(Facultatif) Fait en sorte que Greengrass utilise les points de terminaison FIPS. Pour plus d'informations sur la façon d'activer les points de terminaison FIPS, consultez la section Points de terminaison [FIPS](#).

Sélectionnez l'une des options suivantes :

- `true`Lorsqu'il est défini sur `true`, les points de terminaison utiliseront le point de terminaison FIPS.
- `false`Lorsque la valeur est fausse, les points de terminaison n'utiliseront pas le point de terminaison FIPS.

Valeur par défaut : `false`

`logging`

(Facultatif) Configuration de journalisation pour le périphérique principal. Pour plus d'informations sur la configuration et l'utilisation des journaux Greengrass, consultez. [AWS IoT Greengrass Journaux de surveillance](#)

Cet objet contient les informations suivantes :

level

(Facultatif) Le niveau minimum de messages de journal à afficher.

Choisissez parmi les niveaux de journalisation suivants, listés ici par ordre de niveau :

- DEBUG
- INFO
- WARN
- ERROR

Valeur par défaut : INFO

format

(Facultatif) Le format des données des journaux. Sélectionnez l'une des options suivantes :

- TEXT— Choisissez cette option si vous souhaitez afficher les journaux sous forme de texte.
- JSON— Choisissez cette option si vous souhaitez afficher les journaux à l'aide de la [commande Greengrass CLI logs](#) ou interagir avec les journaux par programmation.

Valeur par défaut : TEXT

outputType

(Facultatif) Type de sortie pour les journaux. Sélectionnez l'une des options suivantes :

- FILE— Le logiciel AWS IoT Greengrass Core génère des journaux dans les fichiers du répertoire que vous spécifiez `outputDirectory`.
- CONSOLE— Le logiciel AWS IoT Greengrass Core imprime les journaux sur `stdout`. Choisissez cette option pour afficher les journaux au fur et à mesure que le périphérique principal les imprime.

Valeur par défaut : FILE

fileSizeKB

(Facultatif) Taille maximale de chaque fichier journal (en kilo-octets). Lorsqu'un fichier journal dépasse cette taille maximale, le logiciel AWS IoT Greengrass Core crée un nouveau fichier journal.

Ce paramètre s'applique uniquement lorsque vous spécifiez FILE pour `outputType`.

Valeur par défaut : 1024

totalLogsSizeKB

(Facultatif) Taille totale maximale des fichiers journaux (en kilo-octets) pour chaque composant, y compris le noyau de Greengrass. [Les fichiers journaux du noyau Greengrass incluent également les journaux des composants du plugin.](#) Lorsque la taille totale des fichiers journaux d'un composant dépasse cette taille maximale, le logiciel AWS IoT Greengrass Core supprime les fichiers journaux les plus anciens de ce composant.

Ce paramètre est équivalent au paramètre de [limite d'espace disque](#) (diskSpaceLimit) [du composant du gestionnaire de journaux](#), que vous pouvez spécifier pour le noyau Greengrass (système) et pour chaque composant. Le logiciel AWS IoT Greengrass Core utilise le minimum des deux valeurs comme taille maximale totale du log pour le noyau Greengrass et chaque composant.

Ce paramètre s'applique uniquement lorsque vous spécifiez FILE pour outputType.

Valeur par défaut : 10240

outputDirectory

(Facultatif) Le répertoire de sortie pour les fichiers journaux.

Ce paramètre s'applique uniquement lorsque vous spécifiez FILE pour outputType.

Par défaut : `/greengrass/v2/logs`, où se `/greengrass/v2` trouve le dossier AWS IoT Greengrass racine.

fleetstatus

Ce paramètre est disponible dans les versions 2.1.0 et ultérieures de ce composant.

(Facultatif) Configuration de l'état du parc pour le périphérique principal.

Cet objet contient les informations suivantes :

periodicStatusPublishIntervalSeconds

(Facultatif) Durée (en secondes) entre laquelle le périphérique principal publie l'état de l'appareil sur le AWS Cloud.

Minimum : 86400 (24 heures)

Par défaut : 86400 (24 heures)

telemetry

(Facultatif) Configuration de la télémétrie de l'état du système pour le périphérique principal. Pour plus d'informations sur les métriques de télémétrie et sur la manière d'agir sur les données de télémétrie, consultez. [Collectez les données de télémétrie relatives à l'état du système à partir des principaux appareils AWS IoT Greengrass](#)

Cet objet contient les informations suivantes :

`enabled`

(Facultatif) Vous pouvez activer ou désactiver la télémétrie.

Valeur par défaut : `true`

`periodicAggregateMetricsIntervalSeconds`

(Facultatif) Intervalle (en secondes) pendant lequel le périphérique principal agrège les métriques.

Si vous définissez cette valeur en dessous de la valeur minimale prise en charge, le noyau utilise plutôt la valeur par défaut.

Minimum : `3600`

Valeur par défaut : `3600`

`periodicPublishMetricsIntervalSeconds`

(Facultatif) Durée (en secondes) entre laquelle le périphérique principal publie des métriques de télémétrie sur AWS Cloud

Si vous définissez cette valeur en dessous de la valeur minimale prise en charge, le noyau utilise plutôt la valeur par défaut.

Minimum : `86400`

Valeur par défaut : `86400`

`deploymentPollingFrequencySeconds`

(Facultatif) Période en secondes pendant laquelle demander les notifications de déploiement.

Valeur par défaut : `15`

`componentStoreMaxSizeBytes`

(Facultatif) Taille maximale sur le disque du magasin de composants, qui comprend les recettes de composants et les artefacts.

Par défaut : 10000000000 (10 Go)

`platformOverride`

(Facultatif) Un dictionnaire d'attributs identifiant la plate-forme de l'appareil principal. Utilisez-le pour définir des attributs de plate-forme personnalisés que les recettes de composants peuvent utiliser pour identifier le cycle de vie et les artefacts appropriés pour le composant. Par exemple, vous pouvez définir un attribut de capacité matérielle pour déployer uniquement l'ensemble minimal d'artefacts nécessaires à l'exécution d'un composant. Pour plus d'informations, consultez le [paramètre de plateforme du manifeste](#) dans la recette du composant.

Vous pouvez également utiliser ce paramètre pour remplacer les attributs os et les attributs de architecture plate-forme du périphérique principal.

`httpClient`

Ce paramètre est disponible dans les versions 2.5.0 et ultérieures de ce composant.

(Facultatif) Configuration du client HTTP pour le périphérique principal. Ces options de configuration s'appliquent à toutes les requêtes HTTP effectuées par ce composant. Si un périphérique principal fonctionne sur un réseau plus lent, vous pouvez augmenter ces délais d'expiration pour éviter que les requêtes HTTP n'expirent.

Cet objet contient les informations suivantes :

`connectionTimeoutMs`

(Facultatif) Durée (en millisecondes) d'attente avant l'ouverture d'une connexion avant l'expiration de la demande de connexion.

Par défaut : 2000 (2 secondes)

`socketTimeoutMs`

(Facultatif) Durée (en millisecondes) pendant laquelle les données doivent être transférées via une connexion ouverte avant l'expiration de la connexion.

Par défaut : 30000 (30 secondes)

deploymentConfigurationTimeSource

Ce paramètre est disponible dans les versions 2.15.0 et ultérieures de ce composant.

(Facultatif) L'horodatage à utiliser lors du traitement d'un déploiement. La valeur par défaut est `deploymentCreationTime`.

Cet objet contient les valeurs suivantes :

deploymentCreationTime

La valeur par défaut de `deploymentConfigurationTimeSource`. L'appareil utilise l'horodatage de création du déploiement pour résoudre les conflits de clés de configuration pendant le traitement. Lorsque ce comportement est sélectionné, la configuration du périphérique local détenue par le noyau peut avoir un horodatage supérieur à celui du déploiement entrant et rejette les modifications de configuration entrantes qui sont désormais considérées comme obsolètes.

deploymentProcessingTime

L'appareil utilise son horodatage local pour résoudre les conflits de clés de configuration lors du processus de déploiement. Lors du traitement, le périphérique met à jour les configurations en fonction de l'horodatage du traitement plutôt que de l'horodatage de création du déploiement. Ce comportement suppose que l'horloge de l'appareil est correctement calibrée.

Configurez ce paramètre de noyau dans l'image ou l'installation initiale de votre appareil plutôt que par le biais d'un déploiement lorsque vous souhaitez que les nouveaux appareils utilisent ce comportement lors de la première connexion. Utilisez l'["--init-config](#) option du programme d'installation de Nucleus Classic pour cette configuration.

Cette configuration initiale est essentielle car les appareils traitent plusieurs déploiements dans un ordre arbitraire. Sans configuration initiale appropriée, un appareil peut traiter les déploiements en utilisant le `deploymentCreationTime` comportement par défaut avant de recevoir le déploiement qui définit la configuration du noyau sur `deploymentProcessingTime`.

Exemple Exemple : mise à jour de la fusion de configurations

```
{
```

```
"iotRoleAlias": "GreengrassCoreTokenExchangeRoleAlias",
"networkProxy": {
  "noProxyAddresses": "http://192.168.0.1,www.example.com",
  "proxy": {
    "url": "http://my-proxy-server:1100",
    "username": "Mary_Major",
    "password": "pass@word1357"
  }
},
"mqtt": {
  "port": 443
},
"greengrassDataPlanePort": 443,
"jvmOptions": "-Xmx64m",
"runWithDefault": {
  "posixUser": "ggc_user:ggc_group"
}
}
```

Fichier journal local

Ce composant utilise le fichier journal suivant.

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

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

Pour consulter les journaux de ce composant

- Exécutez la commande suivante sur le périphérique principal pour afficher le fichier journal de ce composant en temps réel. Remplacez */greengrass/v2* ou *C:\greengrass\v2* par le chemin d'accès au dossier AWS IoT Greengrass racine.

Linux

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


Windows (PowerShell)

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

Journal des modifications

Le tableau suivant décrit les modifications apportées à chaque version du composant.

Version	Modifications
2.16.1	Corrections de bugs et améliorations • Ajoute une configuration pour les intervalles entre les tentatives d'identification après les échecs du service d'échange de jetons.
2.16.0	Nouvelles fonctionnalités • Nucleus prend désormais en charge les cgroups V2. Corrections de bugs et améliorations • Résout un problème selon lequel le noyau ne nettoyait pas les déploiements périmés lorsqu'un déploiement était annulé.
2.15.1	Corrections de bugs et améliorations • Résout un problème selon lequel le noyau ne parvenait pas à nettoyer les processus des composants après le délai d'arrêt de 30 secondes pendant les déploiements bootstrap.
2.15.0	Nouvelles fonctionnalités • Ajoute une fonction de télémétrie pour inclure des informations sur le système hôte, telles que les détails du processeur et du système d'exploitation. Pour plus d'informations, consultez la page du composant de télémétrie. • Ajoute <code>deploymentConfigurationTimeSource</code> une configuration. Pour plus d'informations, consultez la section Configuration des composants du noyau.

Version	Modifications
	Corrections de bugs et améliorations • Nucleus donne désormais la priorité à l'utilisation de versions et d'artefacts locaux par défaut, ce qui améliore la cohérence du déploiement et réduit les dépendances externes. Les utilisateurs peuvent modifier ce comportement en spécifiant d'autres préférences dans le document de déploiement. • Ajoute un nouveau journal d'avertissement lorsqu'un utilisateur Windows fourni ne correspond pas au jeu de caractères normalement accepté par Windows.
2.14.3	Corrections de bugs et améliorations • Permet au service d'échange de jetons de redémarrer en cas de modification de la configuration du port. • Résout un problème selon lequel le service d'état du parc n'envoie pas de messages de modification de l'état des composants pour les lambdas non épinglés s'ils ne sont pas déclenchés. • Résout un problème en raison duquel les composants ne sont pas correctement arrêtés lors du déploiement d'une nouvelle version de composant. • Résout un problème selon lequel les hooks d'injection des plugins intégrés étaient exécutés deux fois, ce qui entraînait des événements de cycle de vie supplémentaires et une journalisation dupliquée. • Améliore la journalisation du cycle de vie des composants sur les appareils Windows.
2.14.2	Corrections de bugs et améliorations • Résout un problème selon lequel un client HTTP n'est pas configuré avec une authentification mutuelle.
2.14.1	Corrections de bugs et améliorations • Résout un problème en raison duquel les composants n'étaient pas correctement arrêtés sur les nouvelles installations de Greengrass.

Version	Modifications
2.14.0	 Warning Cette version n'est plus disponible. Les améliorations apportées à cette version sont disponibles dans les versions ultérieures de ce composant. Nouvelles fonctionnalités • La nouvelle prise en charge des terminaux à double pile permet la communication IPv6 réseau. • Résilience améliorée contre les échecs de redémarrage du noyau et la corruption de Launch répertoires. Corrections de bugs et améliorations • Corrige les fuites de mémoire lors des fermetures PubSub d'abonnements IPC. • Corrige le cycle de vie d'exécution du composant lorsqu'il entre dans l'état ERRORED en raison du délai de démarrage lorsque la condition skipif est vraie. • Résout un problème en raison duquel le périphérique principal ne parvient pas à se connecter AWS IoT Core lorsque la politique TLS est définie sur TLS13 _1_3_2022_10.
2.13.0	Nouvelles fonctionnalités • Support du point de terminaison FIPS dans Nucleus. Corrections de bugs et améliorations • Annuler les améliorations apportées au déploiement : les déploiements peuvent désormais être annulés pendant la fusion d'une nouvelle configuration et en attendant le démarrage des services.
2.12,6	Corrections de bugs et améliorations • Résout un problème qui provoquait un crash au démarrage sur certains ARMv8 processeurs, notamment le Jetson Nano.

Version	Modifications
2,1,5	Corrections de bugs et améliorations • Résout un problème qui empêchait parfois l'annulation d'un déploiement lors de la restauration d'un composant précédemment défectueux présentant des dépendances strictes. • Résout un problème selon lequel le noyau ne publie pas de mises à jour de statut après le provisionnement de la flotte. • Ajoute de nouvelles tentatives pour l'GetDeploymentConfiguration API après l'obtention d'erreurs 404.
2.12.4	Corrections de bugs et améliorations • Résout un problème de blocage du noyau lors du démarrage sur certains appareils Linux.
2.12.3	 Warning Cette version n'est plus disponible. Les améliorations apportées à cette version sont disponibles dans les versions ultérieures de ce composant. Corrections de bugs et améliorations • Résout un problème selon lequel le noyau ne signalait pas l'état correct des composants après le redémarrage du noyau et pendant la restauration des composants. • Correction et amélioration de bogues généraux
2.12.2	Corrections de bugs et améliorations • Résout un problème en raison duquel les anciens journaux n'étaient pas nettoyés correctement. • Correction et amélioration de bogues généraux

Version	Modifications
2.12.1	Corrections de bugs et améliorations • Résout un problème selon lequel le noyau pouvait dupliquer les abonnements MQTT aux sujets de déploiement, ce qui entraînait une journalisation supplémentaire et des publications MQTT.
2.12.0	Nouvelles fonctionnalités • Vous permet d'exécuter les étapes du cycle de vie du bootstrap dans le cadre d'un déploiement rétroactif.
2.11.3	Corrections de bugs et améliorations • Résout un problème dans le noyau qui pouvait entraîner le démarrage incorrect d'un composant en cas d'échec de ses dépendances. Nouvelles fonctionnalités • Ajoute un type de point de terminaison s3 configurable.
2.11.2	Corrections de bugs et améliorations • Résout un problème dans le client Nucleus MQTT 5 qui pouvait apparaître hors ligne lorsqu'un grand nombre (> 50) d'abonnements sont utilisés. • Ajoute une nouvelle tentative en cas d'échec du protocole TCP du docker Dial.
2.11.1	Corrections de bugs et améliorations • Résout un problème selon lequel le noyau ne démarre pas en cas d'échec d'une tâche de démarrage et d'endommagement du fichier de métadonnées de déploiement. • Résout un problème selon lequel les composants Lambda à la demande ne sont pas signalés dans les mises à jour de l'état du déploiement. • Ajoute la prise en charge de la politique d'autorisation dupliquée IDs.

Version	Modifications
2.11.0	Nouvelles fonctionnalités • Vous permet d'annuler un déploiement local. • Vous permet de configurer une politique de gestion des défaillances pour un déploiement local. • Ajoute la prise en charge d'un plugin de spouleur de disque.
2.10.3	Corrections de bugs et améliorations • Résout un problème en raison duquel Greengrass ne s'abonne pas aux notifications de déploiement lorsqu'il utilise le fournisseur PKCS #11.
2.10.2	Corrections de bogues et améliorations • Permet d'analyser le cycle de vie des composants sans distinction majuscules/minuscules. • Résout un problème en raison duquel la variable d'environnement PATH n'était pas recréée correctement. • Corrige le codage de l'URI du proxy pour les composants, y compris le gestionnaire de flux pour les noms d'utilisateur contenant des caractères spéciaux.
2.10.1	Corrections de bogues et améliorations • Résout un problème susceptible de provoquer un crash au démarrage sur certains ARMv8 processeurs, notamment le Jetson Nano. • Greengrass ne ferme plus le standard d'un composant, ce qui rétablit le comportement d'avant la version 2.10.0

Version	Modifications
2.10.0	Nouvelles fonctionnalités • Ajoute la <code>interpolateComponentConfiguration</code> prise en charge de l'expression régulière vide. Greengrass interpole désormais à partir de l'objet de configuration racine. • Ajoute le support pour MQTT5. • Ajoute un mécanisme permettant de charger rapidement les composants du plugin sans les scanner. • Permet à Greengrass d'économiser de l'espace disque en supprimant les images Docker inutilisées. Corrections de bogues et améliorations • Résout un problème selon lequel la restauration laisse certaines valeurs de configuration inchangées lors d'un déploiement. • Résout un problème à cause duquel le noyau Greengrass valide une séquence de AWS domaine dans des points de terminaison de AWS données et des informations d'identification personnalisés. • Met à jour la résolution des dépendances multigroupes afin de résoudre à nouveau toutes les dépendances de groupe par le biais de AWS Cloud la négociation, au lieu de se limiter à la version active. Cette mise à jour supprime également le code d'erreur de déploiement <code>INSTALLED_COMPONENT_NOT_FOUND</code>. • Met à jour le noyau Greengrass pour éviter de télécharger des images Docker lorsqu'elles existent déjà localement. • Met à jour le noyau Greengrass pour redémarrer une étape d'installation d'un composant avant l'expiration du délai imparti. • Corrections et améliorations mineures supplémentaires.

Version	Modifications
2,9,6	Corrections de bogues et améliorations • Résout un problème selon lequel un déploiement de Greengrass échoue avec l'erreur <code>LAUNCH_DIRECTORY_CORRUPTED</code> et un redémarrage ultérieur de l'appareil ne permet pas de démarrer Greengrass. Cette erreur peut se produire lorsque vous déplacez l'appareil Greengrass entre plusieurs groupes d'objets lors de déploiements nécessitant le redémarrage de Greengrass.
2.9.5	Nouvelles fonctionnalités • Ajoute la prise en charge de la vérification des signatures du logiciel Greengrass Nucleus. Corrections de bogues et améliorations • Résout un problème d'échec d'un déploiement lorsque la région de métadonnées de recette locale ne correspond pas à la région de lancement du noyau Greengrass. Le noyau Greengrass renégocie désormais avec le cloud lorsque cela se produit. • Résout un problème à cause duquel le spouleur de messages MQTT se remplit et ne supprime jamais les messages. • Corrections et améliorations mineures supplémentaires.
2.9.4	Corrections de bogues et améliorations • Vérifie la présence d'un message nul avant de supprimer les messages QOS 0. • Tronque les valeurs détaillées du statut de la tâche si elles dépassent la limite de 1024 caractères. • Met à jour le script bootstrap pour Windows afin de lire correctement le chemin racine de Greengrass si ce chemin inclut des espaces. • Met à jour l'abonnement AWS IoT Core afin de supprimer les messages des clients si la réponse à l'abonnement n'a pas été envoyée. • Garantit que le noyau charge sa configuration à partir de fichiers de sauvegarde lorsque le fichier de configuration principal est endommagé ou manquant.

Version	Modifications
2.9.3	Corrections de bogues et améliorations • Garantit que les clients MQTT IDs ne sont pas dupliqués. • Améliore la lecture et l'écriture de fichiers afin d'éviter la corruption et de récupérer les données en cas de corruption. • Réessaie docker image pull en cas d'erreurs spécifiques liées au réseau. • Ajoute l'<code>noProxyAddresses</code> option de connexion MQTT.
2.9.2	Corrections de bogues et améliorations • Résout un problème selon lequel la configuration <code>interpolateComponentConfiguration</code> ne s'appliquait pas à un déploiement en cours. • Utilise OSHI pour répertorier tous les processus enfants.
2.9.1	Corrections de bogues et améliorations • Ajoute un correctif selon lequel Greengrass redémarre si un déploiement supprime un composant du plugin.
2.9.0	Nouvelles fonctionnalités • Permet de créer des sous-déploiements qui relancent les déploiements avec un plus petit sous-ensemble d'appareils. Cette fonctionnalité permet de tester et de résoudre les déploiements infructueux de manière plus efficace. Corrections de bogues et améliorations • Améliore la prise en charge des systèmes qui ne disposent pas de <code>useraddgroupadd</code>, et <code>usermod</code>. • Corrections et améliorations mineures supplémentaires.

Version	Modifications
2.8.1	Corrections de bogues et améliorations • Résout un problème en raison duquel les codes d'erreur de déploiement n'étaient pas générés correctement à cause des erreurs de l'API Greengrass. • Résout un problème selon lequel les mises à jour de l'état du parc envoient des informations inexactes lorsqu'un composant atteint un ERRORRED état au cours d'un déploiement. • Résout un problème en raison duquel les déploiements ne pouvaient pas être achevés lorsque Greengrass avait plus de 50 abonnements existants.

Version	Modifications
2.8.0	Nouvelles fonctionnalités • Met à jour le noyau Greengrass pour signaler une réponse sur l'état de santé du déploiement qui inclut des codes d'erreur détaillés en cas de problème lors du déploiement de composants sur un périphérique principal. Pour de plus amples informations, veuillez consulter Codes d'erreur de déploiement détaillés. • Met à jour le noyau Greengrass pour signaler une réponse sur l'état de santé d'un composant qui inclut des codes d'erreur détaillés lorsqu'un composant passe à l'état BROKEN ou ERRORRED. Pour de plus amples informations, veuillez consulter Codes d'état détaillés des composants. • Étend les champs des messages d'état afin d'améliorer les informations de disponibilité du cloud pour les appareils. • Améliore la robustesse du service d'état de la flotte. Corrections de bogues et améliorations • Permet à un composant défectueux de se réinstaller lorsque sa configuration change. • Résout un problème selon lequel un redémarrage du noyau pendant le déploiement du bootstrap entraîne l'échec d'un déploiement. • Résout un problème dans Windows où l'installation échoue lorsqu'un chemin racine contient des espaces. • Résout un problème selon lequel un composant arrêté lors d'un déploiement utilise le script d'arrêt de la nouvelle version. • Diverses améliorations en matière d'arrêt. • Corrections et améliorations mineures supplémentaires.

Version	Modifications
2.7.0	Nouvelles fonctionnalités • Met à jour le noyau Greengrass pour envoyer des mises à jour de statut au AWS IoT Greengrass cloud lorsque l'appareil principal effectue un déploiement local. • Ajoute la prise en charge des certificats clients signés par une autorité de certification (CA) personnalisée, auprès de laquelle l'autorité de certification n'est pas enregistrée AWS IoT. Pour utiliser cette fonctionnalité, vous pouvez définir la nouvelle option <code>greengrassDataPlaneEndpoint</code> de configuration <code>suriotdata</code>. Pour de plus amples informations, veuillez consulter Utiliser un certificat d'appareil signé par une autorité de certification privée. Corrections de bogues et améliorations • Résout un problème selon lequel le noyau Greengrass annule un déploiement dans certains scénarios lorsque le noyau est arrêté ou redémarré. Le noyau reprend désormais le déploiement après le redémarrage du noyau. • Met à jour le programme d'installation de Greengrass afin de respecter l'<code>--startargument</code> lorsque vous spécifiez de configurer le logiciel en tant que service système. • Met à jour le comportement de SubscribeToComponentUpdates pour définir l'ID de déploiement dans les événements où le noyau a mis à jour un composant. • Corrections et améliorations mineures supplémentaires.

Version	Modifications
2.6.0	Nouvelles fonctionnalités • Ajoute la prise en charge des caractères génériques MQTT lorsque vous vous abonnez à des sujets locaux publish/subscribe . Pour plus d'informations, consultez Publier/souscrire des messages locaux et SubscribeToTopic. • Ajoute la prise en charge des variables de recette dans les configurations de composants, autres que la variable de <i>component_dependency_name</i> :configuration: <i>json_pointer</i> recette. Vous pouvez utiliser ces variables de recettes lorsque vous définissez un composant DefaultConfiguration dans une recette ou lorsque vous configurez un composant dans un déploiement. Pour activer cette fonctionnalité, définissez l'option interpolateComponentConfiguration de configuration sur true. Pour plus d'informations, consultez Variables de recette et Utiliser des variables de recette dans les mises à jour de fusion. • Ajoute une prise en charge complète du * caractère générique dans les politiques d'autorisation de communication interprocessus (IPC). Vous pouvez désormais spécifier le * caractère dans une chaîne de ressources pour qu'il corresponde à n'importe quelle combinaison de caractères. Pour de plus amples informations, veuillez consulter Des caractères génériques dans les politiques d'autorisation. • Ajoute la prise en charge des composants personnalisés pour appeler les opérations IPC utilisées par la CLI Greengrass. Vous pouvez utiliser ces opérations IPC pour gérer les déploiements locaux, afficher les détails des composants et générer un mot de passe que vous pouvez utiliser pour vous connecter à la console de débogage locale. Pour plus d'informations, voir IPC : Gérer les déploiements et les composants locaux. Corrections de bogues et améliorations • Résout un problème selon lequel les composants dépendants ne réagissaient pas lorsque leurs dépendances matérielles redémarraient ou changeaient d'état dans certains scénarios.

Version	Modifications
	• Améliore les messages d'erreur que le périphérique principal envoie au service AWS IoT Greengrass cloud en cas d'échec d'un déploiement. • Résout un problème selon lequel le noyau Greengrass appliquait deux fois le déploiement d'un objet dans certains scénarios lors du redémarrage du noyau. • Corrections et améliorations mineures supplémentaires. Pour plus d'informations, consultez les versions sur GitHub.
2.5.6	Nouvelles fonctionnalités • Ajoute la prise en charge des modules de sécurité matériels qui utilisent des clés ECC. Vous pouvez utiliser un module de sécurité matérielle (HSM) pour stocker en toute sécurité la clé privée et le certificat de l'appareil. Pour de plus amples informations, veuillez consulter Intégration de sécurité matérielle. Corrections de bogues et améliorations • Résout un problème selon lequel le déploiement ne se termine jamais lorsque vous déployez un composant avec un script d'installation défectueux dans certains scénarios. • Améliore les performances au démarrage. • Corrections et améliorations mineures supplémentaires.

Version	Modifications
2.5.5	Nouvelles fonctionnalités • Ajoute la variable d'GG_ROOT_CA_PATH environnement pour les composants, afin que vous puissiez accéder au certificat de l'autorité de certification (CA) racine dans les composants personnalisés. Corrections de bogues et améliorations • Ajoute la prise en charge des appareils Windows qui utilisent une langue d'affichage autre que l'anglais. • Met à jour la façon dont le noyau Greengrass analyse les arguments booléens du programme d'installation, afin que vous puissiez spécifier un argument booléen sans valeur booléenne pour spécifier une valeur. true Par exemple, vous pouvez désormais spécifier le provisionnement automatique des --provision true ressources --provision au lieu d'installer. • Résout un problème selon lequel le périphérique principal ne signalait pas son état au service AWS IoT Greengrass cloud après le provisionnement dans certains scénarios. • Corrections et améliorations mineures supplémentaires.
2.5.4	Corrections de bogues et améliorations • Correction et amélioration de bogues généraux
2.5.3	Nouvelles fonctionnalités • Ajoute la prise en charge de l'intégration de la sécurité matérielle. Vous pouvez utiliser un module de sécurité matérielle (HSM) pour stocker en toute sécurité la clé privée et le certificat de l'appareil. Pour de plus amples informations, veuillez consulter Intégration de sécurité matérielle. Corrections de bogues et améliorations • Résout un problème lié aux exceptions d'exécution lorsque le noyau établit des connexions MQTT avec AWS IoT Core.

Version	Modifications
2.5.2	Corrections de bogues et améliorations • Résout un problème selon lequel, après les mises à jour du noyau Greengrass, le service Windows ne redémarre pas une fois que vous l'avez arrêté ou redémarré l'appareil.
2.5.1	 Warning Cette version n'est plus disponible. Les améliorations apportées à cette version sont disponibles dans les versions ultérieures de ce composant. Corrections de bogues et améliorations • Ajoute la prise en charge des versions 32 bits de l'environnement d'exécution Java (JRE) sous Windows. • Modifie le comportement de suppression des groupes d'objets pour les principaux appareils dont la AWS IoT politique n'accorde pas l'<code>greengrass:ListThingGroupsForCoreDevice</code> autorisation. Dans cette version, le déploiement se poursuit, enregistre un avertissement et ne supprime aucun composant lorsque vous supprimez le périphérique principal d'un groupe d'objets. Pour de plus amples informations, veuillez consulter Déployer AWS IoT Greengrass des composants sur des appareils. • Résout un problème lié aux variables d'environnement système que le noyau de Greengrass met à la disposition des processus des composants de Greengrass. Vous pouvez désormais redémarrer un composant pour qu'il utilise les dernières variables d'environnement système.

Version	Modifications
2.5.0	Nouvelles fonctionnalités • Ajoute la prise en charge des appareils principaux qui exécutent Windows. • Modifiez le comportement de la suppression de groupes d'objets. Avec cette version, vous pouvez supprimer un périphérique principal d'un groupe d'objets pour désinstaller les composants de ce groupe d'objets lors du prochain déploiement. À la suite de cette modification, la AWS IoT politique d'un appareil principal doit être <code>greengrass:ListThingGroupsForCoreDevice</code> autorisée. Si vous avez utilisé le programme d'installation du logiciel AWS IoT Greengrass Core pour provisionner des ressources, la AWS IoT politique par défaut <code>l'authorisegreengrass:*</code>, y compris cette autorisation. Pour de plus amples informations, veuillez consulter Authentification et autorisation de l'appareil pour AWS IoT Greengrass. • Ajoute la prise en charge des configurations de proxy HTTPS. Pour de plus amples informations, veuillez consulter Connexion au port 443 ou via un proxy réseau. • Ajoute le nouveau paramètre <code>windowsUser</code> de configuration. Vous pouvez utiliser ce paramètre pour spécifier l'utilisateur par défaut à utiliser pour exécuter les composants sur un périphérique principal Windows. Pour de plus amples informations, veuillez consulter Configurer l'utilisateur qui exécute les composants. • Ajoute les nouvelles options <code>httpClient</code> de configuration que vous pouvez utiliser pour personnaliser les délais d'expiration des requêtes HTTP afin d'améliorer les performances sur les réseaux lents. Pour plus d'informations, consultez le paramètre de configuration HttpClient. Corrections de bogues et améliorations • Corrige l'option de cycle de vie du bootstrap permettant de redémarrer le périphérique principal à partir d'un composant. • Ajoute la prise en charge des traits d'union dans les variables de recette. • Corrige l'autorisation IPC pour les composants de la fonction Lambda à la demande.

Version	Modifications
	• Améliore les messages de journal et fait passer les journaux non critiques du DEBUG niveau INFO au niveau supérieur, afin que les journaux soient plus utiles. • Supprime l'iot:DescribeCertificate autorisation du rôle d'échange de jetons par défaut créé par le noyau Greengrass lorsque vous installez le logiciel AWS IoT Greengrass Core avec provisionnement automatique. Cette autorisation n'est pas utilisée par le noyau Greengrass. • Résout un problème selon lequel le script de provisionnement automatique ne nécessite pas d'iam:GetPolicy autorisation s'il iam:CreatePolicy est disponible pour la même politique. • Corrections et améliorations mineures supplémentaires.

Version	Modifications
2.4.0	Nouvelles fonctionnalités • Ajoute la prise en charge des limites de ressources du système. Vous pouvez configurer l'utilisation maximale du processeur et de la RAM que les processus de chaque composant peuvent utiliser sur le périphérique principal. Pour de plus amples informations, veuillez consulter Configuration des limites de ressources système pour les composants. • Ajoute des opérations IPC pour suspendre et reprendre les composants. Pour plus d'informations, consultez PauseComponent et ResumeComponent. • Ajoute la prise en charge des plugins de provisionnement. Vous pouvez spécifier un fichier JAR à exécuter pendant l'installation afin de fournir les AWS ressources nécessaires à un périphérique principal Greengrass. Le noyau Greengrass inclut une interface que vous pouvez implémenter pour développer des plugins de provisionnement personnalisés. Pour de plus amples informations, veuillez consulter Installez le logiciel AWS IoT Greengrass Core avec un provisionnement personnalisé des ressources. • Ajoute l'<code>thing-name-policy</code> argument facultatif au programme d'installation du logiciel AWS IoT Greengrass Core. Vous pouvez utiliser cette option pour spécifier une AWS IoT politique existante ou personnalisée lorsque vous installez le logiciel AWS IoT Greengrass Core avec provisionnement automatique des ressources. Corrections de bogues et améliorations • Met à jour la configuration de journalisation au démarrage. Cela résout un problème en raison duquel la configuration de journalisation n'était pas appliquée au démarrage. • Met à jour le lien symbolique du chargeur Nucleus pour qu'il pointe vers le magasin de composants dans le dossier racine de Greengrass lors de l'installation. Cette mise à jour vous permet de supprimer le fichier JAR et les autres artefacts Nucleus que vous téléchargez lorsque vous installez le logiciel AWS IoT Greengrass Core.

Version	Modifications
	• Corrections et améliorations mineures supplémentaires. Pour plus d'informations, consultez les versions sur GitHub.
2.3.0	Nouvelles fonctionnalités • Prend en charge les documents de configuration de déploiement jusqu'à 10 Mo, au lieu de 7 Ko (pour les déploiements ciblant des objets) ou 31 Ko (pour les déploiements ciblant des groupes d'objets). Pour utiliser cette fonctionnalité, la AWS IoT politique d'un appareil principal doit autoriser <code>greengrass:GetDeploymentConfiguration</code> autorisation. Si vous avez utilisé le programme d'installation du logiciel AWS IoT Greengrass Core pour provisionner des ressources, la AWS IoT politique de votre appareil principal le permet <code>greengrass:*</code>, y compris cette autorisation. Pour de plus amples informations, veuillez consulter Authentification et autorisation de l'appareil pour AWS IoT Greengrass. • Ajoute la variable de <code>iot:thingName</code> recette. Vous pouvez utiliser cette variable de recette pour obtenir le nom de l'appareil AWS IoT principal d'une recette. Pour de plus amples informations, veuillez consulter Variables de recette. Corrections de bogues et améliorations • Corrections et améliorations mineures supplémentaires. Pour plus d'informations, consultez les versions sur GitHub.
2.2.0	Nouvelles fonctionnalités • Ajoute des opérations IPC pour la gestion des ombres locales. Corrections de bogues et améliorations • Réduit la taille du fichier JAR. • Réduit l'utilisation de la mémoire. • Résout les problèmes où la configuration du journal n'était pas mise à jour dans certains cas. • Corrections et améliorations mineures supplémentaires. Pour plus d'informations, consultez les versions sur GitHub.

Version	Modifications
2.1.0	Nouvelles fonctionnalités • Prend en charge le téléchargement d'images Docker à partir de référentiels privés sur Amazon ECR. • Ajoute les paramètres suivants pour personnaliser la configuration MQTT sur les appareils principaux : • <code>maxInFlightPublishes</code> — Le nombre maximum de messages MQTT QoS 1 non reconnus qui peuvent être envoyés en même temps. • <code>maxPublishRetry</code> — Le nombre maximum de fois que vous pouvez réessayer un message qui n'est pas publié. • Ajoute le paramètre de <code>fleetstatusservice</code> configuration pour configurer l'intervalle auquel le périphérique principal publie l'état du périphérique dans le AWS Cloud. • Corrections et améliorations mineures supplémentaires. Pour plus d'informations, consultez les versions sur GitHub. Corrections de bogues et améliorations • Résout un problème en raison duquel les déploiements fictifs étaient dupliqués lors du redémarrage du noyau. • Résout un problème qui provoquait le blocage du noyau lorsqu'il rencontrait une exception de chargement de service. • Améliore la résolution des dépendances entre les composants en cas d'échec d'un déploiement incluant une dépendance circulaire. • Résout un problème qui empêchait le redéploiement d'un composant de plug-in s'il avait été précédemment supprimé du périphérique principal. • Résolution d'un problème en raison duquel la variable d'HOMEenvironnement était définie <code>/greengrass/v2 /work</code> dans le répertoire des composants Lambda ou des composants exécutés en tant qu'utilisateur root. La HOME variable est désormais correctement définie dans le répertoire de base de l'utilisateur qui exécute le composant. • Corrections et améliorations mineures supplémentaires. Pour plus d'informations, consultez les versions sur GitHub.

Version	Modifications
2.0.5	Corrections de bogues et améliorations • Achemine correctement le trafic via un proxy réseau configuré lors du téléchargement des composants AWS fournis. • Utilisez le point de terminaison du plan de données Greengrass approprié dans les régions de AWS Chine.

Version	Modifications
2.0.4	Nouvelles fonctionnalités • Active le trafic HTTPS sur le port 443. Vous pouvez utiliser le nouveau paramètre de <code>greengrassDataPlanePort</code> configuration de la version 2.0.4 du composant Nucleus pour configurer la communication HTTPS afin qu'elle passe par le port 443 au lieu du port par défaut 8443. Pour de plus amples informations, veuillez consulter Configuration du protocole HTTPS sur le port 443. • Ajoute la variable de recette du chemin de travail. Vous pouvez utiliser cette variable de recette pour obtenir le chemin d'accès aux dossiers de travail des composants, que vous pouvez utiliser pour partager des fichiers entre les composants et leurs dépendances. Pour plus d'informations, consultez la variable de recette du chemin de travail. Corrections de bogues et améliorations • Empêche la création de la politique de rôle d'échange de jetons Gestion des identités et des accès AWS (IAM) si une stratégie de rôle existe déjà. À la suite de cette modification, le programme d'installation nécessite désormais le <code>iam:GetPolicy</code> et <code>sts:GetCallerIdentity</code> lorsqu'il est exécuté avec <code>--provision true</code>. Pour de plus amples informations, veuillez consulter Politique IAM minimale permettant au programme d'installation de provisionner les ressources. • Gère correctement l'annulation d'un déploiement qui n'a pas encore été enregistré avec succès. • Met à jour la configuration pour supprimer les anciennes entrées avec des horodatages plus récents lors de l'annulation d'un déploiement. • Corrections et améliorations mineures supplémentaires. Pour plus d'informations, consultez les versions sur GitHub.
2.0.3	Première version.

Greengrass Nucleus Lite

Le Greengrass nucleus lite (`aws.greengrass.NucleusLite`) est un environnement d'exécution pour les appareils à périphérie restreinte, optimisé pour un encombrement mémoire minimal (utilise moins de 5 Mo de RAM). Il a été introduit avec AWS IoT Greengrass la version 2.14.0 et est rétrocompatible avec les composants AWS IoT Greengrass génériques, l'API Greengrass V2 et le SDK.

Le Greengrass nucleus lite est proposé comme alternative au [noyau Greengrass commun](#) (`aws.greengrass.Nucleus`) et peut être utilisé dans des flottes hétérogènes d'appareils Greengrass.

Rubriques

- [Versions](#)
- [Système d'exploitation](#)
- [Exigences](#)
- [Compatibilité](#)
- [Téléchargement et installation](#)
- [Configuration](#)
- [Fichier journal local](#)
- [Journal des modifications](#)

Versions

Les versions de ce composant sont les suivantes :

- 2.3.x
- 2.2.x
- 2.1.x
- 2.0.x

Système d'exploitation

Ce composant peut être installé sur les appareils principaux qui exécutent les systèmes d'exploitation suivants :

- Linux (distributions avec systemd)

Pour plus d'informations, voir [Greengrass nucleus](#).

Exigences

Les appareils doivent répondre à certaines exigences pour installer et exécuter le AWS IoT Greengrass Nucleus Lite et le logiciel AWS IoT Greengrass Core. Pour plus d'informations, consultez le [guide de configuration](#).

- 5 Mo d'espace RAM pour le runtime du noyau.
- 5 Mo de stockage (disque/flash).

Les dépendances supplémentaires du système sont documentées dans le [guide de configuration](#).

Le composant Greengrass nucleus est compatible pour fonctionner dans un VPC. Pour déployer ce composant dans un VPC, les éléments suivants sont requis :

- Le noyau Greengrass doit être connecté aux AWS IoT données, aux AWS IoT informations d'identification et à Amazon S3.

Compatibilité

Le AWS IoT Greengrass Nucleus Lite est compatible avec l'API AWS IoT Greengrass v2 (sous-ensemble de) et pris en charge SDKs. Cela ne dépend pas d'un langage spécifique runtimes/VMs , mais les composants ajoutés à un déploiement peuvent nécessiter la présence de runtimes spécifiques (par exemple : Java JVM, Python). Pour plus d'informations sur les fonctionnalités prises en charge par Greengrass nucleus lite, consultez. [Compatibilité des fonctionnalités de Greengrass](#)

Téléchargement et installation

Vous pouvez télécharger un package apt, le [compiler à partir des sources](#), [utiliser une couche Yocto](#) ou [télécharger une image Yocto prédéfinie pour un](#) appareil compatible (par exemple,). RaspberryPi Depuis la [AWS IoT Core console](#), vous pourrez télécharger un kit de connexion contenant toutes les informations d'identification et la configuration initiale de votre appareil. Les instructions d'installation sont incluses dans chaque méthode de distribution spécifique.

Vous pouvez également suivre un tutoriel pour installer le AWS IoT Greengrass Nucleus Lite et découvrir le développement des composants Greengrass. Pour de plus amples informations, veuillez consulter [Tutoriel : Débuter avec AWS IoT Greengrass V2](#).

Configuration

Le noyau fournit les paramètres [de configuration](#) suivants. Certains paramètres nécessitent le redémarrage du logiciel AWS IoT Greengrass Core pour être pris en compte.

`iotRoleAlias`

Alias de AWS IoT rôle qui pointe vers un rôle IAM d'échange de jetons. Le fournisseur AWS IoT d'informations d'identification assume ce rôle pour permettre au dispositif principal de Greengrass d'interagir avec AWS les services. Pour plus d'informations, voir [Autoriser les appareils principaux à interagir avec AWS les services](#).

`iotDataEndpoint`

Le point AWS IoT de terminaison de données pour votre Compte AWS.

`iotCredEndpoint`

Le point de terminaison des informations d' AWS IoT identification de votre Compte AWS.

`greengrassDataPlanePort`

Port à utiliser pour les connexions au plan de données. Pour plus d'informations, voir [Connect sur le port 443 ou via un proxy réseau](#).

Important

Vous devez spécifier un port sur lequel l'appareil peut établir des connexions sortantes. Si vous spécifiez un port bloqué, l'appareil ne pourra pas se connecter pour AWS IoT Greengrass recevoir des déploiements. Sélectionnez l'une des options suivantes :

- 443
- 8443
- Valeur par défaut : 8443

`awsRegion`

Le Région AWS à utiliser.

runWithDefault

L'utilisateur du système à utiliser pour exécuter les composants.

Important

Lorsque vous déployez une modification de ce paramètre de configuration, le logiciel AWS IoT Greengrass Core redémarre pour que la modification prenne effet.

Cet objet contient les informations suivantes :

posixUser

Le nom ou l'ID de l'utilisateur du système et, éventuellement, du groupe système que le périphérique principal utilise pour exécuter des composants génériques. Spécifiez l'utilisateur et le groupe en les séparant par deux points (:) au format suivant : `user:group`. Le groupe est facultatif. Si vous ne spécifiez aucun groupe, le logiciel AWS IoT Greengrass Core utilise le groupe principal pour l'utilisateur. Par exemple, vous définissez `ggc_user` ou `ggc_user:ggc_group`. Pour de plus amples informations, veuillez consulter [Configurer l'utilisateur qui exécute les composants](#).

networkProxy

(Facultatif) Le proxy réseau à utiliser pour toutes les connexions. Pour de plus amples informations, veuillez consulter [Connexion au port 443 ou via un proxy réseau](#).

Important

Lorsque vous déployez une modification de ce paramètre de configuration, la modification prendra effet au prochain redémarrage du logiciel AWS IoT Greengrass Core.

Cet objet contient les informations suivantes :

noProxyAddresses

(Facultatif) Liste séparée par des virgules d'adresses IP ou de noms d'hôtes exemptés du proxy.

proxy

Le proxy auquel se connecter. Cet objet contient les informations suivantes :

url

URL du serveur proxy au format `http://host:port`.

- `scheme`— Le schéma, qui doit l'être `http`.
- `host`— Le nom d'hôte ou l'adresse IP du serveur proxy.
- `port`— (Facultatif) Le numéro de port. Si vous ne spécifiez pas le port, le périphérique principal de Greengrass utilise la valeur par défaut suivante :
 - `http`— 80

Fichier journal local

Les messages sont enregistrés sur stdout et les fichiers journaux sont gérés par systemd.

Pour consulter les journaux de ce composant

- `journalctl` À utiliser pour consulter les journaux.

Journal des modifications

Version	Modifications
2.3.2	Corrections de bogues et améliorations • Met à jour le fichier de version pour un signalement correct des versions de Nucleus.
2.3.1	Corrections de bogues et améliorations • GG ne tentera pas de mettre à jour l'état de déploiement des tâches annulées. • Assurez-vous que le DIR de <code>fdopendir</code> est fermé. • Le provisionnement de la flotte déclenchera désormais le remplacement des fichiers de certificats à chaque exécution. • Autres corrections de bugs mineurs.
2.3.0	Nouvelles fonctionnalités • Support pour l'utilisation de TPM 2.0 pour l'autorisation IoT Core MQTT.

Version	Modifications
	• Les exemples de paquets apt supportent désormais davantage de systèmes d'exploitation : Ubuntu 22.04, Ubuntu 24.04, Debian 12 et Debian 13. • RestartComponent L'IPC est désormais pris en charge. Corrections de bogues et améliorations • Les déploiements locaux ne nécessitent plus d'accès à Internet. • GetConfiguration a été mis à jour pour correspondre au comportement d'exécution de Greengrass Nucleus. (Changement radical) • Correction et amélioration de bogues généraux
2.2.2	Corrections de bogues et améliorations • Corrige les déploiements révisés contenant des versions de composants inchangées qui échouaient lorsqu'un composant inchangé avait un exécutable en cours d'exécution en tant qu'artefact. • Corrige l'interpolation des variables de recette lorsque la valeur interpolée comporte plus de quatre sous-objets imbriqués. • Corrige l'interpolation des variables de recette, y compris les guillemets et les caractères spéciaux du shell. • Corrige la fuite de fichiers FDS en cas d'échec des connexions MQTT.
2.2.1	Corrections de bogues et améliorations • Résout un problème en raison duquel le noyau ne parvient pas à obtenir les informations d'identification TES.
2.2.0	Nouvelles fonctionnalités • Ajoute la prise en charge de l'artefact URIs d'image du conteneur. Corrections de bogues et améliorations • Correction et amélioration de bogues généraux

Version	Modifications
2.1.0	Nouvelles fonctionnalités • Ajoute le support du proxy HTTP qui peut être configuré à l'aide de l'option <code>networkProxy</code> de configuration. Corrections de bogues et améliorations • Réduit l'exigence <code>libcurl</code> de 7.86 à 7.82 pour s'adapter aux appareils exécutant des versions antérieures <code>libcurl</code>. • Met à jour <code>journalctl</code> les journaux afin qu'ils soient attribués aux composants plutôt qu'à Recipe-Runner. • Améliore les réponses aux erreurs pour les appels IPC. • Ajoute de nouvelles tentatives pour les tentatives de téléchargement S3 pour les artefacts de composants génériques. • Correctifs de bogues mineurs.
2.0.2	Corrections de bogues et améliorations • Corrige les dépendances des paquets apt à inclure <code>cgroup-tools</code>.
2.0.1	Corrections de bogues et améliorations • Ajoute la prise en charge de l'interpolation des variables de recette pour la section de délai d'expiration de la recette Greengrass. • Ajoute le support de la commande <code>ValidateAuthorizationToken</code> IPC pour le gestionnaire de flux. • Corrige les avertissements liés au provisionnement de la flotte. • Ajoute une nouvelle tentative et une interruption à l'écouteur de tâches. • Correction et amélioration de bogues généraux
2.0.0	Première version.

Authentification de l'appareil client

Le composant d'authentification du dispositif client (`aws.greengrass.clientdevices.Auth`) authentifie les dispositifs clients et autorise les actions du dispositif client.

 Note

Les appareils clients sont des appareils IoT locaux qui se connectent à un appareil principal de Greengrass pour envoyer des messages MQTT et des données à traiter. Pour de plus amples informations, veuillez consulter [Interagissez avec les appareils IoT locaux](#).

Rubriques

- [Versions](#)
- [Type](#)
- [Système d'exploitation](#)
- [Prérequis](#)
- [Dépendances](#)
- [Configuration](#)
- [Fichier journal local](#)
- [Journal des modifications](#)

Versions

 Note

La version 2.3.0 d'authentification du périphérique client n'est plus disponible. Nous vous recommandons vivement de passer à la version 2.3.1 ou ultérieure de l'authentification du périphérique client.

Les versions de ce composant sont les suivantes :

- 2,5.x
- 2.4.x
- 2.3.x
- 2.2.x
- 2,1x

- 2,0.x

Type

Ce composant est un composant de plugin (`aws.greengrass.plugin`). Le [noyau Greengrass](#) exécute ce composant dans la même machine virtuelle Java (JVM) que le noyau. Le noyau redémarre lorsque vous modifiez la version de ce composant sur le périphérique principal.

Ce composant utilise le même fichier journal que le noyau Greengrass. Pour de plus amples informations, veuillez consulter [AWS IoT Greengrass Journaux de surveillance](#).

Pour de plus amples informations, veuillez consulter [Types de composants](#).

Système d'exploitation

Ce composant peut être installé sur les appareils principaux qui exécutent les systèmes d'exploitation suivants :

- Linux
- Windows

Prérequis

Ce composant répond aux exigences suivantes :

- Le [rôle de service Greengrass](#) doit être associé à votre autorisation Compte AWS et vous accorder cette autorisation. `iot:DescribeCertificate`
- La AWS IoT politique de l'appareil principal doit autoriser les autorisations suivantes :
 - `greengrass:GetConnectivityInfo`, où les ressources incluent l'ARN du périphérique principal qui exécute ce composant
 - `greengrass:VerifyClientDeviceIoTCertificateAssociation`, où les ressources incluent le nom de ressource Amazon (ARN) de chaque appareil client connecté au périphérique principal
 - `greengrass:VerifyClientDeviceIdentity`
 - `greengrass:PutCertificateAuthorities`
 - `iot:Publish`, où les ressources incluent l'ARN de la rubrique MQTT suivante :
 - `$aws/things/coreDeviceThingName*-gci/shadow/get`

- `iot:Subscribe`, où les ressources incluent les ARNs filtres thématiques MQTT suivants :
 - `$aws/things/coreDeviceThingName*-gci/shadow/update/delta`
 - `$aws/things/coreDeviceThingName*-gci/shadow/get/accepted`
- `iot:Receive`, où les ressources incluent les ARNs rubriques MQTT suivantes :
 - `$aws/things/coreDeviceThingName*-gci/shadow/update/delta`
 - `$aws/things/coreDeviceThingName*-gci/shadow/get/accepted`

Pour plus d'informations, consultez [AWS IoT politiques relatives aux opérations du plan de données](#) et [AWS IoT Politique minimale de prise en charge des appareils clients](#).

- (Facultatif) Pour utiliser l'authentification hors ligne, le rôle Gestion des identités et des accès AWS (IAM) utilisé par le AWS IoT Greengrass service doit contenir l'autorisation suivante :
 - `greengrass:ListClientDevicesAssociatedWithCoreDevice` pour permettre au périphérique principal de répertorier les clients pour l'authentification hors ligne.
- Le composant d'authentification du périphérique client est compatible pour s'exécuter dans un VPC. Pour déployer ce composant dans un VPC, les éléments suivants sont requis.
 - Le composant d'authentification de l'appareil client doit disposer d'une connectivité AWS IoT data, d' AWS IoT informations d'identification et d'Amazon S3.

Points de terminaison et ports

Ce composant doit être capable d'effectuer des demandes sortantes vers les points de terminaison et les ports suivants, en plus des points de terminaison et des ports requis pour le fonctionnement de base. Pour de plus amples informations, veuillez consulter [Autoriser le trafic des appareils via un proxy ou un pare-feu](#).

Endpoint	Port	Obligatoire	Description
<code>iot.<i>region</i>.amazonaws.com</code>	443	Oui	Utilisé pour obtenir des informations sur les AWS IoT certificats d'objets.

Dépendances

Lorsque vous déployez un composant, il déploie AWS IoT Greengrass également des versions compatibles de ses dépendances. Cela signifie que vous devez satisfaire aux exigences relatives au composant et à toutes ses dépendances pour réussir le déploiement du composant. Cette section répertorie les dépendances des [versions publiées](#) de ce composant et les contraintes de version sémantiques qui définissent les versions des composants pour chaque dépendance. Vous pouvez également consulter les dépendances de chaque version du composant dans la [AWS IoT Greengrass console](#). Sur la page de détails du composant, recherchez la liste des dépendances.

2.5.5

Le tableau suivant répertorie les dépendances pour la version 2.5.5 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,6,0$ $< 2,17,0$	Flexible

2.5.4

Le tableau suivant répertorie les dépendances pour la version 2.5.4 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,6,0$ $< 2,16,0$	Flexible

2.5.2 – 2.5.3

Le tableau suivant répertorie les dépendances pour les versions 2.5.2 et 2.5.3 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,6,0$ $< 2,15,0$	Flexible

2.5.1

Le tableau suivant répertorie les dépendances pour la version 2.5.1 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,6,0 <2,14,0	Flexible

2.4.4 - 2.5.0

Le tableau suivant répertorie les dépendances pour la version 2.4.4 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,6,0 <2,13,0	Flexible

2.4.3

Le tableau suivant répertorie les dépendances pour la version 2.4.3 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,6,0 <2,12,0	Flexible

2.4.1 and 2.4.2

Le tableau suivant répertorie les dépendances pour les versions 2.4.1 et 2.4.2 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,6,0 <2,11,0	Flexible

2.3.0 – 2.4.0

Le tableau suivant répertorie les dépendances pour les versions 2.3.0 à 2.4.0 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,6,0 <2,1,0	Flexible

2.3.0

Le tableau suivant répertorie les dépendances pour la version 2.3.0 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,6,0 < 2,1,0$	Flexible

2.2.3

Le tableau suivant répertorie les dépendances pour la version 2.2.3 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,6,0 \leq 2,9,0$	Flexible

2.2.2

Le tableau suivant répertorie les dépendances pour la version 2.2.2 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,6,0 \leq 2,8,0$	Flexible

2.2.1

Le tableau suivant répertorie les dépendances pour la version 2.2.1 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,6,0 < 2,8,0$	Flexible

2.2.0

Le tableau suivant répertorie les dépendances pour la version 2.2.0 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,6,0 <2,7,0	Flexible

2.1.0

Le tableau suivant répertorie les dépendances pour la version 2.1.0 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,2,0 <2,7,0	Flexible

2.0.4

Le tableau suivant répertorie les dépendances pour la version 2.0.4 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,2,0 <2,6,0	Flexible

2.0.2 and 2.0.3

Le tableau suivant répertorie les dépendances pour les versions 2.0.2 et 2.0.3 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,2,0 <2,5,0	Flexible

2.0.1

Le tableau suivant répertorie les dépendances pour la version 2.0.1 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,2,0 <2,4,0	Flexible

2.0.0

Le tableau suivant répertorie les dépendances pour la version 2.0.0 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,2,0 <2,3,0	Flexible

Pour plus d'informations sur les dépendances des composants, consultez la [référence de la recette des composants](#).

Configuration

Ce composant fournit les paramètres de configuration suivants que vous pouvez personnaliser lorsque vous déployez le composant.

Note

L'autorisation d'abonnement est évaluée lors d'une demande d'abonnement du client adressée au courtier MQTT local. Si l'autorisation d'abonnement existante du client est révoquée, le client ne pourra plus s'abonner à un sujet. Il continuera toutefois à recevoir des messages provenant de tous les sujets précédemment abonnés. Pour éviter ce comportement, le courtier MQTT local doit être redémarré après avoir révoqué l'autorisation d'abonnement afin de forcer la réautorisation des clients.

Pour le composant du courtier MQTT 5 (EMQX), mettez à jour la `restartIdentifier` configuration pour redémarrer le courtier MQTT 5.

Pour le composant courtier MQTT 3.1.1 (Moquette), il redémarre chaque semaine par défaut lorsque le certificat du serveur change, forçant les clients à se réautoriser. Vous pouvez forcer un redémarrage soit en modifiant les informations de connectivité (adresses IP) du périphérique principal, soit en effectuant un déploiement pour supprimer le composant broker, puis le déployer à nouveau ultérieurement.

v2.5.0 – 2.5.4

deviceGroups

Les groupes d'appareils sont des groupes d'appareils clients autorisés à se connecter et à communiquer avec un appareil principal. Utilisez des règles de sélection pour identifier les

groupes d'appareils clients et définissez des politiques d'autorisation des appareils clients qui spécifient les autorisations pour chaque groupe d'appareils.

Cet objet contient les informations suivantes :

`formatVersion`

Version du format pour cet objet de configuration.

Sélectionnez parmi les options suivantes :

- 2021-03-05

`definitions`

Les groupes de périphériques pour cet appareil principal. Chaque définition spécifie une règle de sélection pour évaluer si un appareil client est membre du groupe. Chaque définition spécifie également la politique d'autorisation à appliquer aux appareils clients qui répondent à la règle de sélection. Si un appareil client est membre de plusieurs groupes d'appareils, les autorisations de l'appareil sont comprises dans la politique d'autorisation de chaque groupe.

Cet objet contient les informations suivantes :

groupNameKey

Nom de ce groupe d'appareils. *groupNameKey* Remplacez-le par un nom qui vous aide à identifier ce groupe d'appareils.

Cet objet contient les informations suivantes :

`selectionRule`

Requête qui indique quels appareils clients sont membres de ce groupe d'appareils. Lorsqu'un dispositif client se connecte, le dispositif principal évalue cette règle de sélection pour déterminer si le dispositif client est membre de ce groupe de dispositifs. Si le dispositif client est membre, le dispositif principal utilise la politique de ce groupe d'appareils pour autoriser les actions du dispositif client.

Chaque règle de sélection comprend au moins une clause de règle de sélection, qui est une requête d'expression unique pouvant correspondre aux dispositifs clients. Les règles de sélection utilisent la même syntaxe de requête que l'indexation des AWS IoT flottes. Pour plus d'informations sur la syntaxe des règles de sélection, consultez la section [Syntaxe des requêtes d'indexation du AWS IoT parc](#) dans le Guide du AWS IoT Core développeur.

Utilisez le * caractère générique pour associer plusieurs appareils clients à une seule clause de règle de sélection. Vous pouvez utiliser ce caractère générique au début et à la fin du nom de l'objet pour faire correspondre les appareils clients dont le nom commence ou se termine par la chaîne que vous spécifiez. Vous pouvez également utiliser ce caractère générique pour correspondre à tous les appareils clients.

 Note

Pour sélectionner une valeur contenant deux points (:), éliminez les deux points par une barre oblique inverse (\). Dans les formats tels que JSON, vous devez éviter les barres obliques inversées. Vous devez donc saisir deux barres obliques inversées avant le caractère deux-points. Par exemple, spécifiez `thingName: MyTeam\\:ClientDevice1` de sélectionner un objet dont le nom est `MyTeam:ClientDevice1`.

Vous pouvez spécifier le sélecteur suivant :

- `thingName`— Le nom de l' AWS IoT objet d'un appareil client.

Exemple Exemple de règle de sélection

La règle de sélection suivante correspond aux appareils clients dont les noms sont `MyClientDevice1` ou `MyClientDevice2`.

```
thingName: MyClientDevice1 OR thingName: MyClientDevice2
```

Exemple Exemple de règle de sélection (utiliser des caractères génériques)

La règle de sélection suivante correspond aux appareils clients dont le nom commence par `MyClientDevice`.

```
thingName: MyClientDevice*
```

Exemple Exemple de règle de sélection (utiliser des caractères génériques)

La règle de sélection suivante correspond aux appareils clients dont le nom se termine par `MyClientDevice`.


```
thingName: *MyClientDevice
```

Exemple Exemple de règle de sélection (correspond à tous les appareils)

La règle de sélection suivante correspond à tous les appareils clients.

```
thingName: *
```

policyName

Politique d'autorisation qui s'applique aux appareils clients de ce groupe d'appareils. Spécifiez le nom d'une politique que vous définissez dans l'objet policies.

policies

Les politiques d'autorisation des appareils clients pour les appareils clients qui se connectent au périphérique principal. Chaque politique d'autorisation spécifie un ensemble d'actions et les ressources sur lesquelles un appareil client peut effectuer ces actions.

Cet objet contient les informations suivantes :

policyNameKey

Nom de cette politique d'autorisation. *policyNameKey* Remplacez-le par un nom qui vous aide à identifier cette politique d'autorisation. Vous utilisez ce nom de stratégie pour définir la stratégie qui s'applique à un groupe d'appareils.

Cet objet contient les informations suivantes :

statementNameKey

Le nom de cette déclaration de politique. *statementNameKey* Remplacez-le par un nom qui vous aide à identifier cette déclaration de politique.

Cet objet contient les informations suivantes :

operations

Liste des opérations permettant d'utiliser les ressources de cette politique.

Vous pouvez inclure l'une des opérations suivantes :

- `mqtt:connect`— Accorde l'autorisation de se connecter à l'appareil principal. Les appareils clients doivent disposer de cette autorisation pour se connecter à un périphérique principal.

Cette opération prend en charge les ressources suivantes :

- `mqtt:clientId:`*deviceClientId*— Limitez l'accès en fonction de l'ID client utilisé par un appareil client pour se connecter au broker MQTT du périphérique principal. Remplacez *deviceClientId* par l'ID client à utiliser.
- `mqtt:publish`— Accorde l'autorisation de publier des messages MQTT sur des sujets.

Cette opération prend en charge les ressources suivantes :

- `mqtt:topic:`*mqttTopic*— Limitez l'accès en fonction du thème MQTT dans lequel un appareil client publie un message. Remplacez *mqttTopic* par le sujet à utiliser.

Cette ressource ne prend pas en charge les caractères génériques des rubriques MQTT.

- `mqtt:subscribe`— Accorde l'autorisation de s'abonner aux filtres de sujets MQTT pour recevoir des messages.

Cette opération prend en charge les ressources suivantes :

- `mqtt:topicfilter:`*mqttTopicFilter*— Limitez l'accès en fonction des sujets MQTT sur lesquels un appareil client peut s'abonner à des messages. Remplacez *mqttTopicFilter* par le filtre de rubrique à utiliser.

Cette ressource ne prend pas en charge les caractères génériques des rubriques MQTT.

resources

La liste des ressources permettant d'effectuer les opérations prévues dans cette politique. Spécifiez les ressources qui correspondent aux opérations de cette politique. Par exemple, vous pouvez spécifier une liste de ressources thématiques MQTT (`mqtt:topic:`*mqttTopic*) dans une politique qui spécifie l'`mqtt:publish` opération.

Vous pouvez spécifier le `*` caractère générique n'importe où dans la variable de ressource pour autoriser l'accès à toutes les ressources. Par exemple, vous pouvez spécifier `mqtt:topic:my*` d'autoriser l'accès aux ressources correspondant à cette entrée.

La variable de ressource suivante est prise en charge :

- `mqtt:topic:${iot:Connection.Thing.ThingName}`

Cela correspond au nom de l'objet dans le AWS IoT Core registre pour lequel la politique est évaluée. AWS IoT Core utilise le certificat présenté par l'appareil lorsqu'il s'authentifie pour déterminer quel élément utiliser pour vérifier la connexion. Cette variable de politique n'est disponible que lorsqu'un appareil se connecte via MQTT ou MQTT via le WebSocket protocole.

`statementDescription`

(Facultatif) Description de cette déclaration de politique.

`certificates`

(Facultatif) Les options de configuration des certificats pour ce périphérique principal. Cet objet contient les informations suivantes :

`serverCertificateValiditySeconds`

(Facultatif) Durée (en secondes) après laquelle le certificat du serveur MQTT local expire. Vous pouvez configurer cette option pour personnaliser la fréquence à laquelle les appareils clients se déconnectent et se reconnectent au périphérique principal.

Ce composant fait pivoter le certificat du serveur MQTT local 24 heures avant son expiration. Le courtier MQTT, tel que le [composant du courtier Moquette MQTT](#), génère un nouveau certificat et redémarre. Dans ce cas, tous les appareils clients connectés à ce périphérique principal sont déconnectés. Les appareils clients peuvent se reconnecter au périphérique principal après un court laps de temps.

Par défaut : 604800 (7 jours)

Valeur minimale : 172800 (2 jours)

Valeur maximale : 864000 (10 jours)

`performance`

(Facultatif) Les options de configuration des performances pour ce périphérique principal. Cet objet contient les informations suivantes :

`maxActiveAuthTokens`

(Facultatif) Le nombre maximum de jetons d'autorisation de l'appareil client actifs. Vous pouvez augmenter ce nombre pour permettre à un plus grand nombre d'appareils clients de se connecter à un appareil central unique, sans les réauthentifier.

Par défaut : 2500

`cloudRequestQueueSize`

(Facultatif) Nombre maximal de AWS Cloud demandes à mettre en file d'attente avant que ce composant ne rejette les demandes.

Par défaut : 100

`maxConcurrentCloudRequests`

(Facultatif) Le nombre maximum de demandes simultanées à envoyer au AWS Cloud. Vous pouvez augmenter ce nombre pour améliorer les performances d'authentification sur les appareils principaux auxquels vous connectez un grand nombre d'appareils clients.

Par défaut : 1

`certificateAuthority`

(Facultatif) Options de configuration de l'autorité de certification pour remplacer l'autorité intermédiaire du périphérique principal par votre propre autorité de certification intermédiaire.

Note

Si vous configurez votre appareil principal Greengrass avec une autorité de certification (CA) personnalisée et que vous utilisez la même autorité de certification pour délivrer les certificats des appareils clients, Greengrass contourne les contrôles de politique d'autorisation pour les opérations MQTT du périphérique client. Le composant d'authentification du périphérique client fait entièrement confiance aux clients à l'aide de certificats signés par l'autorité de certification pour laquelle il est configuré.

Pour limiter ce comportement lors de l'utilisation d'une autorité de certification personnalisée, créez et signez les appareils clients à l'aide d'une autorité de certification différente ou d'une autorité de certification intermédiaire, puis ajustez les `certificateChainUri` champs `certificateUri` et pour qu'ils pointent vers l'autorité de certification intermédiaire appropriée.

Cet objet contient les informations suivantes.

URI du certificat

Emplacement du certificat. Il peut s'agir d'un URI de système de fichiers ou d'un URI pointant vers un certificat stocké dans un module de sécurité matériel.

`certificateChainUri`

Emplacement de la chaîne de certificats pour l'autorité de certification du périphérique principal. Il doit s'agir de la chaîne de certificats complète remontant à votre autorité de certification racine. Il peut s'agir d'un URI de système de fichiers ou d'un URI pointant vers une chaîne de certificats stockée dans un module de sécurité matériel.

`privateKeyUri`

Emplacement de la clé privée de l'appareil principal. Il peut s'agir d'un URI de système de fichiers ou d'un URI pointant vers une clé privée de certificat stockée dans un module de sécurité matériel.

`security`

(Facultatif) Options de configuration de sécurité pour ce périphérique principal. Cet objet contient les informations suivantes.

`clientDeviceTrustDurationMinutes`

Durée en minutes pendant laquelle les informations d'authentification d'un appareil client peuvent être fiables avant qu'il ne soit nécessaire de s'authentifier à nouveau auprès du périphérique principal. La valeur par défaut est 1.

`metrics`

(Facultatif) Les options de métriques pour ce périphérique principal. Les mesures d'erreur ne s'afficheront qu'en cas d'erreur d'authentification du périphérique client. Cet objet contient les informations suivantes :

`disableMetrics`

Si le `disableMetrics` champ est défini sur `true`, l'authentification de l'appareil client ne collectera pas de métriques.

Par défaut : `false`

aggregatePeriodSeconds

Période d'agrégation en secondes qui détermine la fréquence à laquelle l'authentification du dispositif client agrège les métriques et les envoie à l'agent de télémétrie. Cela ne change pas la fréquence à laquelle les métriques sont publiées, car l'agent de télémétrie les publie toujours une fois par jour.

Par défaut : 3600

startupTimeoutSeconds

(Facultatif) Durée maximale en secondes pendant laquelle le composant démarre. L'état du composant passe à `ERROR` s'il dépasse ce délai d'attente.

Par défaut : 120

Exemple Exemple : mise à jour de la fusion de configurations (à l'aide d'une politique restrictive)

L'exemple de configuration suivant indique d'autoriser les appareils clients dont le nom commence par `MyClientDevice` à se connecter et ce, `publish/subscribe` sur tous les sujets.

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyDeviceGroup": {
        "selectionRule": "thingName: MyClientDevice*",
        "policyName": "MyRestrictivePolicy"
      }
    },
    "policies": {
      "MyRestrictivePolicy": {
        "AllowConnect": {
          "statementDescription": "Allow client devices to connect.",
          "operations": [
            "mqtt:connect"
          ],
          "resources": [
            "*"
          ]
        },
        "AllowPublish": {
          "statementDescription": "Allow client devices to publish on test/topic.",
```

```

        "operations": [
            "mqtt:publish"
        ],
        "resources": [
            "mqtt:topic:test/topic"
        ]
    },
    "AllowSubscribe": {
        "statementDescription": "Allow client devices to subscribe to test/topic/
response.",
        "operations": [
            "mqtt:subscribe"
        ],
        "resources": [
            "mqtt:topicfilter:test/topic/response"
        ]
    }
}
}
}
}
}

```

Exemple Exemple : mise à jour de la fusion de configurations (à l'aide d'une politique permissive)

L'exemple de configuration suivant indique d'autoriser tous les appareils clients à se connecter et ce, publish/subscribe sur tous les sujets.

```

{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyPermissiveDeviceGroup": {
        "selectionRule": "thingName: *",
        "policyName": "MyPermissivePolicy"
      }
    },
    "policies": {
      "MyPermissivePolicy": {
        "AllowAll": {
          "statementDescription": "Allow client devices to perform all actions.",
          "operations": [
            "*"
          ],

```

```

        "resources": [
            "*"
        ]
    }
}
}
}
}

```

Exemple Exemple : mise à jour de la fusion de configurations (à l'aide d'une politique de nom d'objet)

L'exemple de configuration suivant permet aux appareils clients de publier sur des sujets commençant par le nom de l'objet du périphérique client et se terminant par la chaîne `topic`.

```

{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "myThing": {
        "selectionRule": "thingName: *",
        "policyName": "MyThingNamePolicy"
      }
    },
    "policies": {
      "MyThingNamePolicy": {
        "policyStatement": {
          "statementDescription": "mqtt publish",
          "operations": [
            "mqtt:publish"
          ],
          "resources": [
            "mqtt:topic:${iot:Connection.Thing.ThingName}/*/topic"
          ]
        }
      }
    }
  }
}

```


v2.4.5

deviceGroups

Les groupes d'appareils sont des groupes d'appareils clients autorisés à se connecter et à communiquer avec un appareil principal. Utilisez des règles de sélection pour identifier les groupes d'appareils clients et définissez des politiques d'autorisation des appareils clients qui spécifient les autorisations pour chaque groupe d'appareils.

Cet objet contient les informations suivantes :

formatVersion

Version du format pour cet objet de configuration.

Sélectionnez parmi les options suivantes :

- 2021-03-05

definitions

Les groupes de périphériques pour cet appareil principal. Chaque définition spécifie une règle de sélection pour évaluer si un appareil client est membre du groupe. Chaque définition spécifie également la politique d'autorisation à appliquer aux appareils clients qui répondent à la règle de sélection. Si un appareil client est membre de plusieurs groupes d'appareils, les autorisations de l'appareil sont comprises dans la politique d'autorisation de chaque groupe.

Cet objet contient les informations suivantes :

groupNameKey

Nom de ce groupe d'appareils. *groupNameKey* Remplacez-le par un nom qui vous aide à identifier ce groupe d'appareils.

Cet objet contient les informations suivantes :

selectionRule

Requête qui indique quels appareils clients sont membres de ce groupe d'appareils. Lorsqu'un dispositif client se connecte, le dispositif principal évalue cette règle de sélection pour déterminer si le dispositif client est membre de ce groupe de dispositifs. Si le dispositif client est membre, le dispositif principal utilise la politique de ce groupe d'appareils pour autoriser les actions du dispositif client.

Chaque règle de sélection comprend au moins une clause de règle de sélection, qui est une requête d'expression unique pouvant correspondre aux dispositifs clients. Les règles de sélection utilisent la même syntaxe de requête que l'indexation des AWS IoT flottes. Pour plus d'informations sur la syntaxe des règles de sélection, consultez la section [Syntaxe des requêtes d'indexation du AWS IoT parc](#) dans le Guide du AWS IoT Core développeur.

Utilisez le * caractère générique pour associer plusieurs appareils clients à une seule clause de règle de sélection. Vous pouvez utiliser ce caractère générique au début et à la fin du nom de l'objet pour faire correspondre les appareils clients dont le nom commence ou se termine par la chaîne que vous spécifiez. Vous pouvez également utiliser ce caractère générique pour correspondre à tous les appareils clients.

 Note

Pour sélectionner une valeur contenant deux points (:), éliminez les deux points par une barre oblique inverse (\/). Dans les formats tels que JSON, vous devez éviter les barres obliques inversées. Vous devez donc saisir deux barres obliques inversées avant le caractère deux-points. Par exemple, spécifiez `thingName: MyTeam\\:ClientDevice1` de sélectionner un objet dont le nom est `MyTeam:ClientDevice1`.

Vous pouvez spécifier le sélecteur suivant :

- `thingName`— Le nom de l' AWS IoT objet d'un appareil client.

Exemple Exemple de règle de sélection

La règle de sélection suivante correspond aux appareils clients dont les noms sont `MyClientDevice1` ou `MyClientDevice2`.

```
thingName: MyClientDevice1 OR thingName: MyClientDevice2
```

Exemple Exemple de règle de sélection (utiliser des caractères génériques)

La règle de sélection suivante correspond aux appareils clients dont le nom commence par `MyClientDevice`.

```
thingName: MyClientDevice*
```

Exemple Exemple de règle de sélection (utiliser des caractères génériques)

La règle de sélection suivante correspond aux appareils clients dont le nom se termine parMyClientDevice.

```
thingName: *MyClientDevice
```

Exemple Exemple de règle de sélection (correspond à tous les appareils)

La règle de sélection suivante correspond à tous les appareils clients.

```
thingName: *
```

policyName

Politique d'autorisation qui s'applique aux appareils clients de ce groupe d'appareils. Spécifiez le nom d'une politique que vous définissez dans l'objectspolicies.

policies

Les politiques d'autorisation des appareils clients pour les appareils clients qui se connectent au périphérique principal. Chaque politique d'autorisation spécifie un ensemble d'actions et les ressources sur lesquelles un appareil client peut effectuer ces actions.

Cet objet contient les informations suivantes :

policyNameKey

Nom de cette politique d'autorisation. *policyNameKey*Remplacez-le par un nom qui vous aide à identifier cette politique d'autorisation. Vous utilisez ce nom de stratégie pour définir la stratégie qui s'applique à un groupe d'appareils.

Cet objet contient les informations suivantes :

statementNameKey

Le nom de cette déclaration de politique. *statementNameKey*Remplacez-le par un nom qui vous aide à identifier cette déclaration de politique.

Cet objet contient les informations suivantes :

operations

Liste des opérations permettant d'utiliser les ressources de cette politique.

Vous pouvez inclure l'une des opérations suivantes :

- `mqtt:connect`— Accorde l'autorisation de se connecter à l'appareil principal. Les appareils clients doivent disposer de cette autorisation pour se connecter à un périphérique principal.

Cette opération prend en charge les ressources suivantes :

- `mqtt:clientId:`*deviceClientId*— Limitez l'accès en fonction de l'ID client utilisé par un appareil client pour se connecter au broker MQTT du périphérique principal. Remplacez *deviceClientId* par l'ID client à utiliser.
- `mqtt:publish`— Accorde l'autorisation de publier des messages MQTT sur des sujets.

Cette opération prend en charge les ressources suivantes :

- `mqtt:topic:`*mqttTopic*— Limitez l'accès en fonction du thème MQTT dans lequel un appareil client publie un message. Remplacez *mqttTopic* par le sujet à utiliser.

Cette ressource ne prend pas en charge les caractères génériques des rubriques MQTT.

- `mqtt:subscribe`— Accorde l'autorisation de s'abonner aux filtres de sujets MQTT pour recevoir des messages.

Cette opération prend en charge les ressources suivantes :

- `mqtt:topicfilter:`*mqttTopicFilter*— Limitez l'accès en fonction des sujets MQTT sur lesquels un appareil client peut s'abonner à des messages. Remplacez *mqttTopicFilter* par le filtre de rubrique à utiliser.

Cette ressource prend en charge les caractères génériques + des rubriques # MQTT et MQTT. Pour plus d'informations, consultez les [rubriques relatives au MQTT](#) dans le Guide du AWS IoT Core développeur.

L'appareil client peut s'abonner aux filtres de rubrique exacts que vous autorisez. Par exemple, si vous autorisez l'appareil client à s'abonner à la

`mqtt:topicfilter:client/+ /status` ressource, il peut s'y abonner, `client/+ /status` mais pas `client/client1/status`.

Vous pouvez spécifier le * caractère générique pour autoriser l'accès à toutes les actions.

`resources`

La liste des ressources permettant d'effectuer les opérations prévues dans cette politique. Spécifiez les ressources qui correspondent aux opérations de cette politique. Par exemple, vous pouvez spécifier une liste de ressources thématiques MQTT (`mqtt:topic:mqttTopic`) dans une politique qui spécifie `mqtt:publish` opération.

Vous pouvez spécifier le * caractère générique pour autoriser l'accès à toutes les ressources. Vous ne pouvez pas utiliser le * caractère générique pour faire correspondre des identificateurs de ressources partiels. Par exemple, vous pouvez spécifier `"resources": "*" ,` mais vous ne pouvez pas le faire `"resources": "mqtt:clientId:*" .`

`statementDescription`

(Facultatif) Description de cette déclaration de politique.

`certificates`

(Facultatif) Les options de configuration des certificats pour ce périphérique principal. Cet objet contient les informations suivantes :

`serverCertificateValiditySeconds`

(Facultatif) Durée (en secondes) après laquelle le certificat du serveur MQTT local expire. Vous pouvez configurer cette option pour personnaliser la fréquence à laquelle les appareils clients se déconnectent et se reconnectent au périphérique principal.

Ce composant fait pivoter le certificat du serveur MQTT local 24 heures avant son expiration. Le courtier MQTT, tel que le [composant du courtier Moquette MQTT](#), génère un nouveau certificat et redémarre. Dans ce cas, tous les appareils clients connectés à ce périphérique principal sont déconnectés. Les appareils clients peuvent se reconnecter au périphérique principal après un court laps de temps.

Par défaut : 604800 (7 jours)

Valeur minimale : 172800 (2 jours)

Valeur maximale : 864000 (10 jours)

performance

(Facultatif) Les options de configuration des performances pour ce périphérique principal. Cet objet contient les informations suivantes :

maxActiveAuthTokens

(Facultatif) Le nombre maximum de jetons d'autorisation de l'appareil client actifs. Vous pouvez augmenter ce nombre pour permettre à un plus grand nombre d'appareils clients de se connecter à un appareil central unique, sans les réauthentifier.

Par défaut : 2500

cloudRequestQueueSize

(Facultatif) Nombre maximal de AWS Cloud demandes à mettre en file d'attente avant que ce composant ne rejette les demandes.

Par défaut : 100

maxConcurrentCloudRequests

(Facultatif) Le nombre maximum de demandes simultanées à envoyer au AWS Cloud. Vous pouvez augmenter ce nombre pour améliorer les performances d'authentification sur les appareils principaux auxquels vous connectez un grand nombre d'appareils clients.

Par défaut : 1

certificateAuthority

(Facultatif) Options de configuration de l'autorité de certification pour remplacer l'autorité intermédiaire du périphérique principal par votre propre autorité de certification intermédiaire.

Note

Si vous configurez votre appareil principal Greengrass avec une autorité de certification (CA) personnalisée et que vous utilisez la même autorité de certification pour délivrer les certificats des appareils clients, Greengrass contourne les contrôles de politique d'autorisation pour les opérations MQTT du périphérique client. Le

composant d'authentification du périphérique client fait entièrement confiance aux clients à l'aide de certificats signés par l'autorité de certification pour laquelle il est configuré.

Pour limiter ce comportement lors de l'utilisation d'une autorité de certification personnalisée, créez et signez les appareils clients à l'aide d'une autorité de certification différente ou d'une autorité de certification intermédiaire, puis ajustez les `certificateChainUri` champs `certificateUri` et pour qu'ils pointent vers l'autorité de certification intermédiaire appropriée.

Cet objet contient les informations suivantes.

URI du certificat

Emplacement du certificat. Il peut s'agir d'un URI de système de fichiers ou d'un URI pointant vers un certificat stocké dans un module de sécurité matériel.

`certificateChainUri`

Emplacement de la chaîne de certificats pour l'autorité de certification du périphérique principal. Il doit s'agir de la chaîne de certificats complète remontant à votre autorité de certification racine. Il peut s'agir d'un URI de système de fichiers ou d'un URI pointant vers une chaîne de certificats stockée dans un module de sécurité matériel.

`privateKeyUri`

Emplacement de la clé privée de l'appareil principal. Il peut s'agir d'un URI de système de fichiers ou d'un URI pointant vers une clé privée de certificat stockée dans un module de sécurité matériel.

`security`

(Facultatif) Options de configuration de sécurité pour ce périphérique principal. Cet objet contient les informations suivantes.

`clientDeviceTrustDurationMinutes`

Durée en minutes pendant laquelle les informations d'authentification d'un appareil client peuvent être fiables avant qu'il ne soit nécessaire de s'authentifier à nouveau auprès du périphérique principal. La valeur par défaut est 1.

metrics

(Facultatif) Les options de métriques pour ce périphérique principal. Les mesures d'erreur ne s'afficheront qu'en cas d'erreur d'authentification du périphérique client. Cet objet contient les informations suivantes :

disableMetrics

Si le `disableMetrics` champ est défini sur `true`, l'authentification de l'appareil client ne collectera pas de métriques.

Par défaut : `false`

aggregatePeriodSeconds

Période d'agrégation en secondes qui détermine la fréquence à laquelle l'authentification du dispositif client agrège les métriques et les envoie à l'agent de télémétrie. Cela ne change pas la fréquence à laquelle les métriques sont publiées, car l'agent de télémétrie les publie toujours une fois par jour.

Par défaut : `3600`

startupTimeoutSeconds

(Facultatif) Durée maximale en secondes pendant laquelle le composant démarre. L'état du composant passe à `ERRORLED` s'il dépasse ce délai d'attente.

Par défaut : `120`

Exemple Exemple : mise à jour de la fusion de configurations (à l'aide d'une politique restrictive)

L'exemple de configuration suivant indique d'autoriser les appareils clients dont le nom commence par `MyClientDevice` à se connecter et ce, `publish/subscribe` sur tous les sujets.

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyDeviceGroup": {
        "selectionRule": "thingName: MyClientDevice*",
        "policyName": "MyRestrictivePolicy"
      }
    }
  },
}
```



```

    "policies": {
      "MyRestrictivePolicy": {
        "AllowConnect": {
          "statementDescription": "Allow client devices to connect.",
          "operations": [
            "mqtt:connect"
          ],
          "resources": [
            "*"
          ]
        },
        "AllowPublish": {
          "statementDescription": "Allow client devices to publish on test/topic.",
          "operations": [
            "mqtt:publish"
          ],
          "resources": [
            "mqtt:topic:test/topic"
          ]
        },
        "AllowSubscribe": {
          "statementDescription": "Allow client devices to subscribe to test/topic/
response.",
          "operations": [
            "mqtt:subscribe"
          ],
          "resources": [
            "mqtt:topicfilter:test/topic/response"
          ]
        }
      }
    }
  }
}

```

Exemple Exemple : mise à jour de la fusion de configurations (à l'aide d'une politique permissive)

L'exemple de configuration suivant indique d'autoriser tous les appareils clients à se connecter et ce, publish/subscribe sur tous les sujets.

```

{
  "deviceGroups": {
    "formatVersion": "2021-03-05",

```

```

"definitions": {
  "MyPermissiveDeviceGroup": {
    "selectionRule": "thingName: *",
    "policyName": "MyPermissivePolicy"
  }
},
"policies": {
  "MyPermissivePolicy": {
    "AllowAll": {
      "statementDescription": "Allow client devices to perform all actions.",
      "operations": [
        "*"
      ],
      "resources": [
        "*"
      ]
    }
  }
}
}
}
}

```

v2.4.2 - v2.4.4

deviceGroups

Les groupes d'appareils sont des groupes d'appareils clients autorisés à se connecter et à communiquer avec un appareil principal. Utilisez des règles de sélection pour identifier les groupes d'appareils clients et définissez des politiques d'autorisation des appareils clients qui spécifient les autorisations pour chaque groupe d'appareils.

Cet objet contient les informations suivantes :

formatVersion

Version du format pour cet objet de configuration.

Sélectionnez parmi les options suivantes :

- 2021-03-05

definitions

Les groupes de périphériques pour cet appareil principal. Chaque définition spécifie une règle de sélection pour évaluer si un appareil client est membre du groupe. Chaque

définition spécifique également la politique d'autorisation à appliquer aux appareils clients qui répondent à la règle de sélection. Si un appareil client est membre de plusieurs groupes d'appareils, les autorisations de l'appareil sont comprises dans la politique d'autorisation de chaque groupe.

Cet objet contient les informations suivantes :

groupNameKey

Nom de ce groupe d'appareils. *groupNameKey* Remplacez-le par un nom qui vous aide à identifier ce groupe d'appareils.

Cet objet contient les informations suivantes :

selectionRule

Requête qui indique quels appareils clients sont membres de ce groupe d'appareils. Lorsqu'un dispositif client se connecte, le dispositif principal évalue cette règle de sélection pour déterminer si le dispositif client est membre de ce groupe de dispositifs. Si le dispositif client est membre, le dispositif principal utilise la politique de ce groupe d'appareils pour autoriser les actions du dispositif client.

Chaque règle de sélection comprend au moins une clause de règle de sélection, qui est une requête d'expression unique pouvant correspondre aux dispositifs clients. Les règles de sélection utilisent la même syntaxe de requête que l'indexation des AWS IoT flottes. Pour plus d'informations sur la syntaxe des règles de sélection, consultez la section [Syntaxe des requêtes d'indexation du AWS IoT parc](#) dans le Guide du AWS IoT Core développeur.

Utilisez le * caractère générique pour associer plusieurs appareils clients à une seule clause de règle de sélection. Vous pouvez utiliser ce caractère générique à la fin du nom de l'objet pour faire correspondre les appareils clients dont le nom commence par une chaîne que vous spécifiez. Vous pouvez également utiliser ce caractère générique pour correspondre à tous les appareils clients.

Note

Pour sélectionner une valeur contenant deux points (:), éliminez les deux points par une barre oblique inverse (\/). Dans les formats tels que JSON, vous devez éviter les barres obliques inversées. Vous devez donc saisir

deux barres obliques inversées avant le caractère deux-points. Par exemple, spécifiez `thingName: MyTeam\\\\:ClientDevice1` de sélectionner un objet dont le nom est `MyTeam:ClientDevice1`.

Vous pouvez spécifier le sélecteur suivant :

- `thingName`— Le nom de l' AWS IoT objet d'un appareil client.

Exemple Exemple de règle de sélection

La règle de sélection suivante correspond aux appareils clients dont les noms sont `MyClientDevice1` ou `MyClientDevice2`.

```
thingName: MyClientDevice1 OR thingName: MyClientDevice2
```

Exemple Exemple de règle de sélection (utiliser des caractères génériques)

La règle de sélection suivante correspond aux appareils clients dont le nom commence par `MyClientDevice`.

```
thingName: MyClientDevice*
```

Exemple Exemple de règle de sélection (correspond à tous les appareils)

La règle de sélection suivante correspond à tous les appareils clients.

```
thingName: *
```

`policyName`

Politique d'autorisation qui s'applique aux appareils clients de ce groupe d'appareils. Spécifiez le nom d'une politique que vous définissez dans l'`policies` objet.

`policies`

Les politiques d'autorisation des appareils clients pour les appareils clients qui se connectent au périphérique principal. Chaque politique d'autorisation spécifie un ensemble d'actions et les ressources sur lesquelles un appareil client peut effectuer ces actions.

Cet objet contient les informations suivantes :

policyNameKey

Nom de cette politique d'autorisation. *policyNameKey* Remplacez-le par un nom qui vous aide à identifier cette politique d'autorisation. Vous utilisez ce nom de stratégie pour définir la stratégie qui s'applique à un groupe d'appareils.

Cet objet contient les informations suivantes :

statementNameKey

Le nom de cette déclaration de politique. *statementNameKey* Remplacez-le par un nom qui vous aide à identifier cette déclaration de politique.

Cet objet contient les informations suivantes :

operations

Liste des opérations permettant d'utiliser les ressources de cette politique.

Vous pouvez inclure l'une des opérations suivantes :

- `mqtt:connect`— Accorde l'autorisation de se connecter à l'appareil principal. Les appareils clients doivent disposer de cette autorisation pour se connecter à un périphérique principal.

Cette opération prend en charge les ressources suivantes :

- `mqtt:clientId:`*deviceClientId*— Limitez l'accès en fonction de l'ID client utilisé par un appareil client pour se connecter au broker MQTT du périphérique principal. Remplacez *deviceClientId* par l'ID client à utiliser.
- `mqtt:publish`— Accorde l'autorisation de publier des messages MQTT sur des sujets.

Cette opération prend en charge les ressources suivantes :

- `mqtt:topic:`*mqttTopic*— Limitez l'accès en fonction du thème MQTT dans lequel un appareil client publie un message. Remplacez *mqttTopic* par le sujet à utiliser.

Cette ressource ne prend pas en charge les caractères génériques des rubriques MQTT.

- `mqtt:subscribe`— Accorde l'autorisation de s'abonner aux filtres de sujets MQTT pour recevoir des messages.

Cette opération prend en charge les ressources suivantes :

- `mqtt:topicfilter:mqttTopicFilter`— Limitez l'accès en fonction des sujets MQTT sur lesquels un appareil client peut s'abonner à des messages. Remplacez *mqttTopicFilter* par le filtre de rubrique à utiliser.

Cette ressource prend en charge les caractères génériques + des rubriques # MQTT et MQTT. Pour plus d'informations, consultez les [rubriques relatives au MQTT](#) dans le Guide du AWS IoT Core développeur.

L'appareil client peut s'abonner aux filtres de rubrique exacts que vous autorisez. Par exemple, si vous autorisez l'appareil client à s'abonner à la `mqtt:topicfilter:client/+/status` ressource, il peut s'y abonner, `client/+/status` mais pas `client/client1/status`.

Vous pouvez spécifier le * caractère générique pour autoriser l'accès à toutes les actions.

resources

La liste des ressources permettant d'effectuer les opérations prévues dans cette politique. Spécifiez les ressources qui correspondent aux opérations de cette politique. Par exemple, vous pouvez spécifier une liste de ressources thématiques MQTT (`mqtt:topic:mqttTopic`) dans une politique qui spécifie l'`mqtt:publish` opération.

Vous pouvez spécifier le * caractère générique pour autoriser l'accès à toutes les ressources. Vous ne pouvez pas utiliser le * caractère générique pour faire correspondre des identificateurs de ressources partiels. Par exemple, vous pouvez spécifier `"resources": "*"`, mais vous ne pouvez pas le faire `"resources": "mqtt:clientId:*"`.

statementDescription

(Facultatif) Description de cette déclaration de politique.

certificates

(Facultatif) Les options de configuration des certificats pour ce périphérique principal. Cet objet contient les informations suivantes :

serverCertificateValiditySeconds

(Facultatif) Durée (en secondes) après laquelle le certificat du serveur MQTT local expire. Vous pouvez configurer cette option pour personnaliser la fréquence à laquelle les appareils clients se déconnectent et se reconnectent au périphérique principal.

Ce composant fait pivoter le certificat du serveur MQTT local 24 heures avant son expiration. Le courtier MQTT, tel que le [composant du courtier Moquette MQTT](#), génère un nouveau certificat et redémarre. Dans ce cas, tous les appareils clients connectés à ce périphérique principal sont déconnectés. Les appareils clients peuvent se reconnecter au périphérique principal après un court laps de temps.

Par défaut : 604800 (7 jours)

Valeur minimale : 172800 (2 jours)

Valeur maximale : 864000 (10 jours)

performance

(Facultatif) Les options de configuration des performances pour ce périphérique principal. Cet objet contient les informations suivantes :

maxActiveAuthTokens

(Facultatif) Le nombre maximum de jetons d'autorisation de l'appareil client actifs. Vous pouvez augmenter ce nombre pour permettre à un plus grand nombre d'appareils clients de se connecter à un appareil central unique, sans les réauthentifier.

Par défaut : 2500

cloudRequestQueueSize

(Facultatif) Nombre maximal de AWS Cloud demandes à mettre en file d'attente avant que ce composant ne rejette les demandes.

Par défaut : 100

maxConcurrentCloudRequests

(Facultatif) Le nombre maximum de demandes simultanées à envoyer au AWS Cloud. Vous pouvez augmenter ce nombre pour améliorer les performances d'authentification sur les appareils principaux auxquels vous connectez un grand nombre d'appareils clients.

Par défaut : 1

`certificateAuthority`

(Facultatif) Options de configuration de l'autorité de certification pour remplacer l'autorité intermédiaire du périphérique principal par votre propre autorité de certification intermédiaire.

 Note

Si vous configurez votre appareil principal Greengrass avec une autorité de certification (CA) personnalisée et que vous utilisez la même autorité de certification pour délivrer les certificats des appareils clients, Greengrass contourne les contrôles de politique d'autorisation pour les opérations MQTT du périphérique client. Le composant d'authentification du périphérique client fait entièrement confiance aux clients à l'aide de certificats signés par l'autorité de certification pour laquelle il est configuré.

Pour limiter ce comportement lors de l'utilisation d'une autorité de certification personnalisée, créez et signez les appareils clients à l'aide d'une autorité de certification différente ou d'une autorité de certification intermédiaire, puis ajustez les `certificateChainUri` champs `certificateUri` et pour qu'ils pointent vers l'autorité de certification intermédiaire appropriée.

Cet objet contient les informations suivantes.

URI du certificat

Emplacement du certificat. Il peut s'agir d'un URI de système de fichiers ou d'un URI pointant vers un certificat stocké dans un module de sécurité matériel.

`certificateChainUri`

Emplacement de la chaîne de certificats pour l'autorité de certification du périphérique principal. Il doit s'agir de la chaîne de certificats complète remontant à votre autorité de certification racine. Il peut s'agir d'un URI de système de fichiers ou d'un URI pointant vers une chaîne de certificats stockée dans un module de sécurité matériel.

`privateKeyUri`

Emplacement de la clé privée de l'appareil principal. Il peut s'agir d'un URI de système de fichiers ou d'un URI pointant vers une clé privée de certificat stockée dans un module de sécurité matériel.

security

(Facultatif) Options de configuration de sécurité pour ce périphérique principal. Cet objet contient les informations suivantes.

clientDeviceTrustDurationMinutes

Durée en minutes pendant laquelle les informations d'authentification d'un appareil client peuvent être fiables avant qu'il ne soit nécessaire de s'authentifier à nouveau auprès du périphérique principal. La valeur par défaut est 1.

metrics

(Facultatif) Les options de métriques pour ce périphérique principal. Les mesures d'erreur ne s'afficheront qu'en cas d'erreur d'authentification du périphérique client. Cet objet contient les informations suivantes :

disableMetrics

Si le `disableMetrics` champ est défini sur `true`, l'authentification de l'appareil client ne collectera pas de métriques.

Par défaut : `false`

aggregatePeriodSeconds

Période d'agrégation en secondes qui détermine la fréquence à laquelle l'authentification du dispositif client agrège les métriques et les envoie à l'agent de télémétrie. Cela ne change pas la fréquence à laquelle les métriques sont publiées, car l'agent de télémétrie les publie toujours une fois par jour.

Par défaut : `3600`

startupTimeoutSeconds

(Facultatif) Durée maximale en secondes pendant laquelle le composant démarre. L'état du composant passe à `ERROR` s'il dépasse ce délai d'attente.

Par défaut : `120`

Exemple Exemple : mise à jour de la fusion de configurations (à l'aide d'une politique restrictive)

L'exemple de configuration suivant indique d'autoriser les appareils clients dont le nom commence par `MyClientDevice` à se connecter et ce, `publish/subscribe` sur tous les sujets.

```

{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyDeviceGroup": {
        "selectionRule": "thingName: MyClientDevice*",
        "policyName": "MyRestrictivePolicy"
      }
    },
    "policies": {
      "MyRestrictivePolicy": {
        "AllowConnect": {
          "statementDescription": "Allow client devices to connect.",
          "operations": [
            "mqtt:connect"
          ],
          "resources": [
            "*"
          ]
        },
        "AllowPublish": {
          "statementDescription": "Allow client devices to publish on test/topic.",
          "operations": [
            "mqtt:publish"
          ],
          "resources": [
            "mqtt:topic:test/topic"
          ]
        },
        "AllowSubscribe": {
          "statementDescription": "Allow client devices to subscribe to test/topic/
response.",
          "operations": [
            "mqtt:subscribe"
          ],
          "resources": [
            "mqtt:topicfilter:test/topic/response"
          ]
        }
      }
    }
  }
}

```

Exemple Exemple : mise à jour de la fusion de configurations (à l'aide d'une politique permissive)

L'exemple de configuration suivant indique d'autoriser tous les appareils clients à se connecter et ce, publish/subscribe sur tous les sujets.

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyPermissiveDeviceGroup": {
        "selectionRule": "thingName: *",
        "policyName": "MyPermissivePolicy"
      }
    },
    "policies": {
      "MyPermissivePolicy": {
        "AllowAll": {
          "statementDescription": "Allow client devices to perform all actions.",
          "operations": [
            "*"
          ],
          "resources": [
            "*"
          ]
        }
      }
    }
  }
}
```

v2.4.0 - v2.4.1

deviceGroups

Les groupes d'appareils sont des groupes d'appareils clients autorisés à se connecter et à communiquer avec un appareil principal. Utilisez des règles de sélection pour identifier les groupes d'appareils clients et définissez des politiques d'autorisation des appareils clients qui spécifient les autorisations pour chaque groupe d'appareils.

Cet objet contient les informations suivantes :

formatVersion

Version du format pour cet objet de configuration.

Sélectionnez parmi les options suivantes :

- 2021-03-05

definitions

Les groupes de périphériques pour cet appareil principal. Chaque définition spécifie une règle de sélection pour évaluer si un appareil client est membre du groupe. Chaque définition spécifie également la politique d'autorisation à appliquer aux appareils clients qui répondent à la règle de sélection. Si un appareil client est membre de plusieurs groupes d'appareils, les autorisations de l'appareil sont comprises dans la politique d'autorisation de chaque groupe.

Cet objet contient les informations suivantes :

groupNameKey

Nom de ce groupe d'appareils. *groupNameKey* Remplacez-le par un nom qui vous aide à identifier ce groupe d'appareils.

Cet objet contient les informations suivantes :

selectionRule

Requête qui indique quels appareils clients sont membres de ce groupe d'appareils. Lorsqu'un dispositif client se connecte, le dispositif principal évalue cette règle de sélection pour déterminer si le dispositif client est membre de ce groupe de dispositifs. Si le dispositif client est membre, le dispositif principal utilise la politique de ce groupe d'appareils pour autoriser les actions du dispositif client.

Chaque règle de sélection comprend au moins une clause de règle de sélection, qui est une requête d'expression unique pouvant correspondre aux dispositifs clients. Les règles de sélection utilisent la même syntaxe de requête que l'indexation des AWS IoT flottes. Pour plus d'informations sur la syntaxe des règles de sélection, consultez la section [Syntaxe des requêtes d'indexation du AWS IoT parc](#) dans le Guide du AWS IoT Core développeur.

Utilisez le * caractère générique pour associer plusieurs appareils clients à une seule clause de règle de sélection. Vous pouvez utiliser ce caractère générique à la fin du nom de l'objet pour faire correspondre les appareils clients dont le nom commence par une chaîne que vous spécifiez. Vous pouvez également utiliser ce caractère générique pour correspondre à tous les appareils clients.

 Note

Pour sélectionner une valeur contenant deux points (:), éliminez les deux points par une barre oblique inverse (\\). Dans les formats tels que JSON, vous devez éviter les barres obliques inversées. Vous devez donc saisir deux barres obliques inversées avant le caractère deux-points. Par exemple, spécifiez `thingName: MyTeam\\\\:ClientDevice1` de sélectionner un objet dont le nom est `MyTeam:ClientDevice1`.

Vous pouvez spécifier le sélecteur suivant :

- `thingName`— Le nom de l' AWS IoT objet d'un appareil client.

Exemple Exemple de règle de sélection

La règle de sélection suivante correspond aux appareils clients dont les noms sont `MyClientDevice1` ou `MyClientDevice2`.

```
thingName: MyClientDevice1 OR thingName: MyClientDevice2
```

Exemple Exemple de règle de sélection (utiliser des caractères génériques)

La règle de sélection suivante correspond aux appareils clients dont le nom commence par `MyClientDevice`.

```
thingName: MyClientDevice*
```

Exemple Exemple de règle de sélection (correspond à tous les appareils)

La règle de sélection suivante correspond à tous les appareils clients.

```
thingName: *
```

`policyName`

Politique d'autorisation qui s'applique aux appareils clients de ce groupe d'appareils. Spécifiez le nom d'une politique que vous définissez dans l'`polices` objet.

policies

Les politiques d'autorisation des appareils clients pour les appareils clients qui se connectent au périphérique principal. Chaque politique d'autorisation spécifie un ensemble d'actions et les ressources sur lesquelles un appareil client peut effectuer ces actions.

Cet objet contient les informations suivantes :

policyNameKey

Nom de cette politique d'autorisation. *policyNameKey* Remplacez-le par un nom qui vous aide à identifier cette politique d'autorisation. Vous utilisez ce nom de stratégie pour définir la stratégie qui s'applique à un groupe d'appareils.

Cet objet contient les informations suivantes :

statementNameKey

Le nom de cette déclaration de politique. *statementNameKey* Remplacez-le par un nom qui vous aide à identifier cette déclaration de politique.

Cet objet contient les informations suivantes :

operations

Liste des opérations permettant d'utiliser les ressources de cette politique.

Vous pouvez inclure l'une des opérations suivantes :

- `mqtt:connect`— Accorde l'autorisation de se connecter à l'appareil principal. Les appareils clients doivent disposer de cette autorisation pour se connecter à un périphérique principal.

Cette opération prend en charge les ressources suivantes :

- `mqtt:clientId:`*deviceClientId*— Limitez l'accès en fonction de l'ID client utilisé par un appareil client pour se connecter au broker MQTT du périphérique principal. Remplacez *deviceClientId* par l'ID client à utiliser.
- `mqtt:publish`— Accorde l'autorisation de publier des messages MQTT sur des sujets.

Cette opération prend en charge les ressources suivantes :

- `mqtt:topic:`*mqttTopic*— Limitez l'accès en fonction du thème MQTT dans lequel un appareil client publie un message. Remplacez *mqttTopic* par le sujet à utiliser.

Cette ressource ne prend pas en charge les caractères génériques des rubriques MQTT.

- `mqtt:subscribe`— Accorde l'autorisation de s'abonner aux filtres de sujets MQTT pour recevoir des messages.

Cette opération prend en charge les ressources suivantes :

- `mqtt:topicfilter:`*mqttTopicFilter*— Limitez l'accès en fonction des sujets MQTT sur lesquels un appareil client peut s'abonner à des messages. Remplacez *mqttTopicFilter* par le filtre de rubrique à utiliser.

Cette ressource prend en charge les caractères génériques + des rubriques # MQTT et MQTT. Pour plus d'informations, consultez les [rubriques relatives au MQTT](#) dans le Guide du AWS IoT Core développeur.

L'appareil client peut s'abonner aux filtres de rubrique exacts que vous autorisez. Par exemple, si vous autorisez l'appareil client à s'abonner à la `mqtt:topicfilter:client/+/status` ressource, il peut s'y abonner, `client/+/status` mais pas `client/client1/status`.

Vous pouvez spécifier le * caractère générique pour autoriser l'accès à toutes les actions.

resources

La liste des ressources permettant d'effectuer les opérations prévues dans cette politique. Spécifiez les ressources qui correspondent aux opérations de cette politique. Par exemple, vous pouvez spécifier une liste de ressources thématiques MQTT (`mqtt:topic:`*mqttTopic*) dans une politique qui spécifie l'`mqtt:publish` opération.

Vous pouvez spécifier le * caractère générique pour autoriser l'accès à toutes les ressources. Vous ne pouvez pas utiliser le * caractère générique pour faire correspondre des identificateurs de ressources partiels. Par exemple, vous pouvez spécifier `"resources": "*"` , mais vous ne pouvez pas le faire `"resources": "mqtt:clientId:*"` .

statementDescription

(Facultatif) Description de cette déclaration de politique.

certificates

(Facultatif) Les options de configuration des certificats pour ce périphérique principal. Cet objet contient les informations suivantes :

serverCertificateValiditySeconds

(Facultatif) Durée (en secondes) après laquelle le certificat du serveur MQTT local expire. Vous pouvez configurer cette option pour personnaliser la fréquence à laquelle les appareils clients se déconnectent et se reconnectent au périphérique principal.

Ce composant fait pivoter le certificat du serveur MQTT local 24 heures avant son expiration. Le courtier MQTT, tel que le [composant du courtier Moquette MQTT](#), génère un nouveau certificat et redémarre. Dans ce cas, tous les appareils clients connectés à ce périphérique principal sont déconnectés. Les appareils clients peuvent se reconnecter au périphérique principal après un court laps de temps.

Par défaut : 604800 (7 jours)

Valeur minimale : 172800 (2 jours)

Valeur maximale : 864000 (10 jours)

performance

(Facultatif) Les options de configuration des performances pour ce périphérique principal. Cet objet contient les informations suivantes :

maxActiveAuthTokens

(Facultatif) Le nombre maximum de jetons d'autorisation de l'appareil client actifs. Vous pouvez augmenter ce nombre pour permettre à un plus grand nombre d'appareils clients de se connecter à un appareil central unique, sans les réauthentifier.

Par défaut : 2500

cloudRequestQueueSize

(Facultatif) Nombre maximal de AWS Cloud demandes à mettre en file d'attente avant que ce composant ne rejette les demandes.

Par défaut : 100

`maxConcurrentCloudRequests`

(Facultatif) Le nombre maximum de demandes simultanées à envoyer au AWS Cloud. Vous pouvez augmenter ce nombre pour améliorer les performances d'authentification sur les appareils principaux auxquels vous connectez un grand nombre d'appareils clients.

Par défaut : 1

`certificateAuthority`

(Facultatif) Options de configuration de l'autorité de certification pour remplacer l'autorité intermédiaire du périphérique principal par votre propre autorité de certification intermédiaire. Cet objet contient les informations suivantes.

Cet objet contient les informations suivantes :

URI du certificat

Emplacement du certificat. Il peut s'agir d'un URI de système de fichiers ou d'un URI pointant vers un certificat stocké dans un module de sécurité matériel.

`certificateChainUri`

Emplacement de la chaîne de certificats pour l'autorité de certification du périphérique principal. Il doit s'agir de la chaîne de certificats complète remontant à votre autorité de certification racine. Il peut s'agir d'un URI de système de fichiers ou d'un URI pointant vers une chaîne de certificats stockée dans un module de sécurité matériel.

`privateKeyUri`

Emplacement de la clé privée de l'appareil principal. Il peut s'agir d'un URI de système de fichiers ou d'un URI pointant vers une clé privée de certificat stockée dans un module de sécurité matériel.

`security`

(Facultatif) Options de configuration de sécurité pour ce périphérique principal. Cet objet contient les informations suivantes.

`clientDeviceTrustDurationMinutes`

Durée en minutes pendant laquelle les informations d'authentification d'un appareil client peuvent être fiables avant qu'il ne soit nécessaire de s'authentifier à nouveau auprès du périphérique principal. La valeur par défaut est 1.

metrics

(Facultatif) Les options de métriques pour ce périphérique principal. Les mesures d'erreur ne s'afficheront qu'en cas d'erreur d'authentification du périphérique client. Cet objet contient les informations suivantes :

disableMetrics

Si le `disableMetrics` champ est défini sur `true`, l'authentification de l'appareil client ne collectera pas de métriques.

Par défaut : `false`

aggregatePeriodSeconds

Période d'agrégation en secondes qui détermine la fréquence à laquelle l'authentification du dispositif client agrège les métriques et les envoie à l'agent de télémétrie. Cela ne change pas la fréquence à laquelle les métriques sont publiées, car l'agent de télémétrie les publie toujours une fois par jour.

Par défaut : `3600`

Exemple Exemple : mise à jour de la fusion de configurations (à l'aide d'une politique restrictive)

L'exemple de configuration suivant indique d'autoriser les appareils clients dont le nom commence par `MyClientDevice` à se connecter et ce, `publish/subscribe` sur tous les sujets.

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyDeviceGroup": {
        "selectionRule": "thingName: MyClientDevice*",
        "policyName": "MyRestrictivePolicy"
      }
    },
    "policies": {
      "MyRestrictivePolicy": {
        "AllowConnect": {
          "statementDescription": "Allow client devices to connect.",
          "operations": [
            "mqtt:connect"
          ],

```

```

        "resources": [
            "*"
        ]
    },
    "AllowPublish": {
        "statementDescription": "Allow client devices to publish on test/topic.",
        "operations": [
            "mqtt:publish"
        ],
        "resources": [
            "mqtt:topic:test/topic"
        ]
    },
    "AllowSubscribe": {
        "statementDescription": "Allow client devices to subscribe to test/topic/
response.",
        "operations": [
            "mqtt:subscribe"
        ],
        "resources": [
            "mqtt:topicfilter:test/topic/response"
        ]
    }
}
}
}
}
}

```

Exemple Exemple : mise à jour de la fusion de configurations (à l'aide d'une politique permissive)

L'exemple de configuration suivant indique d'autoriser tous les appareils clients à se connecter et ce, publish/subscribe sur tous les sujets.

```

{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyPermissiveDeviceGroup": {
        "selectionRule": "thingName: *",
        "policyName": "MyPermissivePolicy"
      }
    },
    "policies": {

```

```
"MyPermissivePolicy": {
  "AllowAll": {
    "statementDescription": "Allow client devices to perform all actions.",
    "operations": [
      "*"
    ],
    "resources": [
      "*"
    ]
  }
}
```

v2.3.x

deviceGroups

Les groupes d'appareils sont des groupes d'appareils clients autorisés à se connecter et à communiquer avec un appareil principal. Utilisez des règles de sélection pour identifier les groupes d'appareils clients et définissez des politiques d'autorisation des appareils clients qui spécifient les autorisations pour chaque groupe d'appareils.

Cet objet contient les informations suivantes :

formatVersion

Version du format pour cet objet de configuration.

Sélectionnez parmi les options suivantes :

- 2021-03-05

definitions

Les groupes de périphériques pour cet appareil principal. Chaque définition spécifie une règle de sélection pour évaluer si un appareil client est membre du groupe. Chaque définition spécifie également la politique d'autorisation à appliquer aux appareils clients qui répondent à la règle de sélection. Si un appareil client est membre de plusieurs groupes d'appareils, les autorisations de l'appareil sont comprises dans la politique d'autorisation de chaque groupe.

Cet objet contient les informations suivantes :

groupNameKey

Nom de ce groupe d'appareils. *groupNameKey* Remplacez-le par un nom qui vous aide à identifier ce groupe d'appareils.

Cet objet contient les informations suivantes :

`selectionRule`

Requête qui indique quels appareils clients sont membres de ce groupe d'appareils. Lorsqu'un dispositif client se connecte, le dispositif principal évalue cette règle de sélection pour déterminer si le dispositif client est membre de ce groupe de dispositifs. Si le dispositif client est membre, le dispositif principal utilise la politique de ce groupe d'appareils pour autoriser les actions du dispositif client.

Chaque règle de sélection comprend au moins une clause de règle de sélection, qui est une requête d'expression unique pouvant correspondre aux dispositifs clients. Les règles de sélection utilisent la même syntaxe de requête que l'indexation des AWS IoT flottes. Pour plus d'informations sur la syntaxe des règles de sélection, consultez la section [Syntaxe des requêtes d'indexation du AWS IoT parc](#) dans le Guide du AWS IoT Core développeur.

Utilisez le * caractère générique pour associer plusieurs appareils clients à une seule clause de règle de sélection. Vous pouvez utiliser ce caractère générique à la fin du nom de l'objet pour faire correspondre les appareils clients dont le nom commence par une chaîne que vous spécifiez. Vous pouvez également utiliser ce caractère générique pour correspondre à tous les appareils clients.

Note

Pour sélectionner une valeur contenant deux points (:), éliminez les deux points par une barre oblique inverse (\/). Dans les formats tels que JSON, vous devez éviter les barres obliques inversées. Vous devez donc saisir deux barres obliques inversées avant le caractère deux-points. Par exemple, spécifiez `thingName: MyTeam\\\/:ClientDevice1` de sélectionner un objet dont le nom est `MyTeam:ClientDevice1`.

Vous pouvez spécifier le sélecteur suivant :

- `thingName`— Le nom de l' AWS IoT objet d'un appareil client.

Exemple Exemple de règle de sélection

La règle de sélection suivante correspond aux appareils clients dont les noms sont MyClientDevice1 ou MyClientDevice2.

```
thingName: MyClientDevice1 OR thingName: MyClientDevice2
```

Exemple Exemple de règle de sélection (utiliser des caractères génériques)

La règle de sélection suivante correspond aux appareils clients dont le nom commence par MyClientDevice.

```
thingName: MyClientDevice*
```

Exemple Exemple de règle de sélection (correspond à tous les appareils)

La règle de sélection suivante correspond à tous les appareils clients.

```
thingName: *
```

policyName

Politique d'autorisation qui s'applique aux appareils clients de ce groupe d'appareils. Spécifiez le nom d'une politique que vous définissez dans l'objet policies.

policies

Les politiques d'autorisation des appareils clients pour les appareils clients qui se connectent au périphérique principal. Chaque politique d'autorisation spécifie un ensemble d'actions et les ressources sur lesquelles un appareil client peut effectuer ces actions.

Cet objet contient les informations suivantes :

policyNameKey

Nom de cette politique d'autorisation. *policyNameKey* Remplacez-le par un nom qui vous aide à identifier cette politique d'autorisation. Vous utilisez ce nom de stratégie pour définir la stratégie qui s'applique à un groupe d'appareils.

Cet objet contient les informations suivantes :

statementNameKey

Le nom de cette déclaration de politique. *statementNameKey* Remplacez-le par un nom qui vous aide à identifier cette déclaration de politique.

Cet objet contient les informations suivantes :

operations

Liste des opérations permettant d'utiliser les ressources de cette politique.

Vous pouvez inclure l'une des opérations suivantes :

- `mqtt:connect`— Accorde l'autorisation de se connecter à l'appareil principal. Les appareils clients doivent disposer de cette autorisation pour se connecter à un périphérique principal.

Cette opération prend en charge les ressources suivantes :

- `mqtt:clientId:`*deviceClientId*— Limitez l'accès en fonction de l'ID client utilisé par un appareil client pour se connecter au broker MQTT du périphérique principal. Remplacez *deviceClientId* par l'ID client à utiliser.
- `mqtt:publish`— Accorde l'autorisation de publier des messages MQTT sur des sujets.

Cette opération prend en charge les ressources suivantes :

- `mqtt:topic:`*mqttTopic*— Limitez l'accès en fonction du thème MQTT dans lequel un appareil client publie un message. Remplacez *mqttTopic* par le sujet à utiliser.

Cette ressource ne prend pas en charge les caractères génériques des rubriques MQTT.

- `mqtt:subscribe`— Accorde l'autorisation de s'abonner aux filtres de sujets MQTT pour recevoir des messages.

Cette opération prend en charge les ressources suivantes :

- `mqtt:topicfilter:`*mqttTopicFilter*— Limitez l'accès en fonction des sujets MQTT sur lesquels un appareil client peut s'abonner à des messages. Remplacez *mqttTopicFilter* par le filtre de rubrique à utiliser.

Cette ressource prend en charge les caractères génériques + des rubriques # MQTT et MQTT. Pour plus d'informations, consultez les [rubriques relatives au MQTT](#) dans le Guide du AWS IoT Core développeur.

L'appareil client peut s'abonner aux filtres de rubrique exacts que vous autorisez. Par exemple, si vous autorisez l'appareil client à s'abonner à la `mqtt:topicfilter:client/+ /status` ressource, il peut s'y abonner, `client/+ /status` mais pas `client/client1/status`.

Vous pouvez spécifier le * caractère générique pour autoriser l'accès à toutes les actions.

resources

La liste des ressources permettant d'effectuer les opérations prévues dans cette politique. Spécifiez les ressources qui correspondent aux opérations de cette politique. Par exemple, vous pouvez spécifier une liste de ressources thématiques MQTT (`mqtt:topic:mqttTopic`) dans une politique qui spécifie l'`mqtt:publish` opération.

Vous pouvez spécifier le * caractère générique pour autoriser l'accès à toutes les ressources. Vous ne pouvez pas utiliser le * caractère générique pour faire correspondre des identificateurs de ressources partiels. Par exemple, vous pouvez spécifier `"resources": "*" ,` mais vous ne pouvez pas le faire `"resources": "mqtt:clientId:*`.

statementDescription

(Facultatif) Description de cette déclaration de politique.

certificates

(Facultatif) Les options de configuration des certificats pour ce périphérique principal. Cet objet contient les informations suivantes :

serverCertificateValiditySeconds

(Facultatif) Durée (en secondes) après laquelle le certificat du serveur MQTT local expire. Vous pouvez configurer cette option pour personnaliser la fréquence à laquelle les appareils clients se déconnectent et se reconnectent au périphérique principal.

Ce composant fait pivoter le certificat du serveur MQTT local 24 heures avant son expiration. Le courtier MQTT, tel que le [composant du courtier Moquette MQTT](#), génère un nouveau certificat et redémarre. Dans ce cas, tous les appareils clients connectés à ce périphérique principal sont déconnectés. Les appareils clients peuvent se reconnecter au périphérique principal après un court laps de temps.

Par défaut : 604800 (7 jours)

Valeur minimale : 172800 (2 jours)

Valeur maximale : 864000 (10 jours)

performance

(Facultatif) Les options de configuration des performances pour ce périphérique principal. Cet objet contient les informations suivantes :

maxActiveAuthTokens

(Facultatif) Le nombre maximum de jetons d'autorisation de l'appareil client actifs. Vous pouvez augmenter ce nombre pour permettre à un plus grand nombre d'appareils clients de se connecter à un appareil central unique sans les réauthentifier.

Par défaut : 2500

cloudRequestQueueSize

(Facultatif) Nombre maximal de AWS Cloud demandes à mettre en file d'attente avant que ce composant ne rejette les demandes.

Par défaut : 100

maxConcurrentCloudRequests

(Facultatif) Le nombre maximum de demandes simultanées à envoyer au AWS Cloud. Vous pouvez augmenter ce nombre pour améliorer les performances d'authentification sur les appareils principaux auxquels vous connectez un grand nombre d'appareils clients.

Par défaut : 1

certificateAuthority

(Facultatif) Options de configuration de l'autorité de certification pour remplacer l'autorité intermédiaire du périphérique principal par votre propre autorité de certification intermédiaire. Cet objet contient les informations suivantes.

URI du certificat

Emplacement du certificat. Il peut s'agir d'un URI de système de fichiers ou d'un URI pointant vers un certificat stocké dans un module de sécurité matériel.

certificateChainUri

Emplacement de la chaîne de certificats pour l'autorité de certification du périphérique principal. Il doit s'agir de la chaîne de certificats complète remontant à votre autorité de certification racine. Il peut s'agir d'un URI de système de fichiers ou d'un URI pointant vers une chaîne de certificats stockée dans un module de sécurité matériel.

privateKeyUri

Emplacement de la clé privée de l'appareil principal. Il peut s'agir d'un URI de système de fichiers ou d'un URI pointant vers une clé privée de certificat stockée dans un module de sécurité matériel.

security

(Facultatif) Options de configuration de sécurité pour ce périphérique principal. Cet objet contient les informations suivantes.

clientDeviceTrustDurationMinutes

Durée en minutes pendant laquelle les informations d'authentification d'un appareil client peuvent être fiables avant qu'il ne soit nécessaire de s'authentifier à nouveau auprès du périphérique principal. La valeur par défaut est 1.

Exemple Exemple : mise à jour de la fusion de configurations (à l'aide d'une politique restrictive)

L'exemple de configuration suivant indique d'autoriser les appareils clients dont le nom commence par MyClientDevice à se connecter et ce, publish/subscribe sur tous les sujets.

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyDeviceGroup": {
        "selectionRule": "thingName: MyClientDevice*",
        "policyName": "MyRestrictivePolicy"
      }
    }
  },
}
```

```

    "policies": {
      "MyRestrictivePolicy": {
        "AllowConnect": {
          "statementDescription": "Allow client devices to connect.",
          "operations": [
            "mqtt:connect"
          ],
          "resources": [
            "*"
          ]
        },
        "AllowPublish": {
          "statementDescription": "Allow client devices to publish on test/topic.",
          "operations": [
            "mqtt:publish"
          ],
          "resources": [
            "mqtt:topic:test/topic"
          ]
        },
        "AllowSubscribe": {
          "statementDescription": "Allow client devices to subscribe to test/topic/
response.",
          "operations": [
            "mqtt:subscribe"
          ],
          "resources": [
            "mqtt:topicfilter:test/topic/response"
          ]
        }
      }
    }
  }
}

```

Exemple Exemple : mise à jour de la fusion de configurations (à l'aide d'une politique permissive)

L'exemple de configuration suivant indique d'autoriser tous les appareils clients à se connecter et ce, publish/subscribe sur tous les sujets.

```

{
  "deviceGroups": {
    "formatVersion": "2021-03-05",

```

```

"definitions": {
  "MyPermissiveDeviceGroup": {
    "selectionRule": "thingName: *",
    "policyName": "MyPermissivePolicy"
  }
},
"policies": {
  "MyPermissivePolicy": {
    "AllowAll": {
      "statementDescription": "Allow client devices to perform all actions.",
      "operations": [
        "*"
      ],
      "resources": [
        "*"
      ]
    }
  }
}
}
}
}

```

v2.2.x

deviceGroups

Les groupes d'appareils sont des groupes d'appareils clients autorisés à se connecter et à communiquer avec un appareil principal. Utilisez des règles de sélection pour identifier les groupes d'appareils clients et définissez des politiques d'autorisation des appareils clients qui spécifient les autorisations pour chaque groupe d'appareils.

Cet objet contient les informations suivantes :

formatVersion

Version du format pour cet objet de configuration.

Sélectionnez parmi les options suivantes :

- 2021-03-05

definitions

Les groupes de périphériques pour cet appareil principal. Chaque définition spécifie une règle de sélection pour évaluer si un appareil client est membre du groupe. Chaque

définition spécifique également la politique d'autorisation à appliquer aux appareils clients qui répondent à la règle de sélection. Si un appareil client est membre de plusieurs groupes d'appareils, les autorisations de l'appareil sont comprises dans la politique d'autorisation de chaque groupe.

Cet objet contient les informations suivantes :

groupNameKey

Nom de ce groupe d'appareils. *groupNameKey* Remplacez-le par un nom qui vous aide à identifier ce groupe d'appareils.

Cet objet contient les informations suivantes :

selectionRule

Requête qui indique quels appareils clients sont membres de ce groupe d'appareils. Lorsqu'un dispositif client se connecte, le dispositif principal évalue cette règle de sélection pour déterminer si le dispositif client est membre de ce groupe de dispositifs. Si le dispositif client est membre, le dispositif principal utilise la politique de ce groupe d'appareils pour autoriser les actions du dispositif client.

Chaque règle de sélection comprend au moins une clause de règle de sélection, qui est une requête d'expression unique pouvant correspondre aux dispositifs clients. Les règles de sélection utilisent la même syntaxe de requête que l'indexation des AWS IoT flottes. Pour plus d'informations sur la syntaxe des règles de sélection, consultez la section [Syntaxe des requêtes d'indexation du AWS IoT parc](#) dans le Guide du AWS IoT Core développeur.

Utilisez le * caractère générique pour associer plusieurs appareils clients à une seule clause de règle de sélection. Vous pouvez utiliser ce caractère générique à la fin du nom de l'objet pour faire correspondre les appareils clients dont le nom commence par une chaîne que vous spécifiez. Vous pouvez également utiliser ce caractère générique pour correspondre à tous les appareils clients.

Note

Pour sélectionner une valeur contenant deux points (:), éliminez les deux points par une barre oblique inverse (\/). Dans les formats tels que JSON, vous devez éviter les barres obliques inversées. Vous devez donc saisir

deux barres obliques inversées avant le caractère deux-points. Par exemple, spécifiez `thingName: MyTeam\\\\:ClientDevice1` de sélectionner un objet dont le nom est `MyTeam:ClientDevice1`.

Vous pouvez spécifier le sélecteur suivant :

- `thingName`— Le nom de l' AWS IoT objet d'un appareil client.

Exemple Exemple de règle de sélection

La règle de sélection suivante correspond aux appareils clients dont les noms sont `MyClientDevice1` ou `MyClientDevice2`.

```
thingName: MyClientDevice1 OR thingName: MyClientDevice2
```

Exemple Exemple de règle de sélection (utiliser des caractères génériques)

La règle de sélection suivante correspond aux appareils clients dont le nom commence par `MyClientDevice`.

```
thingName: MyClientDevice*
```

Exemple Exemple de règle de sélection (correspond à tous les appareils)

La règle de sélection suivante correspond à tous les appareils clients.

```
thingName: *
```

`policyName`

Politique d'autorisation qui s'applique aux appareils clients de ce groupe d'appareils. Spécifiez le nom d'une politique que vous définissez dans l'`policies` objet.

`policies`

Les politiques d'autorisation des appareils clients pour les appareils clients qui se connectent au périphérique principal. Chaque politique d'autorisation spécifie un ensemble d'actions et les ressources sur lesquelles un appareil client peut effectuer ces actions.

Cet objet contient les informations suivantes :

policyNameKey

Nom de cette politique d'autorisation. *policyNameKey* Remplacez-le par un nom qui vous aide à identifier cette politique d'autorisation. Vous utilisez ce nom de stratégie pour définir la stratégie qui s'applique à un groupe d'appareils.

Cet objet contient les informations suivantes :

statementNameKey

Le nom de cette déclaration de politique. *statementNameKey* Remplacez-le par un nom qui vous aide à identifier cette déclaration de politique.

Cet objet contient les informations suivantes :

operations

Liste des opérations permettant d'utiliser les ressources de cette politique.

Vous pouvez inclure l'une des opérations suivantes :

- `mqtt:connect`— Accorde l'autorisation de se connecter à l'appareil principal. Les appareils clients doivent disposer de cette autorisation pour se connecter à un périphérique principal.

Cette opération prend en charge les ressources suivantes :

- `mqtt:clientId:`*deviceClientId*— Limitez l'accès en fonction de l'ID client utilisé par un appareil client pour se connecter au broker MQTT du périphérique principal. Remplacez *deviceClientId* par l'ID client à utiliser.
- `mqtt:publish`— Accorde l'autorisation de publier des messages MQTT sur des sujets.

Cette opération prend en charge les ressources suivantes :

- `mqtt:topic:`*mqttTopic*— Limitez l'accès en fonction du thème MQTT dans lequel un appareil client publie un message. Remplacez *mqttTopic* par le sujet à utiliser.

Cette ressource ne prend pas en charge les caractères génériques des rubriques MQTT.

- `mqtt:subscribe`— Accorde l'autorisation de s'abonner aux filtres de sujets MQTT pour recevoir des messages.

Cette opération prend en charge les ressources suivantes :

- `mqtt:topicfilter:mqttTopicFilter`— Limitez l'accès en fonction des sujets MQTT sur lesquels un appareil client peut s'abonner à des messages. Remplacez *mqttTopicFilter* par le filtre de rubrique à utiliser.

Cette ressource prend en charge les caractères génériques + des rubriques # MQTT et MQTT. Pour plus d'informations, consultez les [rubriques relatives au MQTT](#) dans le Guide du AWS IoT Core développeur.

L'appareil client peut s'abonner aux filtres de rubrique exacts que vous autorisez. Par exemple, si vous autorisez l'appareil client à s'abonner à la `mqtt:topicfilter:client/+ /status` ressource, il peut s'y abonner, `client/+ /status` mais pas `client/client1/status`.

Vous pouvez spécifier le * caractère générique pour autoriser l'accès à toutes les actions.

resources

La liste des ressources permettant d'effectuer les opérations prévues dans cette politique. Spécifiez les ressources qui correspondent aux opérations de cette politique. Par exemple, vous pouvez spécifier une liste de ressources thématiques MQTT (`mqtt:topic:mqttTopic`) dans une politique qui spécifie l'`mqtt:publish` opération.

Vous pouvez spécifier le * caractère générique pour autoriser l'accès à toutes les ressources. Vous ne pouvez pas utiliser le * caractère générique pour faire correspondre des identificateurs de ressources partiels. Par exemple, vous pouvez spécifier `"resources": "*" ,` mais vous ne pouvez pas le faire `"resources": "mqtt:clientId:*`.

statementDescription

(Facultatif) Description de cette déclaration de politique.

certificates

(Facultatif) Les options de configuration des certificats pour ce périphérique principal. Cet objet contient les informations suivantes :

serverCertificateValiditySeconds

(Facultatif) Durée (en secondes) après laquelle le certificat du serveur MQTT local expire. Vous pouvez configurer cette option pour personnaliser la fréquence à laquelle les appareils clients se déconnectent et se reconnectent au périphérique principal.

Ce composant fait pivoter le certificat du serveur MQTT local 24 heures avant son expiration. Le courtier MQTT, tel que le [composant du courtier Moquette MQTT](#), génère un nouveau certificat et redémarre. Dans ce cas, tous les appareils clients connectés à ce périphérique principal sont déconnectés. Les appareils clients peuvent se reconnecter au périphérique principal après un court laps de temps.

Par défaut : 604800 (7 jours)

Valeur minimale : 172800 (2 jours)

Valeur maximale : 864000 (10 jours)

performance

(Facultatif) Les options de configuration des performances pour ce périphérique principal. Cet objet contient les informations suivantes :

maxActiveAuthTokens

(Facultatif) Le nombre maximum de jetons d'autorisation de l'appareil client actifs. Vous pouvez augmenter ce nombre pour permettre à un plus grand nombre d'appareils clients de se connecter à un appareil central unique sans les réauthentifier.

Par défaut : 2500

cloudRequestQueueSize

(Facultatif) Nombre maximal de AWS Cloud demandes à mettre en file d'attente avant que ce composant ne rejette les demandes.

Par défaut : 100

maxConcurrentCloudRequests

(Facultatif) Le nombre maximum de demandes simultanées à envoyer au AWS Cloud. Vous pouvez augmenter ce nombre pour améliorer les performances d'authentification sur les appareils principaux auxquels vous connectez un grand nombre d'appareils clients.

Par défaut : 1

Exemple Exemple : mise à jour de la fusion de configurations (à l'aide d'une politique restrictive)

L'exemple de configuration suivant indique d'autoriser les appareils clients dont le nom commence par MyClientDevice à se connecter et ce, publish/subscribe sur tous les sujets.

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyDeviceGroup": {
        "selectionRule": "thingName: MyClientDevice*",
        "policyName": "MyRestrictivePolicy"
      }
    },
    "policies": {
      "MyRestrictivePolicy": {
        "AllowConnect": {
          "statementDescription": "Allow client devices to connect.",
          "operations": [
            "mqtt:connect"
          ],
          "resources": [
            "*"
          ]
        },
        "AllowPublish": {
          "statementDescription": "Allow client devices to publish on test/topic.",
          "operations": [
            "mqtt:publish"
          ],
          "resources": [
            "mqtt:topic:test/topic"
          ]
        },
        "AllowSubscribe": {
          "statementDescription": "Allow client devices to subscribe to test/topic/
response.",
          "operations": [
            "mqtt:subscribe"
          ],
          "resources": [
            "mqtt:topicfilter:test/topic/response"
          ]
        }
      }
    }
  }
}
```

```

    }
  }
}

```

Exemple Exemple : mise à jour de la fusion de configurations (à l'aide d'une politique permissive)

L'exemple de configuration suivant indique d'autoriser tous les appareils clients à se connecter et ce, publish/subscribe sur tous les sujets.

```

{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyPermissiveDeviceGroup": {
        "selectionRule": "thingName: *",
        "policyName": "MyPermissivePolicy"
      }
    },
    "policies": {
      "MyPermissivePolicy": {
        "AllowAll": {
          "statementDescription": "Allow client devices to perform all actions.",
          "operations": [
            "*"
          ],
          "resources": [
            "*"
          ]
        }
      }
    }
  }
}

```

v2.1.x

deviceGroups

Les groupes d'appareils sont des groupes d'appareils clients autorisés à se connecter et à communiquer avec un appareil principal. Utilisez des règles de sélection pour identifier les groupes d'appareils clients et définissez des politiques d'autorisation des appareils clients qui spécifient les autorisations pour chaque groupe d'appareils.

Cet objet contient les informations suivantes :

`formatVersion`

Version du format pour cet objet de configuration.

Sélectionnez parmi les options suivantes :

- 2021-03-05

`definitions`

Les groupes de périphériques pour cet appareil principal. Chaque définition spécifie une règle de sélection pour évaluer si un appareil client est membre du groupe. Chaque définition spécifie également la politique d'autorisation à appliquer aux appareils clients qui répondent à la règle de sélection. Si un appareil client est membre de plusieurs groupes d'appareils, les autorisations de l'appareil sont comprises dans la politique d'autorisation de chaque groupe.

Cet objet contient les informations suivantes :

`groupNameKey`

Nom de ce groupe d'appareils. *`groupNameKey`* Remplacez-le par un nom qui vous aide à identifier ce groupe d'appareils.

Cet objet contient les informations suivantes :

`selectionRule`

Requête qui indique quels appareils clients sont membres de ce groupe d'appareils. Lorsqu'un dispositif client se connecte, le dispositif principal évalue cette règle de sélection pour déterminer si le dispositif client est membre de ce groupe de dispositifs. Si le dispositif client est membre, le dispositif principal utilise la politique de ce groupe d'appareils pour autoriser les actions du dispositif client.

Chaque règle de sélection comprend au moins une clause de règle de sélection, qui est une requête d'expression unique pouvant correspondre aux dispositifs clients. Les règles de sélection utilisent la même syntaxe de requête que l'indexation des AWS IoT flottes. Pour plus d'informations sur la syntaxe des règles de sélection, consultez la section [Syntaxe des requêtes d'indexation du AWS IoT parc](#) dans le Guide du AWS IoT Core développeur.

Utilisez le * caractère générique pour associer plusieurs appareils clients à une seule clause de règle de sélection. Vous pouvez utiliser ce caractère générique à la fin du nom de l'objet pour faire correspondre les appareils clients dont le nom commence par une chaîne que vous spécifiez. Vous pouvez également utiliser ce caractère générique pour correspondre à tous les appareils clients.

 Note

Pour sélectionner une valeur contenant deux points (:), éliminez les deux points par une barre oblique inverse (\\). Dans les formats tels que JSON, vous devez éviter les barres obliques inversées. Vous devez donc saisir deux barres obliques inversées avant le caractère deux-points. Par exemple, spécifiez `thingName: MyTeam\\\\:ClientDevice1` de sélectionner un objet dont le nom est `MyTeam:ClientDevice1`.

Vous pouvez spécifier le sélecteur suivant :

- `thingName`— Le nom de l' AWS IoT objet d'un appareil client.

Exemple Exemple de règle de sélection

La règle de sélection suivante correspond aux appareils clients dont les noms sont `MyClientDevice1` ou `MyClientDevice2`.

```
thingName: MyClientDevice1 OR thingName: MyClientDevice2
```

Exemple Exemple de règle de sélection (utiliser des caractères génériques)

La règle de sélection suivante correspond aux appareils clients dont le nom commence par `MyClientDevice`.

```
thingName: MyClientDevice*
```

Exemple Exemple de règle de sélection (correspond à tous les appareils)

La règle de sélection suivante correspond à tous les appareils clients.

```
thingName: *
```

policyName

Politique d'autorisation qui s'applique aux appareils clients de ce groupe d'appareils. Spécifiez le nom d'une politique que vous définissez dans l'objet policies.

policies

Les politiques d'autorisation des appareils clients pour les appareils clients qui se connectent au périphérique principal. Chaque politique d'autorisation spécifie un ensemble d'actions et les ressources sur lesquelles un appareil client peut effectuer ces actions.

Cet objet contient les informations suivantes :

policyNameKey

Nom de cette politique d'autorisation. *policyNameKey* Remplacez-le par un nom qui vous aide à identifier cette politique d'autorisation. Vous utilisez ce nom de stratégie pour définir la stratégie qui s'applique à un groupe d'appareils.

Cet objet contient les informations suivantes :

statementNameKey

Le nom de cette déclaration de politique. *statementNameKey* Remplacez-le par un nom qui vous aide à identifier cette déclaration de politique.

Cet objet contient les informations suivantes :

operations

Liste des opérations permettant d'utiliser les ressources de cette politique.

Vous pouvez inclure l'une des opérations suivantes :

- `mqtt:connect`— Accorde l'autorisation de se connecter à l'appareil principal. Les appareils clients doivent disposer de cette autorisation pour se connecter à un périphérique principal.

Cette opération prend en charge les ressources suivantes :

- `mqtt:clientId:`*deviceClientId*— Limitez l'accès en fonction de l'ID client utilisé par un appareil client pour se connecter au broker MQTT du périphérique principal. Remplacez *deviceClientId* par l'ID client à utiliser.

- `mqtt:publish`— Accorde l'autorisation de publier des messages MQTT sur des sujets.

Cette opération prend en charge les ressources suivantes :

- `mqtt:topic:mqttTopic`— Limitez l'accès en fonction du thème MQTT dans lequel un appareil client publie un message. Remplacez *mqttTopic* par le sujet à utiliser.

Cette ressource ne prend pas en charge les caractères génériques des rubriques MQTT.

- `mqtt:subscribe`— Accorde l'autorisation de s'abonner aux filtres de sujets MQTT pour recevoir des messages.

Cette opération prend en charge les ressources suivantes :

- `mqtt:topicfilter:mqttTopicFilter`— Limitez l'accès en fonction des sujets MQTT sur lesquels un appareil client peut s'abonner à des messages. Remplacez *mqttTopicFilter* par le filtre de rubrique à utiliser.

Cette ressource prend en charge les caractères génériques + des rubriques # MQTT et MQTT. Pour plus d'informations, consultez les [rubriques relatives au MQTT](#) dans le Guide du AWS IoT Core développeur.

L'appareil client peut s'abonner aux filtres de rubrique exacts que vous autorisez. Par exemple, si vous autorisez l'appareil client à s'abonner à la `mqtt:topicfilter:client/+/status` ressource, il peut s'y abonner, `client/+/status` mais pas `client/client1/status`.

Vous pouvez spécifier le * caractère générique pour autoriser l'accès à toutes les actions.

resources

La liste des ressources permettant d'effectuer les opérations prévues dans cette politique. Spécifiez les ressources qui correspondent aux opérations de cette politique. Par exemple, vous pouvez spécifier une liste de ressources thématiques MQTT (`mqtt:topic:mqttTopic`) dans une politique qui spécifie l'`mqtt:publish` opération.

Vous pouvez spécifier le * caractère générique pour autoriser l'accès à toutes les ressources. Vous ne pouvez pas utiliser le * caractère générique pour

faire correspondre des identificateurs de ressources partiels. Par exemple, vous pouvez spécifier **"resources": "*"** , mais vous ne pouvez pas le faire **"resources": "mqtt:clientId:*"** .

statementDescription

(Facultatif) Description de cette déclaration de politique.

certificates

(Facultatif) Les options de configuration des certificats pour ce périphérique principal. Cet objet contient les informations suivantes :

serverCertificateValiditySeconds

(Facultatif) Durée (en secondes) après laquelle le certificat du serveur MQTT local expire. Vous pouvez configurer cette option pour personnaliser la fréquence à laquelle les appareils clients se déconnectent et se reconnectent au périphérique principal.

Ce composant fait pivoter le certificat du serveur MQTT local 24 heures avant son expiration. Le courtier MQTT, tel que le [composant du courtier Moquette MQTT](#), génère un nouveau certificat et redémarre. Dans ce cas, tous les appareils clients connectés à ce périphérique principal sont déconnectés. Les appareils clients peuvent se reconnecter au périphérique principal après un court laps de temps.

Par défaut : 604800 (7 jours)

Valeur minimale : 172800 (2 jours)

Valeur maximale : 864000 (10 jours)

Exemple Exemple : mise à jour de la fusion de configurations (à l'aide d'une politique restrictive)

L'exemple de configuration suivant indique d'autoriser les appareils clients dont le nom commence par MyClientDevice à se connecter et ce, publish/subscribe sur tous les sujets.

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyDeviceGroup": {
        "selectionRule": "thingName: MyClientDevice*",
        "policyName": "MyRestrictivePolicy"
      }
    }
  }
}
```



```

    },
    "policies": {
      "MyRestrictivePolicy": {
        "AllowConnect": {
          "statementDescription": "Allow client devices to connect.",
          "operations": [
            "mqtt:connect"
          ],
          "resources": [
            "*"
          ]
        },
        "AllowPublish": {
          "statementDescription": "Allow client devices to publish on test/topic.",
          "operations": [
            "mqtt:publish"
          ],
          "resources": [
            "mqtt:topic:test/topic"
          ]
        },
        "AllowSubscribe": {
          "statementDescription": "Allow client devices to subscribe to test/topic/
response.",
          "operations": [
            "mqtt:subscribe"
          ],
          "resources": [
            "mqtt:topicfilter:test/topic/response"
          ]
        }
      }
    }
  }
}

```

Exemple Exemple : mise à jour de la fusion de configurations (à l'aide d'une politique permissive)

L'exemple de configuration suivant indique d'autoriser tous les appareils clients à se connecter et ce, publish/subscribe sur tous les sujets.

```

{
  "deviceGroups": {

```

```
"formatVersion": "2021-03-05",
"definitions": {
  "MyPermissiveDeviceGroup": {
    "selectionRule": "thingName: *",
    "policyName": "MyPermissivePolicy"
  }
},
"policies": {
  "MyPermissivePolicy": {
    "AllowAll": {
      "statementDescription": "Allow client devices to perform all actions.",
      "operations": [
        "*"
      ],
      "resources": [
        "*"
      ]
    }
  }
}
```

v2.0.x

deviceGroups

Les groupes d'appareils sont des groupes d'appareils clients autorisés à se connecter et à communiquer avec un appareil principal. Utilisez des règles de sélection pour identifier les groupes d'appareils clients et définissez des politiques d'autorisation des appareils clients qui spécifient les autorisations pour chaque groupe d'appareils.

Cet objet contient les informations suivantes :

formatVersion

Version du format pour cet objet de configuration.

Sélectionnez parmi les options suivantes :

- 2021-03-05

definitions

Les groupes de périphériques pour cet appareil principal. Chaque définition spécifie une règle de sélection pour évaluer si un appareil client est membre du groupe. Chaque définition spécifie également la politique d'autorisation à appliquer aux appareils clients qui répondent à la règle de sélection. Si un appareil client est membre de plusieurs groupes d'appareils, les autorisations de l'appareil sont comprises dans la politique d'autorisation de chaque groupe.

Cet objet contient les informations suivantes :

groupNameKey

Nom de ce groupe d'appareils. *groupNameKey* Remplacez-le par un nom qui vous aide à identifier ce groupe d'appareils.

Cet objet contient les informations suivantes :

selectionRule

Requête qui indique quels appareils clients sont membres de ce groupe d'appareils. Lorsqu'un dispositif client se connecte, le dispositif principal évalue cette règle de sélection pour déterminer si le dispositif client est membre de ce groupe de dispositifs. Si le dispositif client est membre, le dispositif principal utilise la politique de ce groupe d'appareils pour autoriser les actions du dispositif client.

Chaque règle de sélection comprend au moins une clause de règle de sélection, qui est une requête d'expression unique pouvant correspondre aux dispositifs clients. Les règles de sélection utilisent la même syntaxe de requête que l'indexation des AWS IoT flottes. Pour plus d'informations sur la syntaxe des règles de sélection, consultez la section [Syntaxe des requêtes d'indexation du AWS IoT parc](#) dans le Guide du AWS IoT Core développeur.

Utilisez le * caractère générique pour associer plusieurs appareils clients à une seule clause de règle de sélection. Vous pouvez utiliser ce caractère générique à la fin du nom de l'objet pour faire correspondre les appareils clients dont le nom commence par une chaîne que vous spécifiez. Vous pouvez également utiliser ce caractère générique pour correspondre à tous les appareils clients.

 Note

Pour sélectionner une valeur contenant deux points (:), éliminez les deux points par une barre oblique inverse (\\). Dans les formats tels que JSON, vous devez éviter les barres obliques inversées. Vous devez donc saisir deux barres obliques inversées avant le caractère deux-points. Par exemple, spécifiez `thingName: MyTeam\\\\:ClientDevice1` de sélectionner un objet dont le nom est `MyTeam:ClientDevice1`.

Vous pouvez spécifier le sélecteur suivant :

- `thingName`— Le nom de l' AWS IoT objet d'un appareil client.

Exemple Exemple de règle de sélection

La règle de sélection suivante correspond aux appareils clients dont les noms sont `MyClientDevice1` ou `MyClientDevice2`.

```
thingName: MyClientDevice1 OR thingName: MyClientDevice2
```

Exemple Exemple de règle de sélection (utiliser des caractères génériques)

La règle de sélection suivante correspond aux appareils clients dont le nom commence par `MyClientDevice`.

```
thingName: MyClientDevice*
```

Exemple Exemple de règle de sélection (correspond à tous les appareils)

La règle de sélection suivante correspond à tous les appareils clients.

```
thingName: *
```

`policyName`

Politique d'autorisation qui s'applique aux appareils clients de ce groupe d'appareils. Spécifiez le nom d'une politique que vous définissez dans l'`policies` objet.

policies

Les politiques d'autorisation des appareils clients pour les appareils clients qui se connectent au périphérique principal. Chaque politique d'autorisation spécifie un ensemble d'actions et les ressources sur lesquelles un appareil client peut effectuer ces actions.

Cet objet contient les informations suivantes :

policyNameKey

Nom de cette politique d'autorisation. *policyNameKey* Remplacez-le par un nom qui vous aide à identifier cette politique d'autorisation. Vous utilisez ce nom de stratégie pour définir la stratégie qui s'applique à un groupe d'appareils.

Cet objet contient les informations suivantes :

statementNameKey

Le nom de cette déclaration de politique. *statementNameKey* Remplacez-le par un nom qui vous aide à identifier cette déclaration de politique.

Cet objet contient les informations suivantes :

operations

Liste des opérations permettant d'utiliser les ressources de cette politique.

Vous pouvez inclure l'une des opérations suivantes :

- `mqtt:connect`— Accorde l'autorisation de se connecter à l'appareil principal. Les appareils clients doivent disposer de cette autorisation pour se connecter à un périphérique principal.

Cette opération prend en charge les ressources suivantes :

- `mqtt:clientId:`*deviceClientId*— Limitez l'accès en fonction de l'ID client utilisé par un appareil client pour se connecter au broker MQTT du périphérique principal. Remplacez *deviceClientId* par l'ID client à utiliser.
- `mqtt:publish`— Accorde l'autorisation de publier des messages MQTT sur des sujets.

Cette opération prend en charge les ressources suivantes :

- `mqtt:topic:mqttTopic`— Limitez l'accès en fonction du thème MQTT dans lequel un appareil client publie un message. Remplacez *mqttTopic* par le sujet à utiliser.

Cette ressource ne prend pas en charge les caractères génériques des rubriques MQTT.

- `mqtt:subscribe`— Accorde l'autorisation de s'abonner aux filtres de sujets MQTT pour recevoir des messages.

Cette opération prend en charge les ressources suivantes :

- `mqtt:topicfilter:mqttTopicFilter`— Limitez l'accès en fonction des sujets MQTT sur lesquels un appareil client peut s'abonner à des messages. Remplacez *mqttTopicFilter* par le filtre de rubrique à utiliser.

Cette ressource prend en charge les caractères génériques + des rubriques # MQTT et MQTT. Pour plus d'informations, consultez les [rubriques relatives au MQTT](#) dans le Guide du AWS IoT Core développeur.

L'appareil client peut s'abonner aux filtres de rubrique exacts que vous autorisez. Par exemple, si vous autorisez l'appareil client à s'abonner à la `mqtt:topicfilter:client/+/status` ressource, il peut s'y abonner, `client/+/status` mais pas `client/client1/status`.

Vous pouvez spécifier le * caractère générique pour autoriser l'accès à toutes les actions.

resources

La liste des ressources permettant d'effectuer les opérations prévues dans cette politique. Spécifiez les ressources qui correspondent aux opérations de cette politique. Par exemple, vous pouvez spécifier une liste de ressources thématiques MQTT (`mqtt:topic:mqttTopic`) dans une politique qui spécifie l'`mqtt:publish` opération.

Vous pouvez spécifier le * caractère générique pour autoriser l'accès à toutes les ressources. Vous ne pouvez pas utiliser le * caractère générique pour faire correspondre des identificateurs de ressources partiels. Par exemple, vous pouvez spécifier `"resources": "*"` , mais vous ne pouvez pas le faire `"resources": "mqtt:clientId:*"` .

statementDescription

(Facultatif) Description de cette déclaration de politique.

Exemple Exemple : mise à jour de la fusion de configurations (à l'aide d'une politique restrictive)

L'exemple de configuration suivant indique d'autoriser les appareils clients dont le nom commence par MyClientDevice à se connecter et ce, publish/subscribe sur tous les sujets.

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyDeviceGroup": {
        "selectionRule": "thingName: MyClientDevice*",
        "policyName": "MyRestrictivePolicy"
      }
    },
    "policies": {
      "MyRestrictivePolicy": {
        "AllowConnect": {
          "statementDescription": "Allow client devices to connect.",
          "operations": [
            "mqtt:connect"
          ],
          "resources": [
            "*"
          ]
        },
        "AllowPublish": {
          "statementDescription": "Allow client devices to publish on test/topic.",
          "operations": [
            "mqtt:publish"
          ],
          "resources": [
            "mqtt:topic:test/topic"
          ]
        },
        "AllowSubscribe": {
          "statementDescription": "Allow client devices to subscribe to test/topic/
response.",
          "operations": [
            "mqtt:subscribe"
          ]
        }
      }
    }
  }
}
```

```
    ],
    "resources": [
      "mqtt:topicfilter:test/topic/response"
    ]
  }
}
}
```

Exemple Exemple : mise à jour de la fusion de configurations (à l'aide d'une politique permissive)

L'exemple de configuration suivant indique d'autoriser tous les appareils clients à se connecter et ce, publish/subscribe sur tous les sujets.

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyPermissiveDeviceGroup": {
        "selectionRule": "thingName: *",
        "policyName": "MyPermissivePolicy"
      }
    },
    "policies": {
      "MyPermissivePolicy": {
        "AllowAll": {
          "statementDescription": "Allow client devices to perform all actions.",
          "operations": [
            "*"
          ],
          "resources": [
            "*"
          ]
        }
      }
    }
  }
}
```


Fichier journal local

Ce composant utilise le même fichier journal que le composant [Greengrass nucleus](#).

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

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

Pour consulter les journaux de ce composant

- Exécutez la commande suivante sur le périphérique principal pour afficher le fichier journal de ce composant en temps réel. Remplacez `/greengrass/v2` ou `C:\greengrass\v2` par le chemin d'accès au dossier AWS IoT Greengrass racine.

Linux

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

Windows (PowerShell)

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

Journal des modifications

Le tableau suivant décrit les modifications apportées à chaque version du composant.

Version	Modifications
2.5.5	Version mise à jour pour la version 2.16.0 de Greengrass Nucleus.
2.5.4	Version mise à jour pour la version 2.15.0 de Greengrass Nucleus.

Version	Modifications
2.5.3	Corrections de bugs et améliorations • Résout un problème empêchant les appareils clients de se connecter au périphérique principal en raison de certificats clients périmés.
2.5.2	Version mise à jour pour la version 2.14.0 de Greengrass Nucleus.
2.5.1	Corrections de bugs et améliorations • Supporte le point de terminaison FIPS.
2.5.0	Nouvelles fonctionnalités • Permet la substitution de <code>\${iot:Connection.Thing.ThingName}</code> variables pour les ressources politiques. • Autorise les ressources politiques avec des caractères génériques tels que <code>mqtt:topic:my*</code>.
2.4.5	Nouvelles fonctionnalités Ajoute la prise en charge des préfixes génériques pour sélectionner des noms d'objets avec le <code>selectionRule</code> paramètre. Corrections de bugs et améliorations Résout un problème selon lequel les certificats ne sont pas mis à jour avec les nouvelles informations de connectivité dans certains cas.
2.4.4	Version mise à jour pour la version 2.12.0 de Greengrass Nucleus.
2.4.3	Version mise à jour pour la version 2.11.0 de Greengrass Nucleus.
2.4.2	Nouvelles fonctionnalités Ajoute une nouvelle option <code>startupTimeoutSeconds</code> de configuration.
2.4.1	Version mise à jour pour la version 2.10.0 de Greengrass Nucleus.

Version	Modifications
2.4.0	Nouvelles fonctionnalités • Ajoute la prise en charge de l'authentification du périphérique client pour émettre des métriques opérationnelles qui seront publiées par l'agent de télémétrie. Corrections de bugs et améliorations • Résout un problème selon lequel l'authentification de l'appareil client prend plus de 10 secondes pour vérifier l'identité d'un appareil client. • Corrections et améliorations mineures supplémentaires.
2.3.2	Corrections de bugs et améliorations • Ajoute la prise en charge de la mise en cache des informations de nom d'hôte afin que le composant génère correctement les sujets de certificat lors du redémarrage en mode hors connexion.
2.3.1	Corrections de bugs et améliorations • Corrige une fuite de mémoire.
2.3.0	 Warning Cette version n'est plus disponible. Les améliorations apportées à cette version sont disponibles dans les versions ultérieures de ce composant. Nouvelles fonctionnalités • Ajoute la prise en charge de l'authentification hors ligne des appareils clients afin qu'ils puissent continuer à se connecter au périphérique principal lorsque celui-ci n'est pas connecté à Internet. • Ajoute la prise en charge de l'autorité de certification fournie par le client que le périphérique principal utilise comme certificat racine pour générer des certificats de courtier MQTT.
2.2.3	Version mise à jour pour la version 2.8.0 de Greengrass Nucleus.

Version	Modifications
2.2.2	Corrections de bugs et améliorations • Résout un problème selon lequel le certificat du serveur MQTT local change plus souvent que prévu dans certains scénarios.
2.2.1	Version mise à jour pour la version 2.7.0 de Greengrass Nucleus.
2.2.0	Nouvelles fonctionnalités • Ajoute la prise en charge de composants personnalisés pour appeler des opérations de communication interprocessus (IPC) afin d'authentifier et d'autoriser les appareils clients. Vous pouvez utiliser ces opérations dans un composant de courtier MQTT personnalisé, par exemple. Pour plus d'informations, voir IPC : Authentifier et autoriser les appareils clients. • Ajoute les <code>threadPoolSize</code> options <code>maxActiveAuthTokens</code> et <code>cloudQueueSize</code>, et que vous pouvez configurer pour ajuster les performances de ce composant.
2.1.0	Nouvelles fonctionnalités • Ajoute l'<code>serverCertificateValiditySeconds</code> option que vous pouvez configurer pour personnaliser la date d'expiration du certificat du serveur de courtage MQTT. Vous pouvez configurer le certificat du serveur pour qu'il expire au bout de 2 à 10 jours. Corrections de bugs et améliorations • Résout les problèmes liés à la façon dont ce composant gère les mises à jour de réinitialisation de configuration. • Résout un problème selon lequel le certificat du serveur MQTT local change plus souvent que prévu dans certains scénarios. Pour appliquer ce correctif, vous devez également utiliser la version 2.1.0 ou ultérieure du composant broker Moquette MQTT. • Améliore les messages enregistrés par ce composant lors de la rotation des certificats. • Version mise à jour pour la version 2.6.0 de Greengrass Nucleus.

Version	Modifications
2.0.4	Version mise à jour pour la version 2.5.0 de Greengrass Nucleus.
2.0.3	Corrections de bugs et améliorations • Les informations d'identification sont désormais actualisées si vous faites pivoter la clé privée de l'appareil principal. • Mises à jour pour rendre les messages du journal plus clairs.
2.0.2	Version mise à jour pour la version 2.4.0 de Greengrass Nucleus.
2.0.1	Version mise à jour pour la version 2.3.0 de Greengrass Nucleus.
2.0.0	Première version.

CloudWatch métriques

Le composant Amazon CloudWatch metrics (`aws.greengrass.Cloudwatch`) publie sur Amazon des métriques personnalisées provenant des principaux appareils de Greengrass. CloudWatch Le composant permet aux composants de publier CloudWatch des métriques, que vous pouvez utiliser pour surveiller et analyser l'environnement de l'appareil principal de Greengrass. Pour plus d'informations, consultez la section [Utilisation CloudWatch des métriques Amazon](#) dans le guide de CloudWatch l'utilisateur Amazon.

Pour publier une CloudWatch métrique avec ce composant, publiez un message dans un sujet auquel ce composant est abonné. Par défaut, ce composant s'abonne à la rubrique de [publication/d'abonnement cloudwatch/metric/put locale](#). Vous pouvez spécifier d'autres sujets, y compris les sujets AWS IoT Core MQTT, lorsque vous déployez ce composant.

Ce composant regroupe les métriques qui se trouvent dans le même espace de noms et les publie à CloudWatch intervalles réguliers.

Note

Ce composant fournit des fonctionnalités similaires à celles du connecteur de CloudWatch métriques de la AWS IoT Greengrass version 1. Pour plus d'informations, consultez

le [connecteur de CloudWatch métriques](#) dans le guide du développeur de la AWS IoT Greengrass V1.

Rubriques

- [Versions](#)
- [Type](#)
- [Système d'exploitation](#)
- [Exigences](#)
- [Dépendances](#)
- [Configuration](#)
- [Données d'entrée](#)
- [Données de sortie](#)
- [Licences](#)
- [Fichier journal local](#)
- [Journal des modifications](#)
- [Consultez aussi](#)

Versions

Les versions de ce composant sont les suivantes :

- 3.2.x
- 3.1.x
- 3.0.x
- 2.1.x
- 2.0.x

Pour plus d'informations sur les modifications apportées à chaque version du composant, consultez le [journal des modifications](#).

Type

v3.x

Ce composant est un composant générique (`aws.greengrass.generic`). Le [noyau Greengrass](#) exécute les scripts de cycle de vie du composant.

v2.x

Ce composant est un composant Lambda (`aws.greengrass.lambda`). [Le noyau Greengrass exécute la fonction Lambda de ce composant à l'aide du composant Lambda Launcher.](#)

Pour de plus amples informations, veuillez consulter [Types de composants](#).

Système d'exploitation

v3.x

Ce composant peut être installé sur les appareils principaux qui exécutent les systèmes d'exploitation suivants :

- Linux
- Windows

v2.x

Ce composant ne peut être installé que sur les appareils principaux de Linux.

Exigences

Ce composant répond aux exigences suivantes :

3.x

- [Python](#) version 3.7 installé sur le périphérique principal et ajouté à la variable d'environnement `PATH`.
- Le [rôle d'appareil Greengrass](#) doit autoriser `cloudwatch:PutMetricData` action, comme illustré dans l'exemple de politique IAM suivant.

JSON

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

Pour plus d'informations, consultez la [référence CloudWatch des autorisations Amazon](#) dans le guide de CloudWatch l'utilisateur Amazon.

2.x

- Votre appareil principal doit répondre aux exigences pour exécuter les fonctions Lambda. Si vous souhaitez que le périphérique principal exécute des fonctions Lambda conteneurisées, le périphérique doit répondre aux exigences requises. Pour de plus amples informations, veuillez consulter [Exigences relatives à la fonction Lambda](#).
- [Python](#) version 3.7 installé sur le périphérique principal et ajouté à la variable d'environnement PATH.
- Le [rôle d'appareil Greengrass](#) doit autoriser l'`cloudwatch:PutMetricData` action, comme illustré dans l'exemple de politique IAM suivant.

JSON

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



```
    }
  ]
}
```

Pour plus d'informations, consultez la [référence CloudWatch des autorisations Amazon](#) dans le guide de CloudWatch l'utilisateur Amazon.

- Pour recevoir les données de sortie de ce composant, vous devez fusionner la mise à jour de configuration suivante pour l'[ancien composant routeur d'abonnement](#) (`aws.greengrass.LegacySubscriptionRouter`) lorsque vous déployez ce composant. Cette configuration indique le sujet dans lequel ce composant publie les réponses.

Legacy subscription router v2.1.x

```
{
  "subscriptions": {
    "aws-greengrass-cloudwatch": {
      "id": "aws-greengrass-cloudwatch",
      "source": "component:aws.greengrass.Cloudwatch",
      "subject": "cloudwatch/metric/put/status",
      "target": "cloud"
    }
  }
}
```

Legacy subscription router v2.0.x

```
{
  "subscriptions": {
    "aws-greengrass-cloudwatch": {
      "id": "aws-greengrass-cloudwatch",
      "source": "arn:aws:lambda:region:aws:function:aws-greengrass-  
cloudwatch:version",
      "subject": "cloudwatch/metric/put/status",
      "target": "cloud"
    }
  }
}
```

- *region* Remplacez-le par Région AWS celui que vous utilisez.
- *version* Remplacez-le par la version de la fonction Lambda exécutée par ce composant. Pour trouver la version de la fonction Lambda, vous devez consulter la recette de la

version de ce composant que vous souhaitez déployer. Ouvrez la page de détails de ce composant dans la [AWS IoT Greengrass console](#) et recherchez la paire clé-valeur de la fonction Lambda. Cette paire clé-valeur contient le nom et la version de la fonction Lambda.

 Important

Vous devez mettre à jour la version de la fonction Lambda sur l'ancien routeur d'abonnement chaque fois que vous déployez ce composant. Cela garantit que vous utilisez la bonne version de la fonction Lambda pour la version du composant que vous déployez.

Pour de plus amples informations, veuillez consulter [Créer des déploiements](#).

Points de terminaison et ports

Ce composant doit être capable d'effectuer des demandes sortantes vers les points de terminaison et les ports suivants, en plus des points de terminaison et des ports requis pour le fonctionnement de base. Pour de plus amples informations, veuillez consulter [Autoriser le trafic des appareils via un proxy ou un pare-feu](#).

Endpoint	Port	Obligatoire	Description
monitoring. <i>region</i> .amazonaws.com	443	Oui	Téléversez CloudWatch des métriques.

Dépendances

Lorsque vous déployez un composant, il déploie AWS IoT Greengrass également des versions compatibles de ses dépendances. Cela signifie que vous devez satisfaire aux exigences relatives au composant et à toutes ses dépendances pour réussir le déploiement du composant. Cette section répertorie les dépendances des [versions publiées](#) de ce composant et les contraintes de version sémantiques qui définissent les versions des composants pour chaque dépendance. Vous

pouvez également consulter les dépendances de chaque version du composant dans la [AWS IoT Greengrass console](#). Sur la page de détails du composant, recherchez la liste des dépendances.

3.2.0

Le tableau suivant répertorie les dépendances pour les versions 3.2.0 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,0.0 < 3,0.0$	Flexible
Service d'échange de jetons	$\geq 0,0.0$	Stricte

3.0.0 - 3.1.0

Le tableau suivant répertorie les dépendances pour les versions 3.0.0 à 3.1.0 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,0.0 < 3,0.0$	Flexible
Service d'échange de jetons	$\geq 0,0.0$	Stricte

2.1.4 - 2.1.9

Le tableau suivant répertorie les dépendances pour les versions 2.1.4 à 2.1.9 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,0.0 < 3,0.0$	Stricte
Lanceur Lambda	$\wedge 2,0.0$	Stricte
Environnements d'exécution (runtimes) Lambda	$\wedge 2,0.0$	Flexible
Service d'échange de jetons	$\wedge 2,0.0$	Stricte

2.1.4 - 2.1.8

Le tableau suivant répertorie les dépendances pour les versions 2.1.4 et 2.1.8 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,0.0 < 3,0.0$	Stricte
Lanceur Lambda	$\wedge 2,0.0$	Stricte
Environnements d'exécution (runtimes) Lambda	$\wedge 2,0.0$	Flexible
Service d'échange de jetons	$\wedge 2,0.0$	Stricte

2.1.2 - 2.1.3

Le tableau suivant répertorie les dépendances pour les versions 2.1.2 et 2.1.3 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,0.0 < 2,8.0$	Stricte
Lanceur Lambda	$\wedge 2,0.0$	Stricte
Environnements d'exécution (runtimes) Lambda	$\wedge 2,0.0$	Flexible
Service d'échange de jetons	$\wedge 2,0.0$	Stricte

2.1.1

Le tableau suivant répertorie les dépendances pour la version 2.1.1 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,0.0 < 2,7.0$	Stricte

Dépendance	Versions compatibles	Type de dépendance
Lanceur Lambda	^2,0.0	Stricte
Environnements d'exécution (runtimes) Lambda	^2,0.0	Flexible
Service d'échange de jetons	^2,0.0	Stricte

2.0.8 - 2.1.0

Le tableau suivant répertorie les dépendances pour les versions 2.0.8 à 2.1.0 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,6.0	Stricte
Lanceur Lambda	^2,0.0	Stricte
Environnements d'exécution (runtimes) Lambda	^2,0.0	Flexible
Service d'échange de jetons	^2,0.0	Stricte

2.0.7

Le tableau suivant répertorie les dépendances pour la version 2.0.7 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,5.0	Stricte
Lanceur Lambda	^2,0.0	Stricte
Environnements d'exécution (runtimes) Lambda	^2,0.0	Flexible
Service d'échange de jetons	^2,0.0	Stricte

2.0.6

Le tableau suivant répertorie les dépendances pour la version 2.0.6 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,4.0	Stricte
Lanceur Lambda	^2,0.0	Stricte
Environnements d'exécution (runtimes) Lambda	^2,0.0	Flexible
Service d'échange de jetons	^2,0.0	Stricte

2.0.5

Le tableau suivant répertorie les dépendances pour la version 2.0.5 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,3.0	Stricte
Lanceur Lambda	^2,0.0	Stricte
Environnements d'exécution (runtimes) Lambda	^2,0.0	Flexible
Service d'échange de jetons	^2,0.0	Stricte

2.0.4

Le tableau suivant répertorie les dépendances pour la version 2.0.4 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,2.0	Stricte
Lanceur Lambda	^2,0.0	Stricte

Dépendance	Versions compatibles	Type de dépendance
Environnements d'exécution (runtimes) Lambda	^2,0.0	Flexible
Service d'échange de jetons	^2,0.0	Stricte

2.0.3

Le tableau suivant répertorie les dépendances pour la version 2.0.3 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0,3 <2,10	Stricte
Lanceur Lambda	>=10,0	Stricte
Environnements d'exécution (runtimes) Lambda	>=10,0	Flexible
Service d'échange de jetons	>=10,0	Stricte

Pour plus d'informations sur les dépendances des composants, consultez la [référence de la recette des composants](#).

Configuration

Ce composant fournit les paramètres de configuration suivants que vous pouvez personnaliser lorsque vous déployez le composant.

v3.x

PublishInterval

(Facultatif) Le nombre maximal de secondes à attendre avant que le composant publie des métriques par lots pour un espace de noms donné. Pour configurer le composant afin qu'il publie les métriques au fur et à mesure qu'il les reçoit, c'est-à-dire sans traitement par lots, spécifiez 0.

Le composant publie CloudWatch après avoir reçu 20 métriques dans le même espace de noms ou après l'intervalle que vous spécifiez.

 Note

Le composant ne précise pas l'ordre dans lequel les événements sont publiés.

Cette valeur peut être de 900 secondes au maximum.

Par défaut : 10 secondes

`MaxMetricsToRetain`

(Facultatif) Le nombre maximum de métriques dans tous les espaces de noms à enregistrer en mémoire avant que le composant ne les remplace par des métriques plus récentes.

Cette limite s'applique lorsque l'appareil principal n'est pas connecté à Internet. Le composant met donc en mémoire tampon les métriques pour les publier ultérieurement. Lorsque la mémoire tampon est pleine, le composant remplace les mesures les plus anciennes par les plus récentes. Les métriques d'un espace de noms donné remplacent uniquement les métriques du même espace de noms.

 Note

Si le processus hôte du composant est interrompu, le composant n'enregistre pas les métriques. Cela peut se produire lors d'un déploiement ou lorsque le périphérique principal redémarre, par exemple.

Cette valeur doit être d'au moins 2 000 métriques.

Par défaut : 5 000 métriques

`InputTopic`

(Facultatif) Rubrique à laquelle le composant s'abonne pour recevoir des messages. Si vous spécifiez `true` pour `PubSubToIoTCore`, vous pouvez utiliser des caractères génériques MQTT (+ et #) dans cette rubrique.

Valeur par défaut : `cloudwatch/metric/put`

OutputTopic

(Facultatif) Rubrique pour laquelle le composant publie les réponses de statut.

Valeur par défaut : `cloudwatch/metric/put/status`

PubSubToIoTCore

(Facultatif) Valeur de chaîne qui définit s'il faut publier des sujets AWS IoT Core MQTT et s'y abonner. Les valeurs prises en charge sont `true` et `false`.

Valeur par défaut : `false`

LogLevel

(Facultatif) Le niveau de journalisation du composant. Choisissez parmi les niveaux de journalisation suivants, listés ici par ordre de niveau :

- `DEBUG`
- `INFO`
- `WARNING`
- `ERROR`
- `CRITICAL`

Valeur par défaut : `INFO`

UseInstaller

(Facultatif) Valeur booléenne qui définit s'il faut utiliser le script d'installation dans ce composant pour installer les dépendances du SDK de ce composant.

Définissez cette valeur sur `false` si vous souhaitez utiliser un script personnalisé pour installer des dépendances ou si vous souhaitez inclure des dépendances d'exécution dans une image Linux prédéfinie. Pour utiliser ce composant, vous devez installer les bibliothèques suivantes, y compris les dépendances éventuelles, et les mettre à la disposition de l'utilisateur du système Greengrass par défaut.

- [Kit SDK des appareils AWS IoT v2 pour Python](#)
- [AWS SDK pour Python \(Boto3\)](#)

Valeur par défaut : `true`

PublishRegion

(Facultatif) Le Région AWS site sur lequel publier CloudWatch les statistiques. Cette valeur remplace la région par défaut pour le périphérique principal. Ce paramètre est requis uniquement pour les métriques interrégionales.

accessControl

(Facultatif) Objet contenant la [politique d'autorisation](#) qui permet au composant de publier et de s'abonner aux rubriques spécifiées. Si vous spécifiez des valeurs personnalisées pour InputTopic et OutputTopic, vous devez mettre à jour les valeurs des ressources dans cet objet.

Valeur par défaut :

```
{
  "aws.greengrass.ipc.pubsub": {
    "aws.greengrass.Cloudwatch:pubsub:1": {
      "policyDescription": "Allows access to subscribe to input topics.",
      "operations": [
        "aws.greengrass#SubscribeToTopic"
      ],
      "resources": [
        "cloudwatch/metric/put"
      ]
    },
    "aws.greengrass.Cloudwatch:pubsub:2": {
      "policyDescription": "Allows access to publish to output topics.",
      "operations": [
        "aws.greengrass#PublishToTopic"
      ],
      "resources": [
        "cloudwatch/metric/put/status"
      ]
    }
  },
  "aws.greengrass.ipc.mqttproxy": {
    "aws.greengrass.Cloudwatch:mqttproxy:1": {
      "policyDescription": "Allows access to subscribe to input topics.",
      "operations": [
        "aws.greengrass#SubscribeToIoTCore"
      ],
      "resources": [
```

```

        "cloudwatch/metric/put"
    ],
    },
    "aws.greengrass.Cloudwatch:mqttproxy:2": {
        "policyDescription": "Allows access to publish to output topics.",
        "operations": [
            "aws.greengrass#PublishToIoTCore"
        ],
        "resources": [
            "cloudwatch/metric/put/status"
        ]
    }
}
}

```

Exemple Exemple : mise à jour de la fusion de configurations

```

{
  "PublishInterval": 0,
  "PubSubToIoTCore": true
}

```

v2.x

Note

La configuration par défaut de ce composant inclut les paramètres de la fonction Lambda. Nous vous recommandons de modifier uniquement les paramètres suivants pour configurer ce composant sur vos appareils.

lambdaParams

Objet contenant les paramètres de la fonction Lambda de ce composant. Cet objet contient les informations suivantes :

EnvironmentVariables

Objet contenant les paramètres de la fonction Lambda. Cet objet contient les informations suivantes :

PUBLISH_INTERVAL

(Facultatif) Le nombre maximal de secondes à attendre avant que le composant publie des métriques par lots pour un espace de noms donné. Pour configurer le composant afin qu'il publie les métriques au fur et à mesure qu'il les reçoit, c'est-à-dire sans traitement par lots, spécifiez 0.

Le composant publie CloudWatch après avoir reçu 20 métriques dans le même espace de noms ou après l'intervalle que vous spécifiez.

Note

Le composant ne garantit pas l'ordre dans lequel les événements sont publiés.

Cette valeur peut être d'au plus 900 secondes.

Par défaut : 10 secondes

MAX_METRICS_TO_RETAIN

(Facultatif) Le nombre maximum de métriques dans tous les espaces de noms à enregistrer en mémoire avant que le composant ne les remplace par des métriques plus récentes.

Cette limite s'applique lorsque l'appareil principal n'est pas connecté à Internet. Le composant met donc en mémoire tampon les métriques pour les publier ultérieurement. Lorsque la mémoire tampon est pleine, le composant remplace les mesures les plus anciennes par les plus récentes. Les métriques d'un espace de noms donné remplacent uniquement les métriques du même espace de noms.

Note

Si le processus hôte du composant est interrompu, le composant n'enregistre pas les métriques. Cela peut se produire lors d'un déploiement ou lorsque le périphérique principal redémarre, par exemple.

Cette valeur doit être d'au moins 2 000 métriques.

Par défaut : 5 000 métriques

PUBLISH_REGION

(Facultatif) Le Région AWS site sur lequel publier CloudWatch les statistiques. Cette valeur remplace la région par défaut pour le périphérique principal. Ce paramètre est requis uniquement pour les métriques interrégionales.

containerMode

(Facultatif) Mode de conteneurisation de ce composant. Sélectionnez l'une des options suivantes :

- `NoContainer`— Le composant ne s'exécute pas dans un environnement d'exécution isolé.
- `GreengrassContainer`— Le composant s'exécute dans un environnement d'exécution isolé à l'intérieur du AWS IoT Greengrass conteneur.

Valeur par défaut : `GreengrassContainer`

containerParams

(Facultatif) Objet contenant les paramètres du conteneur pour ce composant. Le composant utilise ces paramètres si vous le spécifiez `GreengrassContainer` pour `containerMode`.

Cet objet contient les informations suivantes :

memorySize

(Facultatif) La quantité de mémoire (en kilo-octets) à allouer au composant.

La valeur par défaut est de 64 Mo (65 535 Ko).

pubsubTopics

(Facultatif) Objet contenant les rubriques auxquelles le composant s'abonne pour recevoir des messages. Vous pouvez spécifier chaque sujet et indiquer si le composant est abonné à des sujets MQTT provenant de sujets locaux AWS IoT Core ou locaux publish/subscribe .

Cet objet contient les informations suivantes :

0— Il s'agit d'un index de tableau sous forme de chaîne.

Objet contenant les informations suivantes :

type

(Facultatif) Type de publish/subscribe messagerie utilisé par ce composant pour s'abonner aux messages. Sélectionnez l'une des options suivantes :

- PUB_SUB – Abonnez-vous aux messages locaux de publication/abonnement. Si vous choisissez cette option, le sujet ne peut pas contenir de caractères génériques MQTT. Pour plus d'informations sur la façon d'envoyer des messages à partir d'un composant personnalisé lorsque vous spécifiez cette option, consultez [Publier/souscrire des messages locaux](#).
- IOT_CORE— Abonnez-vous aux messages AWS IoT Core MQTT. Si vous choisissez cette option, le sujet peut contenir des caractères génériques MQTT. Pour plus d'informations sur la façon d'envoyer des messages à partir de composants personnalisés lorsque vous spécifiez cette option, consultez [Publier/souscrire AWS IoT Core des messages MQTT](#).

Valeur par défaut : PUB_SUB

topic

(Facultatif) Rubrique à laquelle le composant s'abonne pour recevoir des messages. Si vous spécifiez IotCore pour type, vous pouvez utiliser des caractères génériques MQTT (+et#) dans cette rubrique.

Exemple Exemple : mise à jour de fusion de configuration (mode conteneur)

```
{
  "containerMode": "GreengrassContainer"
}
```

Exemple Exemple : mise à jour de fusion de configuration (pas de mode conteneur)

```
{
  "containerMode": "NoContainer"
}
```

Données d'entrée

Ce composant accepte les métriques relatives au sujet suivant et les publie sur CloudWatch. Par défaut, ce composant s'abonne aux publish/subscribe messages locaux. Pour plus d'informations sur la façon de publier des messages vers ce composant à partir de vos composants personnalisés, consultez [Publier/souscrire des messages locaux](#).

À partir de la version du composant v3.0.0, vous pouvez éventuellement configurer ce composant pour qu'il s'abonne à un sujet MQTT en définissant le paramètre de PubSubToIoTCore configuration sur `true`. Pour plus d'informations sur la publication de messages sur une rubrique MQTT dans vos composants personnalisés, consultez [Publier/souscrire AWS IoT Core des messages MQTT](#).

Rubrique par défaut : `cloudwatch/metric/put`

Le message accepte les propriétés suivantes. Les messages d'entrée doivent être au format JSON.

`request`

La métrique indiquée 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](#) opération.

Type : object qui contient les informations 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.

Type : `string`

Modèle valide : `[^:]*`

`metricData`

Les données pour la métrique.

Type : object qui contient les informations suivantes :

`metricName`

Le nom de la métrique.

Type : string

value

Valeur de la métrique.

 Note

CloudWatch rejette les valeurs trop petites ou trop grandes. La valeur doit être comprise entre $8.515920e-109$ et $1.174271e+108$ (Base 10) ou $2e-360$ et $2e360$ (Base 2). CloudWatch ne prend pas en charge les valeurs spéciales telles que NaN+Infinity, et-Infinity.

Type : double

dimensions

(Facultatif) Dimensions de la métrique. Les dimensions fournissent des informations supplémentaires sur la métrique et ses données. Une métrique peut définir jusqu'à 10 dimensions.

Ce composant inclut automatiquement une dimension nommée `coreName`, dont la valeur est le nom du périphérique principal.

Type : array d'objets contenant chacun les informations suivantes :

name

(Facultatif) Le nom de la dimension.

Type : string

value

(Facultatif) La valeur de la dimension.

Type : string

timestamp

(Facultatif) Heure à laquelle les données métriques ont été reçues, exprimée en secondes à l'époque Unix.

La valeur par défaut est l'heure à laquelle le composant reçoit le message.

Type : `double`

 Note

Si vous utilisez entre les versions 2.0.3 et 2.0.7 de ce composant, 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.

`unit`

(Facultatif) L'unité de la métrique.

Type : `string`

Valeurs

valides : `Seconds`, `Microseconds`, `Milliseconds`, `Bytes`, `Kilobytes`, `Megabytes`, `Gigabytes`, `Second`, `Kilobytes/Second`, `Megabytes/Second`, `Gigabytes/Second`, `Terabytes/Second`, `Bits/Second`, `Kilobits/Second`, `Megabits/Second`, `Gigabits/Second`, `Terabits/Second`, `Count/Second`, `None`

La valeur par défaut est `None` .

 Note

Tous les quotas qui s'appliquent à l' `CloudWatch PutMetricDataAPI` s'appliquent aux métriques que vous publiez avec ce composant. Les quotas suivants sont particulièrement importants :

- Limite de 40 Ko sur la charge utile de l'API
- 20 métriques par requête d'API
- 150 transactions par seconde (TPS) pour l'API `PutMetricData`

Pour plus d'informations, consultez la section [Quotas de CloudWatch service](#) dans le Guide de CloudWatch l'utilisateur.

Exemple 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 composant publie par défaut les réponses sous forme de données de sortie sur le publish/subscribe sujet local suivant. Pour plus d'informations sur la façon de s'abonner à des messages sur ce sujet dans vos composants personnalisés, consultez [Publier/souscrire des messages locaux](#).

Vous pouvez éventuellement configurer ce composant pour qu'il soit publié dans une rubrique MQTT en définissant le paramètre PubSubToIoTCore de configuration sur `true`. Pour plus d'informations sur l'abonnement à des messages sur un sujet MQTT dans vos composants personnalisés, consultez [Publier/souscrire AWS IoT Core des messages MQTT](#).

Note

Les versions 2.0.x des composants publient les réponses sous forme de données de sortie sur un sujet MQTT par défaut. Vous devez spécifier le sujet tel qu'il `subject` figure dans la configuration de l'[ancien composant routeur d'abonnement](#).

Rubrique par défaut : `cloudwatch/metric/put/status`

Exemple 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 Exemple de sortie : échec

```
{
  "response" : {
    "namespace": "Greengrass",
    "error": "InvalidInputException",
    "error_message": "cw metric is invalid",
    "status": "fail"
  }
}
```

Note

Si le composant détecte une erreur qui peut être réessayée, telle qu'une erreur de connexion, il réessaie de publier dans le lot suivant.

Licences

Ce composant inclut les logiciels/licences tiers suivants :

- [AWS SDK pour 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 composant est publié dans le cadre du contrat de [licence logicielle Greengrass Core](#).

Fichier journal local

Ce composant utilise le fichier journal suivant.

Linux

```
/greengrass/v2/logs/aws.greengrass.Cloudwatch.log
```

Windows

```
C:\greengrass\v2\logs\aws.greengrass.Cloudwatch.log
```

Pour consulter les journaux de ce composant

- Exécutez la commande suivante sur le périphérique principal pour afficher le fichier journal de ce composant en temps réel. Remplacez */greengrass/v2* ou *C:\greengrass\v2* par le chemin d'accès au dossier AWS IoT Greengrass racine.

Linux

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

Windows (PowerShell)

```
Get-Content C:\greengrass\v2\logs\aws.greengrass.Cloudwatch.log -Tail 10 -Wait
```

Journal des modifications

Le tableau suivant décrit les modifications apportées à chaque version du composant.

v3.x

Version	Modifications
3.2.0	Nouvelles fonctionnalités • Ajout de supports de recettes pour Greengrass Nucleus Lite
3.1.0	Corrections de bogues et améliorations • Ajoute la prise en charge des configurations de proxy réseau HTTPS. Pour plus d'informations, consultez Connexion au port 443 ou via un proxy réseau et Permettre au périphérique principal de faire confiance à un proxy HTTPS.
3.0.0	Cette version du composant CloudWatch metrics attend des paramètres de configuration différents de ceux de la version 2.x. Si vous utilisez une configuration autre que celle par défaut pour la version 2.x et que vous souhaitez passer de la version 2.x à la version 3.x, vous devez mettre à jour la configuration du composant. Pour plus d'informations, consultez la section Configuration CloudWatch des composants métriques. Nouvelles fonctionnalités • Ajoute la prise en charge des appareils principaux qui exécutent Windows. • Change le type de composant Lambda en composant générique. Ce composant ne dépend désormais plus de l'ancien composant routeur d'abonnement pour créer des abonnements. • Ajoute un nouveau paramètre de InputTopic configuration pour spécifier le sujet auquel le composant s'abonne pour recevoir des messages. • Ajoute un nouveau paramètre de OutputTopic configuration pour spécifier le sujet dans lequel le composant publie les réponses d'état. • Ajoute un nouveau paramètre PubSubToIoTCore de configuration pour spécifier s'il faut publier des sujets AWS IoT Core MQTT et s'y abonner.

Version	Modifications
	Ajoute le nouveau paramètre <code>UseInstaller</code> de configuration qui permet de désactiver éventuellement le script d'installation qui installe les dépendances des composants. Corrections de bogues et améliorations Ajoute la prise en charge des horodatages dupliqués dans les données d'entrée.

v2.x

Version	Modifications
2.1.8	Version mise à jour pour la version 2.13.0 de Greengrass Nucleus.
2.1.3	Version mise à jour pour la version 2.11.0 de Greengrass Nucleus.
2.1.2	Version mise à jour pour la version 2.7.0 de Greengrass Nucleus.
2.1.1	Version mise à jour pour la version 2.6.0 de Greengrass Nucleus.
2.1.0	Nouvelles fonctionnalités Ajoute la prise en charge des configurations de proxy réseau HTTPS. Pour plus d'informations, consultez Connexion au port 443 ou via un proxy réseau et Permettre au périphérique principal de faire confiance à un proxy HTTPS.
2.0.8	Corrections de bogues et améliorations - Ajoute la prise en charge des horodatages dupliqués dans les données d'entrée. - Version mise à jour pour la version 2.5.0 de Greengrass Nucleus.
2.0.7	Version mise à jour pour la version 2.4.0 de Greengrass Nucleus.
2.0.6	Version mise à jour pour la version 2.3.0 de Greengrass Nucleus.
2.0.5	Version mise à jour pour la version 2.2.0 de Greengrass Nucleus.

Version	Modifications
2.0.4	Version mise à jour pour la version 2.1.0 de Greengrass Nucleus.
2.0.3	Première version.

Consultez aussi

- [Utilisation des CloudWatch métriques Amazon](#) dans le guide de CloudWatch l'utilisateur Amazon
- [PutMetricData](#) dans le Amazon CloudWatch API Reference

AWS IoT Device Defender

Le AWS IoT Device Defender composant (`aws.greengrass.DeviceDefender`) informe les administrateurs des modifications de l'état des appareils principaux de Greengrass. Cela peut permettre d'identifier un comportement inhabituel qui pourrait indiquer un appareil compromis. Pour plus d'informations, consultez [AWS IoT Device Defender](#) dans le Guide du développeur AWS IoT Core .

Ce composant lit les métriques du système sur le périphérique principal. Ensuite, il publie les métriques sur AWS IoT Device Defender. Pour plus d'informations sur la façon de lire et d'interpréter les métriques rapportées par ce composant, consultez les [spécifications du document relatif aux métriques des appareils](#) dans le Guide du AWS IoT Core développeur.

Note

Ce composant fournit des fonctionnalités similaires à celles du connecteur Device Defender dans AWS IoT Greengrass V1. Pour plus d'informations, consultez la section relative au [connecteur Device Defender](#) dans le guide du AWS IoT Greengrass V1 développeur.

Rubriques

- [Versions](#)
- [Type](#)
- [Système d'exploitation](#)
- [Prérequis](#)

- [Dépendances](#)
- [Configuration](#)
- [Données d'entrée](#)
- [Données de sortie](#)
- [Fichier journal local](#)
- [Licences](#)
- [Journal des modifications](#)

Versions

Les versions de ce composant sont les suivantes :

- 3.1.x
- 3,0. x
- 2,0.x

Pour plus d'informations sur les modifications apportées à chaque version du composant, consultez le [journal des modifications](#).

Type

v3.x

Ce composant est un composant générique (`aws.greengrass.generic`). Le [noyau Greengrass](#) exécute les scripts de cycle de vie du composant.

v2.x

Ce composant est un composant Lambda () `aws.greengrass.lambda`. [Le noyau Greengrass exécute la fonction Lambda de ce composant à l'aide du composant Lambda Launcher.](#)

Pour de plus amples informations, veuillez consulter [Types de composants](#).

Système d'exploitation

v3.x

Ce composant peut être installé sur les appareils principaux qui exécutent les systèmes d'exploitation suivants :

- Linux
- Windows

v2.x

Ce composant ne peut être installé que sur les appareils principaux de Linux.

Prérequis

Ce composant répond aux exigences suivantes :

v3.x

- [Python](#) version 3.7 installé sur le périphérique principal et ajouté à la variable d'environnement PATH.
- AWS IoT Device Defender configuré pour utiliser la fonction de détection afin de surveiller les violations. Pour plus d'informations, consultez la section [Detect](#) dans le guide du AWS IoT Core développeur.

v2.x

- Votre appareil principal doit répondre aux exigences pour exécuter les fonctions Lambda. Si vous souhaitez que le périphérique principal exécute des fonctions Lambda conteneurisées, le périphérique doit répondre aux exigences requises. Pour de plus amples informations, veuillez consulter [Exigences relatives à la fonction Lambda](#).
- [Python](#) version 3.7 installé sur le périphérique principal et ajouté à la variable d'environnement PATH.
- AWS IoT Device Defender configuré pour utiliser la fonction de détection afin de surveiller les violations. Pour plus d'informations, consultez la section [Detect](#) dans le guide du AWS IoT Core développeur.

- La bibliothèque [psutil](#) installée sur le périphérique principal. La version 5.7.0 est la dernière version dont le fonctionnement avec le composant a été vérifié.
- La bibliothèque [cbor](#) installée sur le périphérique principal. La version 1.0.0 est la dernière version dont le fonctionnement avec le composant a été vérifié.
- Pour recevoir les données de sortie de ce composant, vous devez fusionner la mise à jour de configuration suivante pour l'[ancien composant routeur d'abonnement](#) (`aws.greengrass.LegacySubscriptionRouter`) lorsque vous déployez ce composant. Cette configuration indique le sujet dans lequel ce composant publie les réponses.

Legacy subscription router v2.1.x

```
{
  "subscriptions": {
    "aws-greengrass-device-defender": {
      "id": "aws-greengrass-device-defender",
      "source": "component:aws.greengrass.DeviceDefender",
      "subject": "$aws/things/+/defender/metrics/json",
      "target": "cloud"
    }
  }
}
```

Legacy subscription router v2.0.x

```
{
  "subscriptions": {
    "aws-greengrass-device-defender": {
      "id": "aws-greengrass-device-defender",
      "source": "arn:aws:lambda:region:aws:function:aws-greengrass-device-defender:version",
      "subject": "$aws/things/+/defender/metrics/json",
      "target": "cloud"
    }
  }
}
```

- *region* Remplacez-le par Région AWS celui que vous utilisez.
- *version* Remplacez-le par la version de la fonction Lambda exécutée par ce composant. Pour trouver la version de la fonction Lambda, vous devez consulter la recette de la version de ce composant que vous souhaitez déployer. Ouvrez la page de détails de ce

composant dans la [AWS IoT Greengrass console](#) et recherchez la paire clé-valeur de la fonction Lambda. Cette paire clé-valeur contient le nom et la version de la fonction Lambda.

⚠ Important

Vous devez mettre à jour la version de la fonction Lambda sur l'ancien routeur d'abonnement chaque fois que vous déployez ce composant. Cela garantit que vous utilisez la bonne version de la fonction Lambda pour la version du composant que vous déployez.

Pour de plus amples informations, veuillez consulter [Créer des déploiements](#).

Dépendances

Lorsque vous déployez un composant, il déploie AWS IoT Greengrass également des versions compatibles de ses dépendances. Cela signifie que vous devez satisfaire aux exigences relatives au composant et à toutes ses dépendances pour réussir le déploiement du composant. Cette section répertorie les dépendances des [versions publiées](#) de ce composant et les contraintes de version sémantiques qui définissent les versions des composants pour chaque dépendance. Vous pouvez également consulter les dépendances de chaque version du composant dans la [AWS IoT Greengrass console](#). Sur la page de détails du composant, recherchez la liste des dépendances.

3.1.1

Le tableau suivant répertorie les dépendances pour la version 3.1.1 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,0.0 < 3,0.0$	Flexible
Service d'échange de jetons	$\geq 0,0.0$	Stricte

3.0.0 - 3.0.2

Le tableau suivant répertorie les dépendances pour les versions 3.0.0 à 3.0.2 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <3,0.0	Flexible
Service d'échange de jetons	>=0,0.0	Stricte

2.0.12 - 2.0.17

Le tableau suivant répertorie les dépendances pour les versions 2.0.12 à 2.0.17 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <3,0.0	Stricte
Lanceur Lambda	^2,0.0	Stricte
Environnements d'exécution (runtimes) Lambda	^2,0.0	Flexible
Service d'échange de jetons	^2,0.0	Stricte

2.0.12 - 2.0.16

Le tableau suivant répertorie les dépendances pour la version 2.0.16 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <3,0.0	Stricte
Lanceur Lambda	^2,0.0	Stricte
Environnements d'exécution (runtimes) Lambda	^2,0.0	Flexible
Service d'échange de jetons	^2,0.0	Stricte

2.0.10 - 2.0.11

Le tableau suivant répertorie les dépendances pour les versions 2.0.10 et 2.0.11 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,0.0 < 2,8.0$	Stricte
Lanceur Lambda	$\wedge 2,0.0$	Stricte
Environnements d'exécution (runtimes) Lambda	$\wedge 2,0.0$	Flexible
Service d'échange de jetons	$\wedge 2,0.0$	Stricte

2.0.9

Le tableau suivant répertorie les dépendances pour la version 2.0.9 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,0.0 < 2,7.0$	Stricte
Lanceur Lambda	$\wedge 2,0.0$	Stricte
Environnements d'exécution (runtimes) Lambda	$\wedge 2,0.0$	Flexible
Service d'échange de jetons	$\wedge 2,0.0$	Stricte

2.0.8

Le tableau suivant répertorie les dépendances pour la version 2.0.8 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,0.0 < 2,6.0$	Stricte

Dépendance	Versions compatibles	Type de dépendance
Lanceur Lambda	^2,0.0	Stricte
Environnements d'exécution (runtimes) Lambda	^2,0.0	Flexible
Service d'échange de jetons	^2,0.0	Stricte

2.0.7

Le tableau suivant répertorie les dépendances pour la version 2.0.7 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,5.0	Stricte
Lanceur Lambda	^2,0.0	Stricte
Environnements d'exécution (runtimes) Lambda	^2,0.0	Flexible
Service d'échange de jetons	^2,0.0	Stricte

2.0.6

Le tableau suivant répertorie les dépendances pour la version 2.0.6 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,4.0	Stricte
Lanceur Lambda	^2,0.0	Stricte
Environnements d'exécution (runtimes) Lambda	^2,0.0	Flexible
Service d'échange de jetons	^2,0.0	Stricte

2.0.5

Le tableau suivant répertorie les dépendances pour la version 2.0.5 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,0.0 < 2,3.0$	Stricte
Lanceur Lambda	$\wedge 2,0.0$	Stricte
Environnements d'exécution (runtimes) Lambda	$\wedge 2,0.0$	Flexible
Service d'échange de jetons	$\wedge 2,0.0$	Stricte

2.0.4

Le tableau suivant répertorie les dépendances pour la version 2.0.4 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,0.0 < 2,2.0$	Stricte
Lanceur Lambda	$\wedge 2,0.0$	Stricte
Environnements d'exécution (runtimes) Lambda	$\wedge 2,0.0$	Flexible
Service d'échange de jetons	$\wedge 2,0.0$	Stricte

2.0.3

Le tableau suivant répertorie les dépendances pour la version 2.0.3 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,0,3 < 2,10$	Stricte
Lanceur Lambda	$\geq 10,0$	Stricte

Dépendance	Versions compatibles	Type de dépendance
Environnements d'exécution (runtimes) Lambda	>=10,0	Flexible
Service d'échange de jetons	>=10,0	Stricte

Pour plus d'informations sur les dépendances des composants, consultez la [référence de la recette des composants](#).

Configuration

Ce composant fournit les paramètres de configuration suivants que vous pouvez personnaliser lorsque vous déployez le composant.

v3.x

PublishRetryCount

Le nombre de fois où la publication sera réessayée. Cette fonctionnalité est disponible dans la version 3.1.1.

Le minimum est 0.

Le maximum est de 72.

Par défaut: 5

SampleIntervalSeconds

(Facultatif) Durée en secondes entre chaque cycle pendant lequel le composant collecte et rapporte des métriques.

La valeur minimale est de 300 secondes (5 minutes).

Par défaut : 300 secondes

UseInstaller

(Facultatif) Valeur booléenne qui définit s'il faut utiliser le script d'installation dans ce composant pour installer les dépendances de ce composant.

Définissez cette valeur sur `false` si vous souhaitez utiliser un script personnalisé pour installer des dépendances ou si vous souhaitez inclure des dépendances d'exécution dans une image Linux prédéfinie. Pour utiliser ce composant, vous devez installer les bibliothèques suivantes, y compris les dépendances éventuelles, et les mettre à la disposition de l'utilisateur du système Greengrass par défaut.

- [Kit SDK des appareils AWS IoT v2 pour Python](#)
- bibliothèque [cbor](#). La version 1.0.0 est la dernière version dont le fonctionnement avec le composant a été vérifié.
- [bibliothèque psutil](#). La version 5.7.0 est la dernière version dont le fonctionnement avec le composant a été vérifié.

 Note

Si vous utilisez la version 3.0.0 ou 3.0.1 de ce composant sur des appareils principaux que vous configurez pour utiliser un proxy HTTPS, vous devez définir cette valeur sur `false`. Le script d'installation ne prend pas en charge le fonctionnement derrière un proxy HTTPS dans ces versions de ce composant.

Par défaut : `true`

v2.x

 Note

La configuration par défaut de ce composant inclut les paramètres de la fonction Lambda. Nous vous recommandons de modifier uniquement les paramètres suivants pour configurer ce composant sur vos appareils.

`lambdaParams`

Objet contenant les paramètres de la fonction Lambda de ce composant. Cet objet contient les informations suivantes :

EnvironmentVariables

Objet contenant les paramètres de la fonction Lambda. Cet objet contient les informations suivantes :

PROCFS_PATH

(Facultatif) Le chemin d'accès au `/proc` dossier.

- Pour exécuter ce composant dans un conteneur, utilisez la valeur par défaut, `/host-proc`. Le composant s'exécute dans un conteneur par défaut.
- Pour exécuter ce composant sans mode conteneur, spécifiez `/proc` ce paramètre.

Par défaut : `/host-proc`. Il s'agit du chemin par défaut où ce composant monte le `/proc` dossier dans le conteneur.

Note

Ce composant dispose d'un accès en lecture seule à ce dossier.

SAMPLE_INTERVAL_SECONDS

(Facultatif) Durée en secondes entre chaque cycle pendant lequel le composant collecte et rapporte des métriques.

La valeur minimale est de 300 secondes (5 minutes).

Par défaut : 300 secondes

containerMode

(Facultatif) Mode de conteneurisation de ce composant. Sélectionnez parmi les options suivantes :

- `GreengrassContainer`— Le composant s'exécute dans un environnement d'exécution isolé à l'intérieur du AWS IoT Greengrass conteneur.
- `NoContainer`— Le composant ne s'exécute pas dans un environnement d'exécution isolé.

Si vous spécifiez cette option, vous devez spécifier `/proc` le paramètre de variable d'`PROCFS_PATH` environnement.

Par défaut : GreengrassContainer

containerParams

(Facultatif) Objet contenant les paramètres du conteneur pour ce composant. Le composant utilise ces paramètres si vous le spécifiez GreengrassContainer pour containerMode.

Cet objet contient les informations suivantes :

memorySize

(Facultatif) La quantité de mémoire (en kilo-octets) à allouer au composant.

La valeur par défaut est de 50 000 Ko.

pubsubTopics

(Facultatif) Objet contenant les rubriques auxquelles le composant s'abonne pour recevoir des messages. Vous pouvez spécifier chaque sujet et indiquer si le composant est abonné à des sujets MQTT provenant de sujets locaux AWS IoT Core ou locaux publish/subscribe .

Cet objet contient les informations suivantes :

0— Il s'agit d'un index de tableau sous forme de chaîne.

Un objet qui contient les informations suivantes :

type

(Facultatif) Type de publish/subscribe messagerie utilisé par ce composant pour s'abonner aux messages. Sélectionnez parmi les options suivantes :

- PUB_SUB – Abonnez-vous aux messages locaux de publication/abonnement. Si vous choisissez cette option, le sujet ne peut pas contenir de caractères génériques MQTT. Pour plus d'informations sur la façon d'envoyer des messages à partir d'un composant personnalisé lorsque vous spécifiez cette option, consultez [Publier/souscrire des messages locaux](#).
- IOT_CORE— Abonnez-vous aux messages AWS IoT Core MQTT. Si vous choisissez cette option, le sujet peut contenir des caractères génériques MQTT. Pour plus d'informations sur la façon d'envoyer des messages à partir de composants personnalisés lorsque vous spécifiez cette option, consultez [Publier/souscrire AWS IoT Core des messages MQTT](#).

Par défaut : PUB_SUB

topic

(Facultatif) Rubrique à laquelle le composant s'abonne pour recevoir des messages. Si vous spécifiez `IotCore` pour `type`, vous pouvez utiliser des caractères génériques MQTT (+et#) dans cette rubrique.

Exemple Exemple : mise à jour de fusion de configuration (mode conteneur)

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "PROCFS_PATH": "/host_proc"
    }
  },
  "containerMode": "GreengrassContainer"
}
```

Exemple Exemple : mise à jour de fusion de configuration (pas de mode conteneur)

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "PROCFS_PATH": "/proc"
    }
  },
  "containerMode": "NoContainer"
}
```

Données d'entrée

Ce composant n'accepte pas les messages en tant que données d'entrée.

Données de sortie

Ce composant publie des métriques de sécurité dans la rubrique réservée suivante pour AWS IoT Device Defender. Ce composant est `coreDeviceName` remplacé par le nom du périphérique principal lorsqu'il publie les métriques.

Thème (AWS IoT Core MQTT) : `$aws/things/coreDeviceName/defender/metrics/json`

Exemple Exemple de sortie

```
{
  "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
  }
}
```

Pour plus d'informations sur les métriques rapportées par ce composant, consultez la [spécification du document relatif aux métriques des appareils](#) dans le Guide du AWS IoT Core développeur.

Fichier journal local

Ce composant utilise le fichier journal suivant.

Linux

```
/greengrass/v2/logs/aws.greengrass.DeviceDefender.log
```

Windows

```
C:\greengrass\v2\logs\aws.greengrass.DeviceDefender.log
```

Pour consulter les journaux de ce composant

- Exécutez la commande suivante sur le périphérique principal pour afficher le fichier journal de ce composant en temps réel. Remplacez */greengrass/v2* ou *C:\greengrass\v2* par le chemin d'accès au dossier AWS IoT Greengrass racine.

Linux

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

Windows (PowerShell)

```
Get-Content C:\greengrass\v2\logs\aws.greengrass.DeviceDefender.log -Tail 10 -  
Wait
```

Licences

Ce composant 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 composant.

v3.x

Version	Modifications
3.1.1	Corrections de bugs et améliorations - Ajoute des tentatives de connexion client lorsque la connexion ne parvient pas à se rétablir après une panne réseau. - Ajoute une nouvelle tentative configurable pour publier des métriques.
3.1.0	Corrections de bugs et améliorations Ajoute la prise en charge des configurations de proxy réseau HTTPS. Pour plus d'informations, consultez Connexion au port 443 ou via un proxy réseau et Permettre au périphérique principal de faire confiance à un proxy HTTPS.
3.0.1	Résout un problème lié à la façon dont le composant calcule les valeurs delta pour les métriques.

Version	Modifications
3.0.0	 Warning Cette version n'est plus disponible. Les améliorations apportées à cette version sont disponibles dans les versions ultérieures de ce composant. Première version.

v2.x

Version	Modifications
2,0,17	Version mise à jour pour la version 2.14.0 de Greengrass Nucleus.
2,0,16	Version mise à jour pour la version 2.13.0 de Greengrass Nucleus.
2.0.11	Version mise à jour pour la version 2.11.0 de Greengrass Nucleus.
2.0.10	Version mise à jour pour la version 2.7.0 de Greengrass Nucleus.
2.0.9	Version mise à jour pour la version 2.6.0 de Greengrass Nucleus.
2.0.8	Version mise à jour pour la version 2.5.0 de Greengrass Nucleus.
2.0.7	Version mise à jour pour la version 2.4.0 de Greengrass Nucleus.
2.0.6	Version mise à jour pour la version 2.3.0 de Greengrass Nucleus.
2.0.5	Version mise à jour pour la version 2.2.0 de Greengrass Nucleus.
2.0.4	Version mise à jour pour la version 2.1.0 de Greengrass Nucleus.
2.0.3	Première version.

spouleur à disque

Le composant spouleur de disque (`aws.greengrass.DiskSpooler`) offre une option de stockage permanent pour les messages envoyés depuis les appareils principaux de Greengrass vers. AWS IoT Core Ce composant stockera ces messages sortants sur le disque.

Rubriques

- [Versions](#)
- [Type](#)
- [Système d'exploitation](#)
- [Prérequis](#)
- [Dépendances](#)
- [Utilisation](#)
- [Fichier journal local](#)
- [Journal des modifications](#)

Versions

Les versions de ce composant sont les suivantes :

- 1,0 x

Type

Ce composant est un composant de plugin (`aws.greengrass.plugin`). Le [noyau Greengrass](#) exécute ce composant dans la même machine virtuelle Java (JVM) que le noyau. Le noyau redémarre lorsque vous modifiez la version de ce composant sur le périphérique principal.

Ce composant utilise le même fichier journal que le noyau Greengrass. Pour de plus amples informations, veuillez consulter [AWS IoT Greengrass Journaux de surveillance](#).

Pour de plus amples informations, veuillez consulter [Types de composants](#).

Système d'exploitation

Ce composant peut être installé sur les appareils principaux qui exécutent les systèmes d'exploitation suivants :

- Linux
- Windows

Prérequis

Ce composant répond aux exigences suivantes :

- `storageType` doit être paramétré `Disk` pour utiliser ce composant. Vous pouvez le définir dans la configuration du [noyau de Greengrass](#).
- `maxSizeInBytes` ne doit pas être configuré pour être supérieur à l'espace disponible sur l'appareil. Vous pouvez le définir dans la configuration du [noyau de Greengrass](#).
- Le composant spouleur de disque est compatible pour s'exécuter dans un VPC.

Dépendances

Lorsque vous déployez un composant, il déploie AWS IoT Greengrass également des versions compatibles de ses dépendances. Cela signifie que vous devez satisfaire aux exigences relatives au composant et à toutes ses dépendances pour réussir le déploiement du composant. Cette section répertorie les dépendances des [versions publiées](#) de ce composant et les contraintes de version sémantiques qui définissent les versions des composants pour chaque dépendance. Vous pouvez également consulter les dépendances de chaque version du composant dans la [AWS IoT Greengrass console](#). Sur la page de détails du composant, recherchez la liste des dépendances.

1.0.7

Le tableau suivant répertorie les dépendances pour la version 1.0.7 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	<code>>=2,11,0 <2,17,0</code>	Stricte

1.0.

Le tableau suivant répertorie les dépendances pour la version 1.0.6 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,11,0 <2,16,0	Stricte

1.0.5

Le tableau suivant répertorie les dépendances pour la version 1.0.5 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,11,0 <2,15,0	Stricte

1.0.4

Le tableau suivant répertorie les dépendances pour la version 1.0.4 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,11,0 <2,14,0	Stricte

1.0.1 – 1.0.3

Le tableau suivant répertorie les dépendances pour les versions 1.0.1 à 1.0.3 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,11,0 <2,13,0	Stricte

1.0.0

Le tableau suivant répertorie les dépendances pour la version 1.0.0 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,11,0 <2,12,0	Stricte

Pour plus d'informations sur les dépendances des composants, consultez la [référence de la recette des composants](#).

Utilisation

Pour utiliser le composant spouleur de disque, celui-ci `aws.greengrass.DiskSpooler` doit être déployé.

Pour configurer et utiliser ce composant, vous devez définir la valeur `pluginName` `suraws.greengrass.DiskSpooler`.

Fichier journal local

Ce composant utilise le même fichier journal que le composant [Greengrass nucleus](#).

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

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

Pour consulter les journaux de ce composant

- Exécutez la commande suivante sur le périphérique principal pour afficher le fichier journal de ce composant en temps réel. Remplacez `/greengrass/v2` ou `C:\greengrass\v2` par le chemin d'accès au dossier AWS IoT Greengrass racine.

Linux

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

Windows (PowerShell)

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

Journal des modifications

Le tableau suivant décrit les modifications apportées à chaque version du composant.

Version	Modifications
1.0.7	Version mise à jour pour la version 2.16.0 de Greengrass Nucleus.
1.0.6	Version mise à jour pour la version 2.15.0 de Greengrass Nucleus.
1.0.5	Version mise à jour pour la version 2.14.0 de Greengrass Nucleus.
1.0.4	Corrections de bogues et améliorations Corrections de bugs généraux.
1.0.3	Corrections de bogues et améliorations Améliore les performances en réutilisant les connexions aux bases de données.
1.0.2	Corrections de bogues et améliorations Résout un problème en raison duquel le champ de format de message MQTT n'est pas conservé dans certains cas.
1.0.1	Version mise à jour pour la version 2.12.0 de Greengrass Nucleus.
1.0.0	Première version.

Gestionnaire d'applications Docker

Le composant du gestionnaire d'applications Docker

(`aws.greengrass.DockerApplicationManager`) permet de télécharger des images Docker AWS IoT Greengrass à partir de registres d'images publics et de registres privés hébergés sur Amazon Elastic Container Registry (Amazon ECR). Il permet également AWS IoT Greengrass de gérer automatiquement les informations d'identification pour télécharger en toute sécurité des images à partir de référentiels privés sur Amazon ECR.

Lorsque vous développez un composant personnalisé qui exécute un conteneur Docker, incluez le gestionnaire d'applications Docker en tant que dépendance pour télécharger les images Docker spécifiées comme artefacts dans votre composant. Pour de plus amples informations, veuillez consulter [Exécuter un conteneur Docker](#).

Rubriques

- [Versions](#)
- [Type](#)
- [Système d'exploitation](#)
- [Exigences](#)
- [Dépendances](#)
- [Configuration](#)
- [Fichier journal local](#)
- [Journal des modifications](#)
- [Consultez aussi](#)

Versions

Les versions de ce composant sont les suivantes :

- 2.0.x

Type

Ce composant est un composant générique (`aws.greengrass.generic`). Le [noyau Greengrass](#) exécute les scripts de cycle de vie du composant.

Pour de plus amples informations, veuillez consulter [Types de composants](#).

Système d'exploitation

Ce composant peut être installé sur les appareils principaux qui exécutent les systèmes d'exploitation suivants :

- Linux
- Windows

Exigences

Ce composant répond aux exigences suivantes :

- [Docker Engine](#) 1.9.1 ou version ultérieure installé sur le périphérique principal de Greengrass. La version 20.10 est la dernière version vérifiée pour fonctionner avec le logiciel AWS IoT Greengrass Core. Vous devez installer Docker directement sur le périphérique principal avant de déployer des composants qui exécutent des conteneurs Docker.
- Le daemon Docker a démarré et s'est exécuté sur le périphérique principal avant que vous ne déployiez ce composant.
- Images Docker stockées dans l'une des sources d'images prises en charge suivantes :
 - Référentiels d'images publics et privés dans Amazon Elastic Container Registry (Amazon ECR)
 - Référentiel Docker Hub public
 - Registre public de confiance Docker
- Images Docker incluses sous forme d'artefacts dans vos composants de conteneur Docker personnalisés. Utilisez les formats d'URI suivants pour spécifier vos images Docker :
 - Image Amazon ECR privée : `docker:account-id.dkr.ecr.region.amazonaws.com/repository/image[:tag|@digest]`
 - Image Amazon ECR publique : `docker:public.ecr.aws/repository/image[:tag|@digest]`
 - Image publique du Docker Hub : `docker:name[:tag|@digest]`

Pour de plus amples informations, veuillez consulter [Exécuter un conteneur Docker](#).

Note

Si vous ne spécifiez pas de balise d'image ou de résumé d'image dans l'URI d'artefact d'une image, le gestionnaire d'applications Docker extrait la dernière version disponible de cette image lorsque vous déployez votre composant de conteneur Docker personnalisé. Pour garantir que tous vos appareils principaux exécutent la même version d'une image, nous vous recommandons d'inclure la balise d'image ou le résumé de l'image dans l'URI de l'artefact.

- L'utilisateur du système qui exécute un composant de conteneur Docker doit disposer des autorisations root ou administrateur, ou vous devez configurer Docker pour l'exécuter en tant qu'utilisateur non root ou non administrateur.

- Sur les appareils Linux, vous pouvez ajouter un utilisateur au `docker` groupe sans lequel vous pouvez appeler `docker` des commandes `sudo`.
- Sur les appareils Windows, vous pouvez ajouter un utilisateur au `docker-users` groupe pour appeler des `docker` commandes sans privilèges d'administrateur.

Linux or Unix

Pour ajouter `ggc_user` au `docker` groupe l'utilisateur non root que vous utilisez pour exécuter les composants du conteneur Docker, exécutez la commande suivante.

```
sudo usermod -aG docker ggc_user
```

Pour plus d'informations, consultez [Gérer Docker en tant qu'utilisateur non root](#).

Windows Command Prompt (CMD)

Pour ajouter `ggc_user` au `docker-users` groupe l'utilisateur que vous utilisez pour exécuter les composants du conteneur Docker, exécutez la commande suivante en tant qu'administrateur.

```
net localgroup docker-users ggc_user /add
```

Windows PowerShell

Pour ajouter `ggc_user` au `docker-users` groupe l'utilisateur que vous utilisez pour exécuter les composants du conteneur Docker, exécutez la commande suivante en tant qu'administrateur.

```
Add-LocalGroupMember -Group docker-users -Member ggc_user
```

- Si vous [configurez le logiciel AWS IoT Greengrass Core pour utiliser un proxy réseau](#), vous devez [configurer Docker pour qu'il utilise le même serveur proxy](#).
- Si vos images Docker sont stockées dans un registre privé Amazon ECR, vous devez inclure le composant du service d'échange de jetons en tant que dépendance dans le composant conteneur Docker. En outre, le [rôle d'appareil Greengrass](#) doit autoriser les `ecr:GetDownloadUrlForLayer` actions `ecr:GetAuthorizationToken` `ecr:BatchGetImage`, et, comme indiqué dans l'exemple de politique IAM suivant.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "ecr:GetAuthorizationToken",
        "ecr:BatchGetImage",
        "ecr:GetDownloadUrlForLayer"
      ],
      "Resource": [
        "*"
      ],
      "Effect": "Allow"
    }
  ]
}
```

- Le composant du gestionnaire d'applications docker est compatible avec l'exécution dans un VPC. Pour déployer ce composant dans un VPC, les éléments suivants sont requis.
- Le composant du gestionnaire d'applications docker doit disposer d'une connectivité pour télécharger des images. Par exemple, si vous utilisez l'ECR, vous devez être connecté aux points de terminaison suivants.
 - *.dkr.ecr.*region*.amazonaws.com(point de terminaison VPC)
com.amazonaws.*region*.ecr.dkr
 - api.ecr.*region*.amazonaws.com(point de terminaison VPC)
com.amazonaws.*region*.ecr.api

Points de terminaison et ports

Ce composant doit être capable d'effectuer des demandes sortantes vers les points de terminaison et les ports suivants, en plus des points de terminaison et des ports requis pour le fonctionnement de base. Pour de plus amples informations, veuillez consulter [Autoriser le trafic des appareils via un proxy ou un pare-feu](#).

Endpoint	Port	Obligatoire	Description
<code>ecr.<i>region</i>.amazonaws.com</code>	443	Non	Obligatoire si vous téléchargez des images Docker depuis Amazon ECR.
<code>hub.docker.com</code> <code>registry.hub.docker.com/v1</code>	443	Non	Obligatoire si vous téléchargez des images Docker depuis Docker Hub.

Dépendances

Lorsque vous déployez un composant, il déploie AWS IoT Greengrass également des versions compatibles de ses dépendances. Cela signifie que vous devez satisfaire aux exigences relatives au composant et à toutes ses dépendances pour réussir le déploiement du composant. Cette section répertorie les dépendances des [versions publiées](#) de ce composant et les contraintes de version sémantiques qui définissent les versions des composants pour chaque dépendance. Vous pouvez également consulter les dépendances de chaque version du composant dans la [AWS IoT Greengrass console](#). Sur la page de détails du composant, recherchez la liste des dépendances.

2.0.15

Le tableau suivant répertorie les dépendances pour la version 2.0.15 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,1,0 <2,17,0	Flexible

2.0.14

Le tableau suivant répertorie les dépendances pour la version 2.0.14 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,10 <2,16,0	Flexible

2.0.13

Le tableau suivant répertorie les dépendances pour la version 2.0.13 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,10 <2,15,0	Flexible

2.0.12

Le tableau suivant répertorie les dépendances pour la version 2.0.12 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,10 <2,14,0	Flexible

2.0.11

Le tableau suivant répertorie les dépendances pour la version 2.0.11 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,1,0 <2,13,0	Flexible

2.0.10

Le tableau suivant répertorie les dépendances pour la version 2.0.10 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,1,0 <2,12,0	Flexible

2.0.9

Le tableau suivant répertorie les dépendances pour la version 2.0.9 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,1,0 <2,11,0	Flexible

2.0.8

Le tableau suivant répertorie les dépendances pour la version 2.0.8 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,10 <2,1,0	Flexible

2.0.7

Le tableau suivant répertorie les dépendances pour la version 2.0.7 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,10 <2,9,0	Flexible

2.0.6

Le tableau suivant répertorie les dépendances pour la version 2.0.6 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,10 <2,8,0	Flexible

2.0.5

Le tableau suivant répertorie les dépendances pour la version 2.0.5 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,10 <2,7,0	Flexible

2.0.4

Le tableau suivant répertorie les dépendances pour la version 2.0.4 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,10 <2,6,0	Flexible

2.0.3

Le tableau suivant répertorie les dépendances pour la version 2.0.3 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,10 <2,5,0	Flexible

2.0.2

Le tableau suivant répertorie les dépendances pour la version 2.0.2 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,10 <2,4,0	Flexible

2.0.1

Le tableau suivant répertorie les dépendances pour la version 2.0.1 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,10 <2,3,0	Flexible

2.0.0

Le tableau suivant répertorie les dépendances pour la version 2.0.0 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,10 <2,2,0	Flexible

Pour plus d'informations sur les dépendances des composants, consultez la [référence de la recette des composants](#).

Configuration

Ce composant ne possède aucun paramètre de configuration.

Fichier journal local

Ce composant utilise le même fichier journal que le composant [Greengrass nucleus](#).

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

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

Pour consulter les journaux de ce composant

- Exécutez la commande suivante sur le périphérique principal pour afficher le fichier journal de ce composant en temps réel. Remplacez `/greengrass/v2` ou `C:\greengrass\v2` par le chemin d'accès au dossier AWS IoT Greengrass racine.

Linux

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

Windows (PowerShell)

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

Journal des modifications

Le tableau suivant décrit les modifications apportées à chaque version du composant.

Version	Modifications
2,0,15	Version mise à jour pour la version 2.16.0 de Greengrass Nucleus.
2,0,14	Version mise à jour pour la version 2.15.0 de Greengrass Nucleus.
2.0.13	Version mise à jour pour la version 2.14.0 de Greengrass Nucleus.
2,0,12	Version mise à jour pour la version 2.13.0 de Greengrass Nucleus.
2.0.11	Version mise à jour pour la version 2.12.0 de Greengrass Nucleus.
2.0.10	Version mise à jour pour la version 2.11.0 de Greengrass Nucleus.
2.0.9	Version mise à jour pour la version 2.10.0 de Greengrass Nucleus.
2.0.8	Version mise à jour pour la version 2.9.0 de Greengrass Nucleus.
2.0.7	Version mise à jour pour la version 2.8.0 de Greengrass Nucleus.
2.0.6	Version mise à jour pour la version 2.7.0 de Greengrass Nucleus.

Version	Modifications
2.0.5	Version mise à jour pour la version 2.6.0 de Greengrass Nucleus.
2.0.4	Version mise à jour pour la version 2.5.0 de Greengrass Nucleus.
2.0.3	Version mise à jour pour la version 2.4.0 de Greengrass Nucleus.
2.0.2	Version mise à jour pour la version 2.3.0 de Greengrass Nucleus.
2.0.1	Version mise à jour pour la version 2.2.0 de Greengrass Nucleus.
2.0.0	Première version.

Consultez aussi

- [Exécuter un conteneur Docker](#)

Connecteur Edge pour Kinesis Video Streams

Le connecteur Edge pour le composant Kinesis Video Streams `aws.iot.EdgeConnectorForKVS` () lit les flux vidéo des caméras locales et publie les flux sur Kinesis Video Streams. Vous pouvez configurer ce composant pour lire les flux vidéo des caméras IP (Internet Protocol) à l'aide du protocole RTSP (Real Time Streaming Protocol). Vous pouvez ensuite configurer des tableaux de bord [sur Amazon Managed Grafana ou](#) sur des serveurs Grafana locaux pour surveiller et interagir avec les flux vidéo.

Vous pouvez intégrer ce composant AWS IoT TwinMaker pour afficher et contrôler les flux vidéo dans les tableaux de bord Grafana. AWS IoT TwinMaker est un AWS service qui vous permet de créer des jumeaux numériques opérationnels de systèmes physiques. Vous pouvez les utiliser AWS IoT TwinMaker pour visualiser les données provenant de capteurs, de caméras et d'applications d'entreprise afin de suivre vos usines physiques, vos bâtiments ou vos installations industrielles. Vous pouvez également utiliser ces données pour surveiller les opérations, diagnostiquer les erreurs et réparer les erreurs. Pour plus d'informations, voir [Qu'est-ce que c'est AWS IoT TwinMaker ?](#) dans le guide de AWS IoT TwinMaker l'utilisateur.

Ce composant stocke sa configuration dans AWS IoT SiteWise un AWS service qui modélise et stocke les données industrielles. Dans AWS IoT SiteWise, les actifs représentent des objets tels

que des appareils, des équipements ou des groupes d'autres objets. Pour configurer et utiliser ce composant, vous devez créer une AWS IoT SiteWise ressource pour chaque périphérique principal Greengrass et pour chaque caméra IP connectée à chaque périphérique principal. Chaque ressource possède des propriétés que vous configurez pour contrôler les fonctionnalités, telles que la diffusion en direct, le téléchargement à la demande et la mise en cache locale. Pour spécifier l'URL de chaque caméra, vous devez créer un secret AWS Secrets Manager contenant l'URL de la caméra. Si la caméra nécessite une authentification, vous devez également spécifier un nom d'utilisateur et un mot de passe dans l'URL. Ensuite, vous spécifiez ce secret dans une propriété d'actif pour la caméra IP.

Ce composant télécharge le flux vidéo de chaque caméra vers un flux vidéo Kinesis. Vous spécifiez le nom du flux vidéo Kinesis de destination dans la configuration des AWS IoT SiteWise ressources pour chaque caméra. Si le flux vidéo Kinesis n'existe pas, ce composant le crée pour vous.

AWS IoT TwinMaker fournit un script que vous pouvez exécuter pour créer ces AWS IoT SiteWise actifs et les secrets de Secrets Manager. Pour plus d'informations sur la création de ces ressources et sur l'installation, la configuration et l'utilisation de ce composant, consultez la section [Intégration AWS IoT TwinMaker vidéo](#) dans le guide de AWS IoT TwinMaker l'utilisateur.

Note

Le connecteur Edge pour le composant Kinesis Video Streams est uniquement disponible dans les Régions AWS versions suivantes :

- USA Est (Virginie du Nord)
- USA Ouest (Oregon)
- Europe (Francfort)
- Europe (Irlande)
- Asie-Pacifique (Singapour)
- Asie-Pacifique (Tokyo)
- Asie-Pacifique (Séoul)
- Asie-Pacifique (Sydney)
- Asie-Pacifique (Mumbai)
- Chine (Pékin)

Rubriques

- [Versions](#)
- [Type](#)
- [Système d'exploitation](#)
- [Exigences](#)
- [Dépendances](#)
- [Configuration](#)
- [Licences](#)
- [Usage](#)
- [Fichier journal local](#)
- [Journal des modifications](#)
- [Consultez aussi](#)

Versions

Les versions de ce composant sont les suivantes :

- 1,0 x

Type

Ce composant est un composant générique (`aws.greengrass.generic`). Le [noyau Greengrass](#) exécute les scripts de cycle de vie du composant.

Pour de plus amples informations, veuillez consulter [Types de composants](#).

Système d'exploitation

Ce composant ne peut être installé que sur les appareils principaux de Linux.

Exigences

Ce composant répond aux exigences suivantes :

- Vous ne pouvez déployer ce composant que sur des appareils à cœur unique, car la configuration du composant doit être unique pour chaque périphérique principal. Vous ne pouvez pas déployer ce composant sur des groupes d'appareils principaux.

- [GStreamer](#) 1.18.4 ou version ultérieure installée sur le périphérique principal. Pour plus d'informations, consultez la section [Installation GStreamer](#).

Sur un appareil doté de apt, vous pouvez exécuter les commandes suivantes pour effectuer l'installation GStreamer.

```
sudo apt install -y libgstreamer1.0-dev libgstreamer-plugins-base1.0-dev
gstreamer1.0-plugins-base-apps
sudo apt install -y gstreamer1.0-libav
sudo apt install -y gstreamer1.0-plugins-bad gstreamer1.0-plugins-good gstreamer1.0-
plugins-ugly gstreamer1.0-tools
```

- Un AWS IoT SiteWise atout pour chaque appareil principal. Cet AWS IoT SiteWise actif représente le dispositif principal. Pour plus d'informations sur la création de cette ressource, consultez la section [Intégration AWS IoT TwinMaker vidéo](#) dans le guide de AWS IoT TwinMaker l'utilisateur.
- Un AWS IoT SiteWise atout pour chaque caméra IP que vous connectez à chaque appareil principal. Ces AWS IoT SiteWise actifs représentent les caméras qui diffusent des vidéos sur chaque appareil principal. La ressource de chaque caméra doit être associée à la ressource du périphérique principal qui se connecte à la caméra. Les actifs de caméra possèdent des propriétés que vous pouvez configurer pour spécifier un flux vidéo Kinesis, un secret d'authentification et des paramètres de diffusion vidéo. Pour plus d'informations sur la création et la configuration des éléments de caméra, consultez la section [Intégration AWS IoT TwinMaker vidéo](#) dans le guide de AWS IoT TwinMaker l'utilisateur.
- Un AWS Secrets Manager secret pour chaque caméra IP. Ce secret doit définir une paire clé-valeur, où se trouve la clé et la valeur est l'URL de la caméra. `RTSPStreamUrl` Si la caméra nécessite une authentification, incluez le nom d'utilisateur et le mot de passe dans cette URL. Vous pouvez utiliser un script pour créer un secret lorsque vous créez les ressources requises par ce composant. Pour plus d'informations, consultez la section [Intégration AWS IoT TwinMaker vidéo](#) dans le guide de AWS IoT TwinMaker l'utilisateur.

Vous pouvez également utiliser la console et l'API Secrets Manager pour créer des secrets supplémentaires. Pour plus d'informations, voir [Création d'un secret](#) dans le guide de AWS Secrets Manager l'utilisateur.

- Le [rôle d'échange de jetons Greengrass](#) doit autoriser les actions suivantes AWS Secrets Manager AWS IoT SiteWise, ainsi que les actions Kinesis Video Streams, comme illustré dans l'exemple de politique IAM suivant.

 Note

Cet exemple de politique permet au périphérique d'obtenir la valeur des secrets nommés **IPCamera1Url** et **IPCamera2Url**. Lorsque vous configurez chaque caméra IP, vous spécifiez un secret contenant l'URL de cette caméra. Si la caméra nécessite une authentification, vous devez également spécifier un nom d'utilisateur et un mot de passe dans l'URL. Le rôle d'échange de jetons du périphérique principal doit permettre l'accès au secret pour que chaque caméra IP puisse se connecter.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "secretsmanager:GetSecretValue"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:secretsmanager:us-east-1:123456789012:secret:IPCamera1Url",
        "arn:aws:secretsmanager:us-east-1:123456789012:secret:IPCamera2Url"
      ]
    },
    {
      "Action": [
        "iotsitewise:BatchPutAssetPropertyValue",
        "iotsitewise:DescribeAsset",
        "iotsitewise:DescribeAssetModel",
        "iotsitewise:DescribeAssetProperty",
        "iotsitewise:GetAssetPropertyValue",
        "iotsitewise:ListAssetRelationships",
        "iotsitewise:ListAssets",
        "iotsitewise:ListAssociatedAssets",
        "kinesisvideo:CreateStream",
        "kinesisvideo:DescribeStream",
        "kinesisvideo:GetDataEndpoint",
        "kinesisvideo:PutMedia",
        "kinesisvideo:TagStream"
      ]
    }
  ]
}
```

```
    ],  
    "Effect": "Allow",  
    "Resource": [  
        "*"   
    ]  
  }  
]  
}
```

Note

Si vous utilisez une AWS Key Management Service clé gérée par le client pour chiffrer des secrets, le rôle de l'appareil doit également autoriser `kms:Decrypt`.

Points de terminaison et ports

Ce composant doit être capable d'effectuer des demandes sortantes vers les points de terminaison et les ports suivants, en plus des points de terminaison et des ports requis pour le fonctionnement de base. Pour de plus amples informations, veuillez consulter [Autoriser le trafic des appareils via un proxy ou un pare-feu](#).

Endpoint	Port	Obligatoire	Description
kinesisvideo. <i>region</i> .amazonaws.com	443	Oui	Téléchargez des données vers Kinesis Video Streams.
data.iotsitewise. <i>region</i> .amazonaws.com	443	Oui	Publiez les métadonnées du flux vidéo sur AWS IoT SiteWise.

Endpoint	Port	Obligatoire	Description
secretsmanager. <i>region</i> .amazonaws.com	443	Oui	Téléchargez les secrets de l'URL de la caméra sur l'appareil principal.

Dépendances

Lorsque vous déployez un composant, il déploie AWS IoT Greengrass également des versions compatibles de ses dépendances. Cela signifie que vous devez satisfaire aux exigences relatives au composant et à toutes ses dépendances pour réussir le déploiement du composant. Cette section répertorie les dépendances des [versions publiées](#) de ce composant et les contraintes de version sémantiques qui définissent les versions des composants pour chaque dépendance. Vous pouvez également consulter les dépendances de chaque version du composant dans la [AWS IoT Greengrass console](#). Sur la page de détails du composant, recherchez la liste des dépendances.

Le tableau suivant répertorie les dépendances pour les versions 1.0.0 à 1.0.5 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Service d'échange de jetons	>=2,0,3	Stricte
Gestionnaire de flux	>=2,0,9	Stricte

Pour plus d'informations sur les dépendances des composants, consultez la [référence de la recette des composants](#).

Configuration

Ce composant fournit les paramètres de configuration suivants que vous pouvez personnaliser lorsque vous déployez le composant.

SiteWiseAssetIdForHub

L'ID de la AWS IoT SiteWise ressource qui représente cet appareil principal. Pour plus d'informations sur la création de cette ressource et son utilisation pour interagir avec ce composant, consultez la section [Intégration AWS IoT TwinMaker vidéo](#) dans le guide de AWS IoT TwinMaker l'utilisateur.

Exemple Exemple : mise à jour de la fusion de configurations

```
{
  "SiteWiseAssetIdForHub": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
}
```

Licences

Ce composant inclut les logiciels/licences tiers suivants :

- [Planificateur de tâches Quartz//Apache](#) License 2.0
- [liaisons Java pour GStreamer 1.x/Licence publique](#) générale limitée GNU v3.0

Usage

Pour configurer et interagir avec ce composant, vous pouvez définir des propriétés sur les AWS IoT SiteWise ressources qui représentent le périphérique principal et les caméras IP auxquelles il se connecte. Vous pouvez également visualiser et interagir avec les flux vidéo dans les tableaux de bord Grafana via. AWS IoT TwinMaker Pour plus d'informations, consultez la section [Intégration AWS IoT TwinMaker vidéo](#) dans le guide de AWS IoT TwinMaker l'utilisateur.

Fichier journal local

Ce composant utilise le fichier journal suivant.

```
/greengrass/v2/logs/aws.iot.EdgeConnectorForKVS.log
```

Pour consulter les journaux de ce composant

- Exécutez la commande suivante sur le périphérique principal pour afficher le fichier journal de ce composant en temps réel. Remplacez */greengrass/v2* par le chemin d'accès au dossier AWS IoT Greengrass racine.

```
sudo tail -f /greengrass/v2/logs/aws.iot.EdgeConnectorForKVS.log
```

Journal des modifications

Le tableau suivant décrit les modifications apportées à chaque version du composant.

Version	Modifications
1.0.5	Correction et amélioration de bogues généraux
1.0.4	Corrections de bogues et améliorations • Résout un problème à l'origine de l'arrêt du téléchargement en direct.
1.0.3	Correction et amélioration de bogues généraux
1.0.1	Correction et amélioration de bogues généraux
1.0.0	Première version.

Consultez aussi

- [Qu'est-ce que c'est AWS IoT TwinMaker ?](#) dans le guide de AWS IoT TwinMaker l'utilisateur
- [AWS IoT TwinMaker intégration vidéo](#) dans le guide de AWS IoT TwinMaker l'utilisateur
- [Qu'est-ce que c'est AWS IoT SiteWise ?](#) dans le guide de AWS IoT SiteWise l'utilisateur
- [Mise à jour des valeurs d'attribut](#) dans le guide de AWS IoT SiteWise l'utilisateur
- [Qu'est-ce que AWS Secrets Manager ?](#) dans le Guide de l'utilisateur AWS Secrets Manager
- [Création et gestion de secrets](#) dans le guide de AWS Secrets Manager l'utilisateur

Greengrass CLI

Le composant Greengrass CLI (`aws.greengrass.Cli`) fournit une interface de ligne de commande locale que vous pouvez utiliser sur les appareils principaux pour développer et déboguer des composants localement. La CLI Greengrass vous permet de créer des déploiements locaux et de redémarrer des composants sur le périphérique principal, par exemple.

Vous pouvez installer ce composant lors de l'installation du logiciel AWS IoT Greengrass Core. Pour de plus amples informations, veuillez consulter [Tutoriel : Débuter avec AWS IoT Greengrass V2](#).

Important

Nous vous recommandons d'utiliser ce composant uniquement dans les environnements de développement, et non dans les environnements de production. Ce composant permet d'accéder à des informations et à des opérations dont vous n'avez généralement pas besoin dans un environnement de production. Respectez le principe du moindre privilège en déployant ce composant uniquement sur les appareils principaux là où vous en avez besoin.

Après avoir installé ce composant, exécutez la commande suivante pour consulter sa documentation d'aide. Lorsque ce composant est installé, il ajoute un lien symbolique `greengrass-cli` dans le `/greengrass/v2/bin` dossier. Vous pouvez exécuter la CLI Greengrass à partir de ce chemin ou l'ajouter à votre variable d'PATH d'environnement pour l'exécuter `greengrass-cli` sans son chemin absolu.

Linux or Unix

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

Windows

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

La commande suivante redémarre un composant nommé `com.example.HelloWorld`, par exemple.

Linux or Unix

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

Windows

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

Pour de plus amples informations, veuillez consulter [Interface de ligne de commande Greengrass](#).

Rubriques

- [Versions](#)
- [Type](#)
- [Système d'exploitation](#)
- [Exigences](#)
- [Dépendances](#)
- [Configuration](#)
- [Fichier journal local](#)
- [Journal des modifications](#)

Versions

Les versions de ce composant sont les suivantes :

- 2,16.x
- 2,15.x
- 2,14.x
- 2,13.x
- 2,12. x
- 2,11.x
- 2.10.x
- 2,9. x
- 2,8. x
- 2.7.x
- 2.6.x
- 2,5.x
- 2.4.x
- 2.3.x
- 2.2.x
- 2.1.x

- 2.0.x

Type

Ce composant est un composant de plugin (`aws.greengrass.plugin`). Le [noyau Greengrass](#) exécute ce composant dans la même machine virtuelle Java (JVM) que le noyau. Le noyau redémarre lorsque vous modifiez la version de ce composant sur le périphérique principal.

Ce composant utilise le même fichier journal que le noyau Greengrass. Pour de plus amples informations, veuillez consulter [AWS IoT Greengrass Journaux de surveillance](#).

Pour de plus amples informations, veuillez consulter [Types de composants](#).

Système d'exploitation

Ce composant peut être installé sur les appareils principaux qui exécutent les systèmes d'exploitation suivants :

- Linux
- Windows

Exigences

Ce composant répond aux exigences suivantes :

- Vous devez être autorisé à utiliser la CLI Greengrass pour interagir avec le logiciel AWS IoT Greengrass principal. Pour utiliser la CLI Greengrass, effectuez l'une des opérations suivantes :
 - Utilisez l'utilisateur du système qui exécute le logiciel AWS IoT Greengrass Core.
 - Utilisez un utilisateur doté d'autorisations root ou administratives. Sur les appareils principaux de Linux, vous pouvez l'utiliser `sudo` pour obtenir des autorisations root.
 - Utilisez un utilisateur système appartenant à un groupe que vous spécifiez dans les paramètres de `AuthorizedWindowsGroups` configuration `AuthorizedPosixGroups` ou lorsque vous déployez le composant. Pour plus d'informations, consultez la section Configuration des [composants de la CLI Greengrass](#).
- Le composant Greengrass CLI est compatible avec l'exécution dans un VPC.

Dépendances

Lorsque vous déployez un composant, il déploie AWS IoT Greengrass également des versions compatibles de ses dépendances. Cela signifie que vous devez satisfaire aux exigences relatives au composant et à toutes ses dépendances pour réussir le déploiement du composant. Cette section répertorie les dépendances des [versions publiées](#) de ce composant et les contraintes de version sémantiques qui définissent les versions des composants pour chaque dépendance. Vous pouvez également consulter les dépendances de chaque version du composant dans la [AWS IoT Greengrass console](#). Sur la page de détails du composant, recherchez la liste des dépendances.

2.16.0

Le tableau suivant répertorie les dépendances pour la version 2.16.0 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,12,0 <2,17,0	Flexible

2.15.1

Le tableau suivant répertorie les dépendances pour la version 2.15.1 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,12,0 <2,16,0	Flexible

2.15.0

Le tableau suivant répertorie les dépendances pour la version 2.15.0 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,12,0 <2,16,0	Flexible

2.14.0 – 2.14.3

Le tableau suivant répertorie les dépendances pour les versions 2.14.0 et 2.14.3 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,12,0 < 2,15,0$	Flexible

2.13.0

Le tableau suivant répertorie les dépendances pour la version 2.13.0 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,12,0 < 2,14,0$	Flexible

2.12.0 – 2.12.6

Le tableau suivant répertorie les dépendances pour les versions 2.12.0 à 2.12.6 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,12,0 < 2,13,0$	Flexible

2.11.0 – 2.11.3

Le tableau suivant répertorie les dépendances pour les versions 2.11.0 à 2.11.3 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,11,0 < 2,12,0$	Flexible

2.10.0 – 2.10.3

Le tableau suivant répertorie les dépendances pour les versions 2.10.0 à 2.10.3 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,5,0 <2,11,0	Flexible

2.9.0 – 2.9.6

Le tableau suivant répertorie les dépendances pour les versions 2.9.0 à 2.9.6 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,5,0 <2,1,0	Flexible

2.8.0 – 2.8.1

Le tableau suivant répertorie les dépendances pour les versions 2.8.0 et 2.8.1 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,5,0 <2,9,0	Flexible

2.7.0

Le tableau suivant répertorie les dépendances pour la version 2.7.0 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,5,0 <2,8,0	Flexible

2.6.0

Le tableau suivant répertorie les dépendances pour la version 2.6.0 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,5,0 <2,7,0	Flexible

2.5.0 – 2.5.6

Le tableau suivant répertorie les dépendances pour les versions 2.5.0 à 2.5.6 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,5,0 <2,6,0	Flexible

2.4.0

Le tableau suivant répertorie les dépendances pour la version 2.4.0 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,10 <2,5,0	Flexible

2.3.0

Le tableau suivant répertorie les dépendances pour la version 2.3.0 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,10 <2,4,0	Flexible

2.2.0

Le tableau suivant répertorie les dépendances pour la version 2.2.0 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,10 <2,3,0	Flexible

2.1.0

Le tableau suivant répertorie les dépendances pour la version 2.1.0 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,10 <2,2,0	Flexible

2.0.x

Le tableau suivant répertorie les dépendances pour la version 2.0.x de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,10	Flexible

Note

La version minimale compatible du noyau Greengrass correspond à la version patch du composant Greengrass CLI.

Pour plus d'informations sur les dépendances des composants, consultez la [référence de la recette des composants](#).

Configuration

Ce composant fournit les paramètres de configuration suivants que vous pouvez personnaliser lorsque vous déployez le composant.

2.5.x - 2.14.x

AuthorizedPosixGroups

(Facultatif) Chaîne contenant une liste de groupes de systèmes séparés par des virgules. Vous autorisez ces groupes de systèmes à utiliser la CLI Greengrass pour interagir avec le logiciel AWS IoT Greengrass principal. Vous pouvez spécifier des noms de groupes ou des groupes IDs. Par exemple, `group1,1002,group3` autorise trois groupes de systèmes (`group11002`, `etgroup3`) à utiliser la CLI Greengrass.

Si vous ne spécifiez aucun groupe à autoriser, vous pouvez utiliser la CLI Greengrass en tant qu'utilisateur root (sudo) ou en tant qu'utilisateur système qui exécute le logiciel AWS IoT Greengrass Core.

AuthorizedWindowsGroups

(Facultatif) Chaîne contenant une liste de groupes de systèmes séparés par des virgules. Vous autorisez ces groupes de systèmes à utiliser la CLI Greengrass pour interagir avec le logiciel AWS IoT Greengrass principal. Vous pouvez spécifier des noms de groupes ou des groupes IDs. Par exemple, `group1,1002,group3` autorise trois groupes de systèmes (`group11002`, `etgroup3`) à utiliser la CLI Greengrass.

Si vous ne spécifiez aucun groupe à autoriser, vous pouvez utiliser la CLI Greengrass en tant qu'administrateur ou en tant qu'utilisateur du système qui exécute le logiciel AWS IoT Greengrass principal.

Exemple Exemple : mise à jour de la fusion de configurations

L'exemple de configuration suivant indique d'autoriser trois groupes de systèmes POSIX (`group1,1002`, `etgroup3`) et deux groupes d'utilisateurs Windows (`Device Operators` et `QA Engineers`) à utiliser la CLI Greengrass.

```
{
  "AuthorizedPosixGroups": "group1,1002,group3",
  "AuthorizedWindowsGroups": "Device Operators,QA Engineers"
}
```

2.4.x - 2.0.x

AuthorizedPosixGroups

(Facultatif) Chaîne contenant une liste de groupes de systèmes séparés par des virgules. Vous autorisez ces groupes de systèmes à utiliser la CLI Greengrass pour interagir avec le logiciel AWS IoT Greengrass principal. Vous pouvez spécifier des noms de groupes ou des groupes IDs. Par exemple, `group1,1002,group3` autorise trois groupes de systèmes (`group11002`, `etgroup3`) à utiliser la CLI Greengrass.

Si vous ne spécifiez aucun groupe à autoriser, vous pouvez utiliser la CLI Greengrass en tant qu'utilisateur root (sudo) ou en tant qu'utilisateur système qui exécute le logiciel AWS IoT Greengrass Core.

Exemple Exemple : mise à jour de la fusion de configurations

L'exemple de configuration suivant indique d'autoriser trois groupes de systèmes (group1,1002, etgroup3) à utiliser la CLI Greengrass.

```
{  
  "AuthorizedPosixGroups": "group1,1002,group3"  
}
```

Fichier journal local

Ce composant utilise le même fichier journal que le composant [Greengrass nucleus](#).

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

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

Pour consulter les journaux de ce composant

- Exécutez la commande suivante sur le périphérique principal pour afficher le fichier journal de ce composant en temps réel. Remplacez */greengrass/v2* ou *C:\greengrass\v2* par le chemin d'accès au dossier AWS IoT Greengrass racine.

Linux

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

Windows (PowerShell)

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

Journal des modifications

Le tableau suivant décrit les modifications apportées à chaque version du composant.

Version	Modifications
2.16.1	Version mise à jour pour la version 2.16.1 de Greengrass Nucleus.
2.16.0	Version mise à jour pour la version 2.16.0 de Greengrass Nucleus.
2.15.1	Version mise à jour pour la version 2.15.1 de Greengrass Nucleus.
2.15.0	Corrections de bogues et améliorations • Ajoute un correctif afin que lorsque la configuration d'authentification change, le composant ne se réinstalle pas mais redémarre tout de même. • Correction et amélioration de bogues généraux
2.14.3	Version mise à jour pour la version 2.14.3 de Greengrass Nucleus.
2.14.2	Version mise à jour pour la version 2.14.2 de Greengrass Nucleus.
2.14.1	Version mise à jour pour la version 2.14.1 de Greengrass Nucleus.
2.14.0	 Warning Cette version n'est plus disponible. Les améliorations apportées à cette version sont disponibles dans les versions ultérieures de ce composant. Corrections de bogues et améliorations • Validez le paramètre de cible de déploiement dans la commande cli.
2.13.0	Version mise à jour pour la version 2.13.0 de Greengrass Nucleus.
2.12.6	Version mise à jour pour la version 2.12.6 de Greengrass Nucleus.
2.12.5	Version mise à jour pour la version 2.12.5 de Greengrass Nucleus.
2.12.4	Version mise à jour pour la version 2.12.4 de Greengrass Nucleus.

Version	Modifications
2.12.3	 Warning Cette version n'est plus disponible. Les améliorations apportées à cette version sont disponibles dans les versions ultérieures de ce composant. Version mise à jour pour la version 2.12.3 de Greengrass Nucleus.
2.12.2	Version mise à jour pour la version 2.12.2 de Greengrass Nucleus.
2.12.1	Version mise à jour pour la version 2.12.1 de Greengrass Nucleus.
2.12.0	Version mise à jour pour la version 2.12.0 de Greengrass Nucleus.
2.11.3	Version mise à jour pour la version 2.11.3 de Greengrass Nucleus.
2.11.2	Version mise à jour pour la version 2.11.2 de Greengrass Nucleus.
2.11.1	Version mise à jour pour la version 2.11.1 de Greengrass Nucleus.
2.11.0	Nouvelles fonctionnalités • Vous permet d'annuler un déploiement local. • Vous permet de configurer une politique de gestion des défaillances pour un déploiement local. • Améliore les rapports détaillés sur l'état du déploiement.
2.10.3	Version mise à jour pour la version 2.10.3 de Greengrass Nucleus.
2.10.2	Version mise à jour pour la version 2.10.2 de Greengrass Nucleus.
2.10.1	Version mise à jour pour la version 2.10.1 de Greengrass Nucleus.
2.10.0	Version mise à jour pour la version 2.10.0 de Greengrass Nucleus.
2,9,6	Version mise à jour pour la version 2.9.6 de Greengrass Nucleus.

Version	Modifications
2.9.5	Version mise à jour pour la version 2.9.5 de Greengrass Nucleus.
2.9.4	Version mise à jour pour la version 2.9.4 de Greengrass Nucleus.
2.9.3	Version mise à jour pour la version 2.9.3 de Greengrass Nucleus.
2.9.2	Version mise à jour pour la version 2.9.2 de Greengrass Nucleus.
2.9.1	Version mise à jour pour la version 2.9.1 de Greengrass Nucleus.
2.9.0	Version mise à jour pour la version 2.9.0 de Greengrass Nucleus.
2.8.1	Version mise à jour pour la version 2.8.1 de Greengrass Nucleus.
2.8.0	Version mise à jour pour la version 2.8.0 de Greengrass Nucleus.
2.7.0	Version mise à jour pour la version 2.7.0 de Greengrass Nucleus.
2.6.0	Nouvelles fonctionnalités - Ajoute la prise en charge de composants personnalisés pour appeler les opérations de communication interprocessus (IPC) utilisées par la Greengrass CLI. Vous pouvez utiliser ces opérations IPC pour gérer les déploiements locaux, afficher les détails des composants et générer un mot de passe que vous pouvez utiliser pour vous connecter à la console de débogage locale. Pour plus d'informations, voir IPC : Gérer les déploiements et les composants locaux. Corrections de bogues et améliorations - Corrections et améliorations mineures supplémentaires.
2.5.6	Version mise à jour pour la version 2.5.6 de Greengrass Nucleus.
2.5.5	Version mise à jour pour la version 2.5.5 de Greengrass Nucleus.
2.5.4	Version mise à jour pour la version 2.5.4 de Greengrass Nucleus.
2.5.3	Version mise à jour pour la version 2.5.3 de Greengrass Nucleus.
2.5.2	Version mise à jour pour la version 2.5.2 de Greengrass Nucleus.

Version	Modifications
2.5.1	Version mise à jour pour la version 2.5.1 de Greengrass Nucleus.
2.5.0	Nouvelles fonctionnalités • Ajoute la prise en charge des appareils principaux qui exécutent Windows. • Ajoute le nouveau paramètre <code>AuthorizedWindowsGroups</code> de configuration que vous pouvez spécifier pour autoriser les groupes de systèmes à utiliser la CLI Greengrass sur les appareils Windows. • Ajoute le <code>windowsUser</code> paramètre pour les déploiements locaux. Vous pouvez utiliser ce paramètre pour spécifier l'utilisateur à utiliser pour exécuter les composants sur un périphérique principal Windows.
2.4.0	Nouvelles fonctionnalités • Ajoute la prise en charge des limites de ressources du système. Lorsque vous créez un déploiement local, vous pouvez configurer la quantité maximale d'utilisation du processeur et de la RAM que les processus de chaque composant peuvent utiliser sur le périphérique principal. Pour plus d'informations, consultez la section Configuration des limites de ressources système pour les composants et la commande Deployment Create.
2.3.0	Version mise à jour pour la version 2.3.0 de Greengrass Nucleus.
2.2.0	Version mise à jour pour la version 2.2.0 de Greengrass Nucleus.
2.1.0	Version mise à jour pour la version 2.1.0 de Greengrass Nucleus.
2.0.5	Version mise à jour pour la version 2.0.5 de Greengrass Nucleus.
2.0.4	Version mise à jour pour la version 2.0.4 de Greengrass Nucleus.
2.0.3	Première version.

Détecteur IP

Le composant du détecteur IP (`aws.greengrass.clientdevices.IPDetector`) effectue les opérations suivantes :

- Surveille les informations de connectivité réseau de l'appareil central Greengrass. Ces informations incluent les points de terminaison réseau du périphérique principal et le port sur lequel fonctionne un courtier MQTT.
- Met à jour les informations de connectivité de l'appareil principal dans le service AWS IoT Greengrass cloud.

Les appareils clients peuvent utiliser Greengrass Cloud Discovery pour récupérer les informations de connectivité des principaux appareils associés. Les appareils clients peuvent ensuite essayer de se connecter à chaque périphérique principal jusqu'à ce qu'ils se connectent correctement.

Note

Les appareils clients sont des appareils IoT locaux qui se connectent à un appareil principal de Greengrass pour envoyer des messages MQTT et des données à traiter. Pour de plus amples informations, veuillez consulter [Interagissez avec les appareils IoT locaux](#).

Le composant du détecteur IP remplace les informations de connectivité existantes d'un appareil principal par les informations qu'il détecte. Comme ce composant supprime les informations existantes, vous pouvez soit utiliser le composant de détection IP, soit gérer manuellement les informations de connectivité.

Rubriques

- [Versions](#)
- [Type](#)
- [Système d'exploitation](#)
- [Prérequis](#)
- [Dépendances](#)
- [Configuration](#)
- [Fichier journal local](#)

- [Journal des modifications](#)

Versions

Les versions de ce composant sont les suivantes :

- 2.2.x
- 2,1x
- 2,0.x

Type

Ce composant est un composant de plugin (`aws.greengrass.plugin`). Le [noyau Greengrass](#) exécute ce composant dans la même machine virtuelle Java (JVM) que le noyau. Le noyau redémarre lorsque vous modifiez la version de ce composant sur le périphérique principal.

Ce composant utilise le même fichier journal que le noyau Greengrass. Pour de plus amples informations, veuillez consulter [AWS IoT Greengrass Journaux de surveillance](#).

Pour de plus amples informations, veuillez consulter [Types de composants](#).

Système d'exploitation

Ce composant peut être installé sur les appareils principaux qui exécutent les systèmes d'exploitation suivants :

- Linux
- Windows

Prérequis

Ce composant répond aux exigences suivantes :

- Le [rôle de service Greengrass](#) doit être associé à vos autorisations Compte AWS et autoriser les `autorisationsiot:GetThingShadow`. `iot:UpdateThingShadow`
- La AWS IoT politique de l'appareil principal doit autoriser `!greengrass:UpdateConnectivityInfo` autorisation. Pour plus d'informations, consultez [AWS](#)

[IoT politiques relatives aux opérations du plan de données](#) et [AWS IoT Politique minimale de prise en charge des appareils clients](#).

- Si vous configurez le composant broker MQTT du périphérique principal pour utiliser un port autre que le port par défaut 8883, vous devez utiliser le détecteur IP v2.1.0 ou version ultérieure. Configurez-le pour signaler le port sur lequel le broker opère.
- Si vous avez une configuration réseau complexe, le composant du détecteur IP risque de ne pas être en mesure d'identifier les points de terminaison auxquels les appareils clients peuvent se connecter au périphérique principal. Si le composant du détecteur IP ne peut pas gérer les points de terminaison, vous devez plutôt gérer manuellement les points de terminaison de l'appareil principal. Par exemple, si le périphérique principal se trouve derrière un routeur qui lui transmet le port du broker MQTT, vous devez spécifier l'adresse IP du routeur comme point de terminaison pour le périphérique principal. Pour de plus amples informations, veuillez consulter [Gérez les principaux points de terminaison des appareils](#).
- Le composant du détecteur IP est compatible pour fonctionner dans un VPC.

Dépendances

Lorsque vous déployez un composant, il déploie AWS IoT Greengrass également des versions compatibles de ses dépendances. Cela signifie que vous devez satisfaire aux exigences relatives au composant et à toutes ses dépendances pour réussir le déploiement du composant. Cette section répertorie les dépendances des [versions publiées](#) de ce composant et les contraintes de version sémantiques qui définissent les versions des composants pour chaque dépendance. Vous pouvez également consulter les dépendances de chaque version du composant dans la [AWS IoT Greengrass console](#). Sur la page de détails du composant, recherchez la liste des dépendances.

2.2.3

Le tableau suivant répertorie les dépendances pour la version 2.2.3 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,2,0 <2,17,0	Flexible

2.2.2

Le tableau suivant répertorie les dépendances pour la version 2.2.2 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,2,0 <2,16,0	Flexible

2.2.1

Le tableau suivant répertorie les dépendances pour la version 2.2.1 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,2,0 <2,15,0	Flexible

2.2.0

Le tableau suivant répertorie les dépendances pour la version 2.2.0 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,2,0 <2,14,0	Flexible

2.1.8 – 2.1.9

Le tableau suivant répertorie les dépendances pour les versions 2.1.8 et 2.1.9 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,2,0 <2,13,0	Flexible

2.1.7

Le tableau suivant répertorie les dépendances pour la version 2.1.7 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,2,0 <2,12,0	Flexible

2.1.6

Le tableau suivant répertorie les dépendances pour la version 2.1.6 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,2,0$ $< 2,11,0$	Flexible

2.1.5

Le tableau suivant répertorie les dépendances pour la version 2.1.5 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,2,0$ $< 2,1,0$	Flexible

2.1.4

Le tableau suivant répertorie les dépendances pour la version 2.1.4 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,2,0$ $< 2,9,0$	Flexible

2.1.3

Le tableau suivant répertorie les dépendances pour la version 2.1.3 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,2,0$ $< 2,8,0$	Flexible

2.1.2

Le tableau suivant répertorie les dépendances pour la version 2.1.2 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,2,0 <2,7,0	Flexible

2.1.1

Le tableau suivant répertorie les dépendances pour la version 2.1.1 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,2,0 <2,6,0	Flexible

2.1.0 and 2.0.2

Le tableau suivant répertorie les dépendances pour les versions 2.1.0 et 2.0.2 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,2,0 <2,5,0	Flexible

2.0.1

Le tableau suivant répertorie les dépendances pour la version 2.0.1 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,2,0 <2,4,0	Flexible

2.0.0

Le tableau suivant répertorie les dépendances pour la version 2.0.0 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,2,0 <2,3,0	Flexible

Pour plus d'informations sur les dépendances des composants, consultez la [référence de la recette des composants](#).

Configuration

Ce composant fournit les paramètres de configuration suivants que vous pouvez personnaliser lorsque vous déployez le composant.

2.2.x

`defaultPort`

(Facultatif) Le port du broker MQTT à signaler lorsque ce composant détecte des adresses IP. Vous devez spécifier ce paramètre si vous configurez le broker MQTT pour utiliser un port différent du port par défaut 8883.

Par défaut : 8883

`includeIPv4LoopbackAddr`

(Facultatif) Vous pouvez activer cette option pour détecter et signaler les adresses de IPv4 boucle. Il s'agit d'adresses IP, par exemple `localhost`, où un appareil peut communiquer avec lui-même. Utilisez cette option dans les environnements de test dans lesquels le périphérique principal et le périphérique client s'exécutent sur le même système.

Par défaut : `false`

`includeIPv4LinkLocalAddr`

(Facultatif) Vous pouvez activer cette option pour détecter et signaler les adresses IPv4 [locales liées aux liens](#). Utilisez cette option si le réseau du périphérique principal ne dispose pas du protocole DHCP (Dynamic Host Configuration Protocol) ou d'adresses IP attribuées de manière statique.

Par défaut : `false`

`includeIPv6LoopbackAddr`

(Facultatif) Vous pouvez activer cette option pour détecter et signaler les adresses de IPv6 boucle. Il s'agit d'adresses IP, par exemple `localhost`, où un appareil peut communiquer avec lui-même. Utilisez cette option dans les environnements de test dans lesquels le périphérique principal et le périphérique client s'exécutent sur le même système. Vous devez

`includeIPv4Addr` définir sur `false` et `includeIPv6Addr` sur `true` pour utiliser cette option.

Par défaut : `false`

`includeIPv6LinkLocalAddr`

(Facultatif) Vous pouvez activer cette option pour détecter et signaler les adresses IPv6 [locales liées aux liens](#). Utilisez cette option si le réseau du périphérique principal ne dispose pas du protocole DHCP (Dynamic Host Configuration Protocol) ou d'adresses IP attribuées de manière statique. Vous devez `includeIPv4Addr` définir sur `false` et `includeIPv6Addr` sur `true` pour utiliser cette option.

Par défaut : `false`

`includeIPv4Addr`

(Facultatif) La valeur par défaut est définie sur `true`. Vous pouvez activer cette option pour publier IPv4 les adresses présentes sur l'appareil principal.

Par défaut : `true`

`includeIPv6Addr`

(Facultatif) Vous pouvez activer cette option pour publier IPv6 les adresses présentes sur l'appareil principal. Réglez `false` sur `includeIPv4Addr` pour utiliser cette option.

Par défaut : `false`

2.1.x

`defaultPort`

(Facultatif) Le port du broker MQTT à signaler lorsque ce composant détecte des adresses IP. Vous devez spécifier ce paramètre si vous configurez le broker MQTT pour utiliser un port différent du port par défaut 8883.

Par défaut : 8883

`includeIPv4LoopbackAddr`

(Facultatif) Vous pouvez activer cette option pour détecter et signaler les adresses de IPv4 boucle. Il s'agit d'adresses IP, par exemple `localhost`, où un appareil peut communiquer

avec lui-même. Utilisez cette option dans les environnements de test dans lesquels le périphérique principal et le périphérique client s'exécutent sur le même système.

Par défaut : `false`

`includeIPv4LinkLocalAddr`

(Facultatif) Vous pouvez activer cette option pour détecter et signaler les adresses IPv4 [locales liées aux liens](#). Utilisez cette option si le réseau du périphérique principal ne dispose pas du protocole DHCP (Dynamic Host Configuration Protocol) ou d'adresses IP attribuées de manière statique.

Par défaut : `false`

2.0.x

`includeIPv4LoopbackAddr`

(Facultatif) Vous pouvez activer cette option pour détecter et signaler les adresses de IPv4 boucle. Il s'agit d'adresses IP, par exemple `localhost`, où un appareil peut communiquer avec lui-même. Utilisez cette option dans les environnements de test dans lesquels le périphérique principal et le périphérique client s'exécutent sur le même système.

Par défaut : `false`

`includeIPv4LinkLocalAddr`

(Facultatif) Vous pouvez activer cette option pour détecter et signaler les adresses IPv4 [locales liées aux liens](#). Utilisez cette option si le réseau du périphérique principal ne dispose pas du protocole DHCP (Dynamic Host Configuration Protocol) ou d'adresses IP attribuées de manière statique.

Par défaut : `false`

Fichier journal local

Ce composant utilise le même fichier journal que le composant [Greengrass nucleus](#).

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

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

Pour consulter les journaux de ce composant

- Exécutez la commande suivante sur le périphérique principal pour afficher le fichier journal de ce composant en temps réel. Remplacez `/greengrass/v2` ou `C:\greengrass\v2` par le chemin d'accès au dossier AWS IoT Greengrass racine.

Linux

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

Windows (PowerShell)

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

Journal des modifications

Le tableau suivant décrit les modifications apportées à chaque version du composant.

Version	Modifications
2.2.3	Version mise à jour pour la version 2.16.0 de Greengrass Nucleus.
2.2.2	Version mise à jour pour la version 2.15.0 de Greengrass Nucleus.
2.2.1	Version mise à jour pour la version 2.14.0 de Greengrass Nucleus.
2.2.0	Version mise à jour pour la version 2.13.0 de Greengrass Nucleus. Nouvelles fonctionnalités Ajoute la prise en charge de IPv6. Vous pouvez désormais l'utiliser IPv6 pour la messagerie locale.

Version	Modifications
2.1.9	Corrections de bogues et améliorations Ajuste l'étape d'acquisition de l'adresse IP pour envoyer uniquement les journaux au niveau du journal de débogage.
2.1.8	Version mise à jour pour la version 2.12.0 de Greengrass Nucleus.
2.1.7	Version mise à jour pour la version 2.11.0 de Greengrass Nucleus.
2.1.6	Version mise à jour pour la version 2.10.0 de Greengrass Nucleus.
2.1.5	Version mise à jour pour la version 2.9.0 de Greengrass Nucleus.
2.1.4	Version mise à jour pour la version 2.8.0 de Greengrass Nucleus.
2.1.3	Version mise à jour pour la version 2.7.0 de Greengrass Nucleus.
2.1.2	Corrections de bogues et améliorations - Améliore les messages d'erreur enregistrés par ce composant dans certains scénarios. - Version mise à jour pour la version 2.6.0 de Greengrass Nucleus.
2.1.1	Version mise à jour pour la version 2.5.0 de Greengrass Nucleus.
2.1.0	Améliorations - Ajoute le <code>defaultPort</code> paramètre, qui vous permet d'utiliser un port de broker MQTT autre que celui par défaut. - Mises à jour pour rendre les messages du journal plus clairs.
2.0.2	Version mise à jour pour la version 2.4.0 de Greengrass Nucleus.
2.0.1	Version mise à jour pour la version 2.3.0 de Greengrass Nucleus.
2.0.0	Première version.

Firehose

Le composant Firehose (`aws.greengrass.KinesisFirehose`) publie des données via les flux de diffusion Amazon Data Firehose vers des destinations telles qu'Amazon S3, Amazon Redshift et Amazon Service. OpenSearch Pour plus d'informations, consultez [Qu'est-ce qu'Amazon Data Firehose ?](#) dans le manuel Amazon Data Firehose Developer Guide.

Pour publier sur un flux de diffusion Kinesis avec ce composant, publiez un message dans un sujet auquel ce composant est abonné. Par défaut, ce composant s'abonne aux rubriques de `kinesisfirehose/message` [publication/d'abonnement kinesisfirehose/message/binary/# locales](#). Vous pouvez spécifier d'autres sujets, y compris les sujets AWS IoT Core MQTT, lorsque vous déployez ce composant.

Note

Ce composant fournit des fonctionnalités similaires à celles du connecteur Firehose de la version 1. AWS IoT Greengrass Pour plus d'informations, consultez la section [Connecteur Firehose](#) dans le Guide du développeur de la AWS IoT Greengrass V1.

Rubriques

- [Versions](#)
- [Type](#)
- [Système d'exploitation](#)
- [Exigences](#)
- [Dépendances](#)
- [Configuration](#)
- [Données d'entrée](#)
- [Données de sortie](#)
- [Fichier journal local](#)
- [Licences](#)
- [Journal des modifications](#)
- [Consultez aussi](#)

Versions

Les versions de ce composant sont les suivantes :

- 2.1.x
- 2.0.x

Type

Ce composant est un composant Lambda () `aws.greengrass.lambda`. [Le noyau Greengrass exécute la fonction Lambda de ce composant à l'aide du composant Lambda Launcher.](#)

Pour de plus amples informations, veuillez consulter [Types de composants](#).

Système d'exploitation

Ce composant ne peut être installé que sur les appareils principaux de Linux.

Exigences

Ce composant répond aux exigences suivantes :

- Votre appareil principal doit répondre aux exigences pour exécuter les fonctions Lambda. Si vous souhaitez que le périphérique principal exécute des fonctions Lambda conteneurisées, le périphérique doit répondre aux exigences requises. Pour de plus amples informations, veuillez consulter [Exigences relatives à la fonction Lambda](#).
- [Python](#) version 3.7 installé sur le périphérique principal et ajouté à la variable d'environnement PATH.
- Le [rôle d'appareil Greengrass](#) doit autoriser les `firehose:PutRecordBatch` actions `firehose:PutRecord` et, comme illustré dans l'exemple de politique IAM suivant.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "firehose:PutRecord",
        "firehose:PutRecordBatch"
      ]
    }
  ]
}
```

```

    ],
    "Effect": "Allow",
    "Resource": [
        "arn:aws:firehose:us-east-1:123456789012:deliverystream/stream-name"
    ]
}
]
}

```

Vous pouvez remplacer dynamiquement le flux de diffusion par défaut dans la charge utile du message d'entrée pour ce composant. Si votre application 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 recevoir les données de sortie de ce composant, vous devez fusionner la mise à jour de configuration suivante pour l'[ancien composant routeur d'abonnement](#) (`aws.greengrass.LegacySubscriptionRouter`) lorsque vous déployez ce composant. Cette configuration indique le sujet dans lequel ce composant publie les réponses.

Legacy subscription router v2.1.x

```

{
  "subscriptions": {
    "aws-greengrass-kinesisfirehose": {
      "id": "aws-greengrass-kinesisfirehose",
      "source": "component:aws.greengrass.KinesisFirehose",
      "subject": "kinesisfirehose/message/status",
      "target": "cloud"
    }
  }
}

```

Legacy subscription router v2.0.x

```

{
  "subscriptions": {
    "aws-greengrass-kinesisfirehose": {
      "id": "aws-greengrass-kinesisfirehose",
      "source": "arn:aws:lambda:region:aws:function:aws-greengrass-kinesisfirehose:version",
      "subject": "kinesisfirehose/message/status",

```

```

    "target": "cloud"
  }
}
}

```

- **region** Remplacez-le par Région AWS celui que vous utilisez.
- **version** Remplacez-le par la version de la fonction Lambda exécutée par ce composant. Pour trouver la version de la fonction Lambda, vous devez consulter la recette de la version de ce composant que vous souhaitez déployer. Ouvrez la page de détails de ce composant dans la [AWS IoT Greengrass console](#) et recherchez la paire clé-valeur de la fonction Lambda. Cette paire clé-valeur contient le nom et la version de la fonction Lambda.

Important

Vous devez mettre à jour la version de la fonction Lambda sur l'ancien routeur d'abonnement chaque fois que vous déployez ce composant. Cela garantit que vous utilisez la bonne version de la fonction Lambda pour la version du composant que vous déployez.

Pour de plus amples informations, veuillez consulter [Créer des déploiements](#).

- Le composant Firehose est compatible pour s'exécuter dans un VPC. Pour déployer ce composant dans un VPC, les éléments suivants sont requis.
 - Le composant Firehose doit disposer d'une connectivité `firehose.region.amazonaws.com` dont le point de terminaison VPC est de `com.amazonaws.region.kinesis-firehose`

Points de terminaison et ports

Ce composant doit être capable d'effectuer des demandes sortantes vers les points de terminaison et les ports suivants, en plus des points de terminaison et des ports requis pour le fonctionnement de base. Pour de plus amples informations, veuillez consulter [Autoriser le trafic des appareils via un proxy ou un pare-feu](#).

Endpoint	Port	Obligatoire	Description
<code>firehose. region.amazonaws.com</code>	443	Oui	Téléchargez les

Endpoint	Port	Obligatoire	Description
			données vers Firehose.

Dépendances

Lorsque vous déployez un composant, il déploie AWS IoT Greengrass également des versions compatibles de ses dépendances. Cela signifie que vous devez satisfaire aux exigences relatives au composant et à toutes ses dépendances pour réussir le déploiement du composant. Cette section répertorie les dépendances des [versions publiées](#) de ce composant et les contraintes de version sémantiques qui définissent les versions des composants pour chaque dépendance. Vous pouvez également consulter les dépendances de chaque version du composant dans la [AWS IoT Greengrass console](#). Sur la page de détails du composant, recherchez la liste des dépendances.

2.1.10

Le tableau suivant répertorie les dépendances pour la version 2.1.10 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0,0 <2,16,0	Stricte
Lanceur Lambda	^2,0.0	Stricte
Environnements d'exécution (runtimes) Lambda	^2,0.0	Flexible
Service d'échange de jetons	^2,0.0	Stricte

2.1.9

Le tableau suivant répertorie les dépendances pour la version 2.1.9 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,15.0	Stricte

Dépendance	Versions compatibles	Type de dépendance
Lanceur Lambda	^2,0.0	Stricte
Environnements d'exécution (runtimes) Lambda	^2,0.0	Flexible
Service d'échange de jetons	^2,0.0	Stricte

2.1.8

Le tableau suivant répertorie les dépendances pour la version 2.1.8 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,14.0	Stricte
Lanceur Lambda	^2,0.0	Stricte
Environnements d'exécution (runtimes) Lambda	^2,0.0	Flexible
Service d'échange de jetons	^2,0.0	Stricte

2.1.7

Le tableau suivant répertorie les dépendances pour la version 2.1.7 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0,0 <2,13,0	Stricte
Lanceur Lambda	^2,0.0	Stricte
Environnements d'exécution (runtimes) Lambda	^2,0.0	Flexible
Service d'échange de jetons	^2,0.0	Stricte

2.1.6

Le tableau suivant répertorie les dépendances pour la version 2.1.6 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,0,0 < 2,12,0$	Stricte
Lanceur Lambda	$\wedge 2,0,0$	Stricte
Environnements d'exécution (runtimes) Lambda	$\wedge 2,0,0$	Flexible
Service d'échange de jetons	$\wedge 2,0,0$	Stricte

2.1.5

Le tableau suivant répertorie les dépendances pour la version 2.1.5 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,0,0 < 2,11,0$	Stricte
Lanceur Lambda	$\wedge 2,0,0$	Stricte
Environnements d'exécution (runtimes) Lambda	$\wedge 2,0,0$	Flexible
Service d'échange de jetons	$\wedge 2,0,0$	Stricte

2.1.4

Le tableau suivant répertorie les dépendances pour la version 2.1.4 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,0,0 < 2,1,0$	Stricte

Dépendance	Versions compatibles	Type de dépendance
Lanceur Lambda	^2,0.0	Stricte
Environnements d'exécution (runtimes) Lambda	^2,0.0	Flexible
Service d'échange de jetons	^2,0.0	Stricte

2.1.3

Le tableau suivant répertorie les dépendances pour la version 2.1.3 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,9.0	Stricte
Lanceur Lambda	^2,0.0	Stricte
Environnements d'exécution (runtimes) Lambda	^2,0.0	Flexible
Service d'échange de jetons	^2,0.0	Stricte

2.1.2

Le tableau suivant répertorie les dépendances pour la version 2.1.2 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,8.0	Stricte
Lanceur Lambda	^2,0.0	Stricte
Environnements d'exécution (runtimes) Lambda	^2,0.0	Flexible
Service d'échange de jetons	^2,0.0	Stricte

2.1.1

Le tableau suivant répertorie les dépendances pour la version 2.1.1 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,0.0 < 2,7.0$	Stricte
Lanceur Lambda	$\wedge 2,0.0$	Stricte
Environnements d'exécution (runtimes) Lambda	$\wedge 2,0.0$	Flexible
Service d'échange de jetons	$\wedge 2,0.0$	Stricte

2.0.8 - 2.1.0

Le tableau suivant répertorie les dépendances pour les versions 2.0.8 et 2.1.0 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,0.0 < 2,6.0$	Stricte
Lanceur Lambda	$\wedge 2,0.0$	Stricte
Environnements d'exécution (runtimes) Lambda	$\wedge 2,0.0$	Flexible
Service d'échange de jetons	$\wedge 2,0.0$	Stricte

2.0.7

Le tableau suivant répertorie les dépendances pour la version 2.0.7 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,0.0 < 2,5.0$	Stricte

Dépendance	Versions compatibles	Type de dépendance
Lanceur Lambda	^2,0.0	Stricte
Environnements d'exécution (runtimes) Lambda	^2,0.0	Flexible
Service d'échange de jetons	^2,0.0	Stricte

2.0.6

Le tableau suivant répertorie les dépendances pour la version 2.0.6 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,4.0	Stricte
Lanceur Lambda	^2,0.0	Stricte
Environnements d'exécution (runtimes) Lambda	^2,0.0	Flexible
Service d'échange de jetons	^2,0.0	Stricte

2.0.5

Le tableau suivant répertorie les dépendances pour la version 2.0.5 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,3.0	Stricte
Lanceur Lambda	^2,0.0	Stricte
Environnements d'exécution (runtimes) Lambda	^2,0.0	Flexible
Service d'échange de jetons	^2,0.0	Stricte

2.0.4

Le tableau suivant répertorie les dépendances pour la version 2.0.4 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,0.0 < 2,2.0$	Stricte
Lanceur Lambda	$\wedge 2,0.0$	Stricte
Environnements d'exécution (runtimes) Lambda	$\wedge 2,0.0$	Flexible
Service d'échange de jetons	$\wedge 2,0.0$	Stricte

2.0.3

Le tableau suivant répertorie les dépendances pour la version 2.0.3 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,0,3 < 2,10$	Stricte
Lanceur Lambda	$\geq 10,0$	Stricte
Environnements d'exécution (runtimes) Lambda	$\geq 10,0$	Flexible
Service d'échange de jetons	$\geq 10,0$	Stricte

Pour plus d'informations sur les dépendances des composants, consultez la [référence de la recette des composants](#).

Configuration

Ce composant fournit les paramètres de configuration suivants que vous pouvez personnaliser lorsque vous déployez le composant.

Note

La configuration par défaut de ce composant inclut les paramètres de la fonction Lambda. Nous vous recommandons de modifier uniquement les paramètres suivants pour configurer ce composant sur vos appareils.

lambdaParams

Objet contenant les paramètres de la fonction Lambda de ce composant. Cet objet contient les informations suivantes :

EnvironmentVariables

Objet contenant les paramètres de la fonction Lambda. Cet objet contient les informations suivantes :

DEFAULT_DELIVERY_STREAM_ARN

L'ARN du flux de diffusion Firehose par défaut où le composant envoie des données. Vous pouvez remplacer le flux de destination par la `delivery_stream_arn` propriété dans la charge utile du message d'entrée.

Note

Le rôle principal de l'appareil doit autoriser les actions requises sur tous les flux de diffusion cibles. Pour de plus amples informations, veuillez consulter [Exigences](#).

PUBLISH_INTERVAL

(Facultatif) Le nombre maximal de secondes à attendre avant que le composant publie des données par lots dans Firehose. Pour configurer le composant afin qu'il publie les métriques au fur et à mesure qu'il les reçoit, c'est-à-dire sans traitement par lots, spécifiez 0.

Cette valeur peut être d'au plus 900 secondes.

Par défaut : 10 secondes

DELIVERY_STREAM_QUEUE_SIZE

(Facultatif) Nombre maximal d'enregistrements à conserver en mémoire avant que le composant rejette de nouveaux enregistrements pour le même flux de diffusion.

Cette valeur doit être d'au moins 2 000 enregistrements.

Par défaut : 5 000 enregistrements

containerMode

(Facultatif) Mode de conteneurisation de ce composant. Sélectionnez l'une des options suivantes :

- `NoContainer`— Le composant ne s'exécute pas dans un environnement d'exécution isolé.
- `GreengrassContainer`— Le composant s'exécute dans un environnement d'exécution isolé à l'intérieur du AWS IoT Greengrass conteneur.

Valeur par défaut : `GreengrassContainer`

containerParams

(Facultatif) Objet contenant les paramètres du conteneur pour ce composant. Le composant utilise ces paramètres si vous le spécifiez `GreengrassContainer` pour `containerMode`.

Cet objet contient les informations suivantes :

memorySize

(Facultatif) La quantité de mémoire (en kilo-octets) à allouer au composant.

La valeur par défaut est de 64 Mo (65 535 Ko).

pubsubTopics

(Facultatif) Objet contenant les rubriques auxquelles le composant s'abonne pour recevoir des messages. Vous pouvez spécifier chaque sujet et indiquer si le composant est abonné à des sujets MQTT provenant de sujets locaux AWS IoT Core ou locaux publish/subscribe .

Cet objet contient les informations suivantes :

0— Il s'agit d'un index de tableau sous forme de chaîne.

Un objet qui contient les informations suivantes :

type

(Facultatif) Type de publish/subscribe messagerie utilisé par ce composant pour s'abonner aux messages. Sélectionnez l'une des options suivantes :

- **PUB_SUB** – Abonnez-vous aux messages locaux de publication/abonnement. Si vous choisissez cette option, le sujet ne peut pas contenir de caractères génériques MQTT. Pour plus d'informations sur la façon d'envoyer des messages à partir d'un composant personnalisé lorsque vous spécifiez cette option, consultez [Publier/souscrire des messages locaux](#).
- **IOT_CORE**— Abonnez-vous aux messages AWS IoT Core MQTT. Si vous choisissez cette option, le sujet peut contenir des caractères génériques MQTT. Pour plus d'informations sur la façon d'envoyer des messages à partir de composants personnalisés lorsque vous spécifiez cette option, consultez [Publier/souscrire AWS IoT Core des messages MQTT](#).

Valeur par défaut : **PUB_SUB**

topic

(Facultatif) Rubrique à laquelle le composant s'abonne pour recevoir des messages. Si vous spécifiez `IotCore` pour `type`, vous pouvez utiliser des caractères génériques MQTT (+et#) dans cette rubrique.

Exemple Exemple : mise à jour de fusion de configuration (mode conteneur)

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "DEFAULT_DELIVERY_STREAM_ARN": "arn:aws:firehose:us-
west-2:123456789012:deliverystream/mystream"
    }
  },
  "containerMode": "GreengrassContainer"
}
```

Exemple Exemple : mise à jour de fusion de configuration (pas de mode conteneur)

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "DEFAULT_DELIVERY_STREAM_ARN": "arn:aws:firehose:us-
west-2:123456789012:deliverystream/mystream"
    }
  },
}
```

```
"containerMode": "NoContainer"
}
```

Données d'entrée

Ce composant accepte le contenu du flux sur les sujets suivants et envoie le contenu au flux de diffusion cible. Le composant accepte deux types de données d'entrée :

- les données JSON dans la rubrique `kinesisfirehose/message`.
- les données binaires dans la rubrique `kinesisfirehose/message/binary/#`.

Rubrique par défaut pour les données JSON (publication/abonnement locaux) : `kinesisfirehose/message`

Le message accepte les propriétés suivantes. Les messages d'entrée doivent être au format JSON.

`request`

Données à envoyer au flux de diffusion et au flux de diffusion cible, s'il est différent du flux par défaut.

Type : object qui contient les informations suivantes :

`data`

Données à envoyer au flux de diffusion.

Type : string

`delivery_stream_arn`

(Facultatif) L'ARN du flux de diffusion Firehose cible. Spécifiez cette propriété pour remplacer le flux de diffusion par défaut.

Type : string

`id`

ID arbitraire de la demande. Utilisez cette propriété pour associer une demande d'entrée à une réponse de sortie. Lorsque vous spécifiez cette propriété, le composant définit la `id` propriété de l'objet de réponse sur cette valeur.

Type : string

Exemple 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"
}
```

Rubrique par défaut pour les données binaires (publication/abonnement locaux) :
kinesisfirehose/message/binary/#

Utilisez cette rubrique pour envoyer un message qui contient des données binaires. Le composant n'analyse pas les données binaires. Le composant diffuse les données telles quelles.

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/. Assurez-vous d'inclure la barre oblique finale (/).

Données de sortie

Ce composant publie par défaut les réponses sous forme de données de sortie sur le sujet MQTT suivant. Vous devez spécifier cette rubrique subject dans la configuration de l'[ancien composant routeur d'abonnement](#). Pour plus d'informations sur la façon de s'abonner à des messages sur ce sujet dans vos composants personnalisés, consultez [Publier/souscrire AWS IoT Core des messages MQTT](#).

Rubrique par défaut (AWS IoT Core MQTT) : kinesisfirehose/message/status

Exemple 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 composant détecte une erreur qui peut être réessayée, telle qu'une erreur de connexion, il réessaie de publier dans le lot suivant.

Fichier journal local

Ce composant utilise le fichier journal suivant.

```
/greengrass/v2/logs/aws.greengrass.KinesisFirehose.log
```

Pour consulter les journaux de ce composant

- Exécutez la commande suivante sur le périphérique principal pour afficher le fichier journal de ce composant en temps réel. Remplacez */greengrass/v2* par le chemin d'accès au dossier AWS IoT Greengrass racine.

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

Licences

Ce composant inclut les logiciels/licences tiers suivants :

- [AWS SDK pour 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 composant 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 composant.

Version	Modifications
2.1.10	Version mise à jour pour la version 2.15.0 de Greengrass Nucleus.
2.1.9	Version mise à jour pour la version 2.14.0 de Greengrass Nucleus.
2.1.8	Version mise à jour pour la version 2.13.0 de Greengrass Nucleus.
2.1.7	Version mise à jour pour la version 2.12.0 de Greengrass Nucleus.
2.1.6	Version mise à jour pour la version 2.11.0 de Greengrass Nucleus.
2.1.5	Version mise à jour pour la version 2.10.0 de Greengrass Nucleus.
2.1.4	Version mise à jour pour la version 2.9.0 de Greengrass Nucleus.
2.1.3	Version mise à jour pour la version 2.8.0 de Greengrass Nucleus.
2.1.2	Version mise à jour pour la version 2.7.0 de Greengrass Nucleus.

Version	Modifications
2.1.1	Version mise à jour pour la version 2.6.0 de Greengrass Nucleus.
2.1.0	Nouvelles fonctionnalités Ajoute la prise en charge des configurations de proxy réseau HTTPS. Pour plus d'informations, consultez Connexion au port 443 ou via un proxy réseau et Permettre au périphérique principal de faire confiance à un proxy HTTPS.
2.0.8	Version mise à jour pour la version 2.5.0 de Greengrass Nucleus.
2.0.7	Version mise à jour pour la version 2.4.0 de Greengrass Nucleus.
2.0.6	Version mise à jour pour la version 2.3.0 de Greengrass Nucleus.
2.0.5	Version mise à jour pour la version 2.2.0 de Greengrass Nucleus.
2.0.4	Version mise à jour pour la version 2.1.0 de Greengrass Nucleus.
2.0.3	Première version.

Consultez aussi

- [Qu'est-ce qu'Amazon Data Firehose ?](#) dans le guide du développeur Amazon Data Firehose

Lanceur Lambda

Le composant du lanceur Lambda (`aws.greengrass.LambdaLauncher`) démarre et arrête les AWS Lambda fonctions sur AWS IoT Greengrass les appareils principaux. Ce composant configure également toute conteneurisation et exécute les processus en fonction des utilisateurs que vous spécifiez.

 Note

Lorsque vous déployez un composant de fonction Lambda sur un périphérique principal, le déploiement inclut également ce composant. Pour de plus amples informations, veuillez consulter [Exécuter AWS Lambda des fonctions](#).

Rubriques

- [Versions](#)
- [Type](#)
- [Système d'exploitation](#)
- [Prérequis](#)
- [Dépendances](#)
- [Configuration](#)
- [Fichier journal local](#)
- [Journal des modifications](#)

Versions

Les versions de ce composant sont les suivantes :

- 2,0.x

Type

Ce composant est un composant générique (`aws.greengrass.generic`). Le [noyau Greengrass](#) exécute les scripts de cycle de vie du composant.

Pour de plus amples informations, veuillez consulter [Types de composants](#).

Système d'exploitation

Ce composant ne peut être installé que sur les appareils principaux de Linux.

Prérequis

Ce composant répond aux exigences suivantes :

- Votre appareil principal doit répondre aux exigences pour exécuter les fonctions Lambda. Si vous souhaitez que le périphérique principal exécute des fonctions Lambda conteneurisées, le périphérique doit répondre aux exigences requises. Pour de plus amples informations, veuillez consulter [Exigences relatives à la fonction Lambda](#).
- Le composant Lambda Launcher peut être exécuté dans un VPC.

Dépendances

Lorsque vous déployez un composant, il déploie AWS IoT Greengrass également des versions compatibles de ses dépendances. Cela signifie que vous devez satisfaire aux exigences relatives au composant et à toutes ses dépendances pour réussir le déploiement du composant. Cette section répertorie les dépendances des [versions publiées](#) de ce composant et les contraintes de version sémantiques qui définissent les versions des composants pour chaque dépendance. Vous pouvez également consulter les dépendances de chaque version du composant dans la [AWS IoT Greengrass console](#). Sur la page de détails du composant, recherchez la liste des dépendances.

2.0.11 – 2.0.13

Le tableau suivant répertorie les dépendances pour les versions 2.0.11 à 2.0.13 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Gestionnaire Lambda	>=2,0.0 <2,4.0	Stricte

2.0.9 – 2.0.10

Le tableau suivant répertorie les dépendances pour les versions 2.0.9 à 2.0.10 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Gestionnaire Lambda	>=2,0.0 <2,3.0	Stricte

2.0.4 - 2.0.8

Le tableau suivant répertorie les dépendances pour les versions 2.0.4 à 2.0.8 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Gestionnaire Lambda	>=2,0.0 <2,2.0	Stricte

2.0.3

Le tableau suivant répertorie les dépendances pour la version 2.0.3 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Gestionnaire Lambda	>=2,0,3 <2,10	Stricte

Pour plus d'informations sur les dépendances des composants, consultez la [référence de la recette des composants](#).

Configuration

Ce composant ne possède aucun paramètre de configuration.

Fichier journal local

Ce composant utilise le fichier journal suivant.

```
/greengrass/v2/logs/lambdaFunctionComponentName.log
```

Pour consulter les journaux de ce composant

- Exécutez la commande suivante sur le périphérique principal pour afficher le fichier journal de ce composant en temps réel. Remplacez */greengrass/v2* par le chemin d'accès au dossier AWS IoT Greengrass racine et remplacez *lambdaFunctionComponentName* par le nom du composant de fonction Lambda lancé par ce composant.

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

Journal des modifications

Le tableau suivant décrit les modifications apportées à chaque version du composant.

Version	Modifications
2.0.13	Corrections de bogues et améliorations Correction et amélioration de bogues généraux
2.0.12	Corrections de bogues et améliorations Résout un problème selon lequel le lanceur Lambda pouvait générer une erreur si le processus précédent n'était pas correctement arrêté.
2.0.11	Support pour Lambda Manager 2.3.0.
2.0.10	Corrections de bogues et améliorations • Correction et amélioration de bogues généraux
2.0.9	Version mise à jour pour la version 2.5.0 de Greengrass Nucleus.
2.0.8	Version mise à jour pour la version 2.4.0 de Greengrass Nucleus.
2.0.7	Version mise à jour pour la version 2.3.0 de Greengrass Nucleus.
2.0.6	Améliorations des performances générales et correctifs de bogues.
2.0.4	Corrections de bogues et améliorations • Résout un problème en raison duquel le composant ne passe pas correctement <code>AddGroupOwner</code> au conteneur de fonctions Lambda.
2.0.3	Première version.

Gestionnaire Lambda

Le composant Lambda Manager (`aws.greengrass.LambdaManager`) gère les éléments de travail et la communication entre les processus pour les AWS Lambda fonctions exécutées sur le périphérique principal de Greengrass.

 Note

Lorsque vous déployez un composant de fonction Lambda sur un périphérique principal, le déploiement inclut également ce composant. Pour de plus amples informations, veuillez consulter [Exécuter AWS Lambda des fonctions](#).

Rubriques

- [Versions](#)
- [Système d'exploitation](#)
- [Type](#)
- [Prérequis](#)
- [Dépendances](#)
- [Configuration](#)
- [Fichier journal local](#)
- [Journal des modifications](#)

Versions

Les versions de ce composant sont les suivantes :

- 2.3.x
- 2.2.x
- 2,1x
- 2,0.x

Système d'exploitation

Ce composant ne peut être installé que sur les appareils principaux de Linux.

Type

Ce composant est un composant de plugin (`aws.greengrass.plugin`). Le [noyau Greengrass](#) exécute ce composant dans la même machine virtuelle Java (JVM) que le noyau. Le noyau redémarre lorsque vous modifiez la version de ce composant sur le périphérique principal.

Ce composant utilise le même fichier journal que le noyau Greengrass. Pour de plus amples informations, veuillez consulter [AWS IoT Greengrass Journaux de surveillance](#).

Pour de plus amples informations, veuillez consulter [Types de composants](#).

Prérequis

Ce composant répond aux exigences suivantes :

- Votre appareil principal doit répondre aux exigences pour exécuter les fonctions Lambda. Si vous souhaitez que le périphérique principal exécute des fonctions Lambda conteneurisées, le périphérique doit répondre aux exigences requises. Pour de plus amples informations, veuillez consulter [Exigences relatives à la fonction Lambda](#).
- Le composant Lambda Manager est compatible avec l'exécution dans un VPC.

Dépendances

Lorsque vous déployez un composant, il déploie AWS IoT Greengrass également des versions compatibles de ses dépendances. Cela signifie que vous devez satisfaire aux exigences relatives au composant et à toutes ses dépendances pour réussir le déploiement du composant. Cette section répertorie les dépendances des [versions publiées](#) de ce composant et les contraintes de version sémantiques qui définissent les versions des composants pour chaque dépendance. Vous pouvez également consulter les dépendances de chaque version du composant dans la [AWS IoT Greengrass console](#). Sur la page de détails du composant, recherchez la liste des dépendances.

2.3.7

Le tableau suivant répertorie les dépendances pour la version 2.3.7 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0,0 <2,17,0	Flexible

2.3.6

Le tableau suivant répertorie les dépendances pour la version 2.3.6 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0,0 <2,16,0	Flexible

2.3.5

Le tableau suivant répertorie les dépendances pour la version 2.3.5 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0,0 <2,15,0	Flexible

2.3.4

Le tableau suivant répertorie les dépendances pour la version 2.3.4 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0,0 <2,14,0	Flexible

2.3.2 and 2.3.3

Le tableau suivant répertorie les dépendances pour les versions 2.3.2 et 2.3.3 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0,0 <2,13,0	Flexible

2.2.10 and 2.3.1

Le tableau suivant répertorie les dépendances pour les versions 2.2.10 et 2.3.1 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0,0 <2,12,0	Flexible

2.2.8 and 2.2.9

Le tableau suivant répertorie les dépendances pour les versions 2.2.8 et 2.2.9 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,0.0 < 2,11.0$	Flexible

2.2.7

Le tableau suivant répertorie les dépendances pour la version 2.2.7 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,0,0 < 2,1,0$	Flexible

2.2.6

Le tableau suivant répertorie les dépendances pour la version 2.2.6 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,0.0 < 2,9.0$	Flexible

2.2.5

Le tableau suivant répertorie les dépendances pour la version 2.2.5 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,0.0 < 2,8.0$	Flexible

2.2.4

Le tableau suivant répertorie les dépendances pour la version 2.2.4 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,7.0	Flexible

2.2.1 - 2.2.3

Le tableau suivant répertorie les dépendances pour les versions 2.2.1 à 2.2.3 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,6.0	Flexible

2.2.0

Le tableau suivant répertorie les dépendances pour la version 2.2.0 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,5,0 <2,6,0	Flexible

2.1.3 and 2.1.4

Le tableau suivant répertorie les dépendances pour les versions 2.1.3 et 2.1.4 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,5.0	Flexible

2.1.2

Le tableau suivant répertorie les dépendances pour la version 2.1.2 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,4.0	Flexible

2.1.1

Le tableau suivant répertorie les dépendances pour la version 2.1.1 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,3.0	Flexible

2.1.0

Le tableau suivant répertorie les dépendances pour la version 2.1.0 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,2.0	Flexible

2.0.x

Le tableau suivant répertorie les dépendances pour la version 2.0.x de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0,3 <2,10	Flexible

Pour plus d'informations sur les dépendances des composants, consultez la [référence de la recette des composants](#).

Configuration

Ce composant fournit les paramètres de configuration suivants que vous pouvez personnaliser lorsque vous déployez le composant.

logHandlerMode

Note

Uniquement pour les versions 2.3.0+ de Lambda Manager

Permet de choisir l'implémentation du gestionnaire de journaux Lambda à utiliser. Définissez la valeur sur `optimized` pour utiliser moins de threads pour lire les logs Lambda.

`getResultTimeoutInSeconds`

(Facultatif) Durée maximale en secondes pendant laquelle les fonctions Lambda peuvent être exécutées avant leur expiration.

Par défaut : 60

Fichier journal local

Ce composant utilise le même fichier journal que le composant [Greengrass nucleus](#).

```
/greengrass/v2/logs/greengrass.log
```

Pour consulter les journaux de ce composant

- Exécutez la commande suivante sur le périphérique principal pour afficher le fichier journal de ce composant en temps réel. Remplacez `/greengrass/v2` par le chemin d'accès au dossier AWS IoT Greengrass racine.

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

Journal des modifications

Le tableau suivant décrit les modifications apportées à chaque version du composant.

Version	Modifications
2.3.7	Version mise à jour pour la version 2.16.0 de Greengrass Nucleus.
2.3.6	Version mise à jour pour la version 2.15.0 de Greengrass Nucleus.
2.3.5	Corrections de bogues et améliorations Améliore les performances en utilisant <code>epoll</code> au lieu de <code>nio</code> lorsqu'il est disponible.
2.3.4	Version mise à jour pour la version 2.13.0 de Greengrass Nucleus.

Version	Modifications
2.3.3	Corrections de bogues et améliorations • Correction et amélioration de bogues généraux
2.3.2	Version mise à jour pour la version 2.12.0 de Greengrass Nucleus.
2.3.1	Corrections de bogues et améliorations • Ajuste les niveaux de journalisation pour certaines erreurs.
2.3.0	Nouvelles fonctionnalités • Le gestionnaire de journaux a été optimisé pour réduire la charge du processeur. Utilisez cette fonctionnalité en définissant l'option de configuration <code>logHandlerMode</code> <code>suroptimized</code>. Corrections de bogues et améliorations • Il n'enregistre plus l'intégralité du <code>stacktraceWorkQueueFullException</code>, ce qui améliore les journaux et les performances. • Définit le délai d'arrêt Lambda de 15 secondes à 300 secondes afin d'éviter les délais d'arrêt. • Résout un problème selon lequel les lambdas à la demande peuvent ne pas redémarrer après une modification de configuration.
2.2.11	Corrections de bogues et améliorations • Résout un problème selon lequel la <code>LegacySubscriptionRouter</code> configuration ne se met pas à jour lorsque la configuration Lambda change.
2.2.10	Version mise à jour pour la version 2.11.0 de Greengrass Nucleus.
2.2.9	Corrections de bogues et améliorations Résout un problème d'altération du numéro de port en raison d'une horloge asymétrique.
2.2.8	Version mise à jour pour la version 2.10.0 de Greengrass Nucleus.
2.2.7	Version mise à jour pour la version 2.9.0 de Greengrass Nucleus.
2.2.6	Version mise à jour pour la version 2.8.0 de Greengrass Nucleus.

Version	Modifications
2.2.5	Nouvelles fonctionnalités - Ajoute la prise en charge des caractères génériques de sujets MQTT dans les sources d'événements où vous vous abonnez à des messages locaux publish/subscribe . Cette fonctionnalité nécessite la version 2.6.0 ou ultérieure du composant Greengrass nucleus. - Version mise à jour pour la version 2.7.0 de Greengrass Nucleus.
2.2.4	Version mise à jour pour la version 2.6.0 de Greengrass Nucleus.
2.2.3	Corrections de bogues et améliorations Résout un problème selon lequel plusieurs instances d'une fonction Lambda partagent un même cgroup. Ce composant utilise des cgroups pour gérer l'utilisation des ressources pour les fonctions Lambda.
2.2.2	Corrections de bogues et améliorations Résout un problème en raison duquel les composants de la fonction Lambda épinglés redémarrent de manière inattendue dans certains scénarios.
2.2.1	Corrections de bogues et améliorations Modifie les contraintes de version de dépendance du noyau Greengrass de ce composant pour résoudre un problème de résolution des dépendances.

Version	Modifications
2.2.0	Corrections de bogues et améliorations • Résout un problème en raison duquel les fonctions Lambda ne pouvaient pas écrire de journaux après un redémarrage. • Résout un problème selon lequel l'ancien routeur d'abonnement envoie des messages dupliqués lorsque le sujet contient des caractères génériques. • Résout un problème en raison duquel les fonctions Lambda non épinglées ne pouvaient pas utiliser la bibliothèque de communication interprocessus (IPC) Greengrass dans le Kit SDK des appareils AWS IoT
2.1.4	Corrections de bogues et améliorations • Résout un problème en raison duquel les fonctions Lambda utilisant les environnements d'exécution NodeJS ne traitaient qu'un seul message. • Version mise à jour pour la version 2.5.0 de Greengrass Nucleus.
2.1.3	Version mise à jour pour la version 2.4.0 de Greengrass Nucleus.
2.1.2	Version mise à jour pour la version 2.3.0 de Greengrass Nucleus.
2.1.1	Version mise à jour pour la version 2.2.0 de Greengrass Nucleus.
2.1.0	Version mise à jour pour la version 2.1.0 de Greengrass Nucleus.
2.0.3	Première version.

Environnements d'exécution (runtimes) Lambda

Le composant Lambda runtimes (`aws.greengrass.LambdaRuntimes`) fournit les environnements d'exécution que les appareils principaux de Greengrass utilisent pour exécuter des fonctions. AWS Lambda

 Note

Lorsque vous déployez un composant de fonction Lambda sur un périphérique principal, le déploiement inclut également ce composant. Pour de plus amples informations, veuillez consulter [Exécuter AWS Lambda des fonctions](#).

Rubriques

- [Versions](#)
- [Type](#)
- [Système d'exploitation](#)
- [Prérequis](#)
- [Dépendances](#)
- [Configuration](#)
- [Fichier journal local](#)
- [Journal des modifications](#)

Versions

Les versions de ce composant sont les suivantes :

- 2,0.x

Type

Ce composant est un composant générique (`aws.greengrass.generic`). Le [noyau Greengrass](#) exécute les scripts de cycle de vie du composant.

Pour de plus amples informations, veuillez consulter [Types de composants](#).

Système d'exploitation

Ce composant ne peut être installé que sur les appareils principaux de Linux.

Prérequis

Ce composant répond aux exigences suivantes :

- Votre appareil principal doit répondre aux exigences pour exécuter les fonctions Lambda. Si vous souhaitez que le périphérique principal exécute des fonctions Lambda conteneurisées, le périphérique doit répondre aux exigences requises. Pour de plus amples informations, veuillez consulter [Exigences relatives à la fonction Lambda](#).
- Le composant Lambda runtimes est compatible pour s'exécuter dans un VPC.

Dépendances

Ce composant n'a aucune dépendance.

Configuration

Ce composant ne possède aucun paramètre de configuration.

Fichier journal local

Ce composant ne génère pas de journaux.

Journal des modifications

Le tableau suivant décrit les modifications apportées à chaque version du composant.

Version	Modifications
2.0.9	Corrections de bogues et améliorations Corrige un avertissement de syntaxe avec Python 3.12
2.0.8	Version mise à jour pour la version 2.5.0 de Greengrass Nucleus.
2.0.7	Version mise à jour pour la version 2.4.0 de Greengrass Nucleus.
2.0.6	Version mise à jour pour la version 2.3.0 de Greengrass Nucleus.
2.0.5	Version mise à jour pour la version 2.2.0 de Greengrass Nucleus.
2.0.4	Version mise à jour pour la version 2.1.0 de Greengrass Nucleus.
2.0.3	Première version.

Routeur d'abonnement Legacy

L'ancien routeur d'abonnement (`aws.greengrass.LegacySubscriptionRouter`) gère les abonnements sur l'appareil principal Greengrass. Les abonnements sont une fonctionnalité de la AWS IoT Greengrass version 1 qui définit les sujets que les fonctions Lambda peuvent utiliser pour la messagerie MQTT sur un appareil principal. Pour plus d'informations, consultez la section [Abonnements gérés dans le flux de travail de messagerie MQTT](#) du guide du développeur de la version AWS IoT Greengrass 1.

Vous pouvez utiliser ce composant pour activer les abonnements aux composants du connecteur et aux composants de la fonction Lambda qui utilisent le SDK AWS IoT Greengrass Core.

Note

L'ancien composant routeur d'abonnement n'est requis que si votre fonction Lambda utilise la `publish()` fonction du SDK AWS IoT Greengrass Core. Si vous mettez à jour votre code de fonction Lambda pour utiliser l'interface de communication interprocessus (IPC) de la Kit SDK des appareils AWS IoT V2, vous n'avez pas besoin de déployer l'ancien composant routeur d'abonnement. Pour plus d'informations, consultez les services de [communication interprocessus](#) suivants :

- [Publier/souscrire des messages locaux](#)
- [Publier/souscrire AWS IoT Core des messages MQTT](#)

Rubriques

- [Versions](#)
- [Type](#)
- [Système d'exploitation](#)
- [Prérequis](#)
- [Dépendances](#)
- [Configuration](#)
- [Fichier journal local](#)
- [Journal des modifications](#)

Versions

Les versions de ce composant sont les suivantes :

- 2,1x
- 2,0.x

Type

Ce composant est un composant générique (`aws.greengrass.generic`). Le [noyau Greengrass](#) exécute les scripts de cycle de vie du composant.

Pour de plus amples informations, veuillez consulter [Types de composants](#).

Système d'exploitation

Ce composant ne peut être installé que sur les appareils principaux de Linux.

Prérequis

Ce composant répond aux exigences suivantes :

- L'ancien routeur d'abonnement est compatible pour fonctionner dans un VPC.

Dépendances

Lorsque vous déployez un composant, il déploie AWS IoT Greengrass également des versions compatibles de ses dépendances. Cela signifie que vous devez satisfaire aux exigences relatives au composant et à toutes ses dépendances pour réussir le déploiement du composant. Cette section répertorie les dépendances des [versions publiées](#) de ce composant et les contraintes de version sémantiques qui définissent les versions des composants pour chaque dépendance. Vous pouvez également consulter les dépendances de chaque version du composant dans la [AWS IoT Greengrass console](#). Sur la page de détails du composant, recherchez la liste des dépendances.

2.1.15

Le tableau suivant répertorie les dépendances pour la version 2.1.15 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,0,0 < 2,17,0$	Flexible

2.1.14

Le tableau suivant répertorie les dépendances pour la version 2.1.14 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,0,0 < 2,16,0$	Flexible

2.1.13

Le tableau suivant répertorie les dépendances pour la version 2.1.13 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,0,0 < 2,15,0$	Flexible

2.1.12

Le tableau suivant répertorie les dépendances pour la version 2.1.12 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,0,0 < 2,14,0$	Flexible

2.1.11

Le tableau suivant répertorie les dépendances pour la version 2.1.11 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,0,0 < 2,13,0$	Flexible

2.1.10

Le tableau suivant répertorie les dépendances pour la version 2.1.10 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,12.0	Flexible

2.1.9

Le tableau suivant répertorie les dépendances pour la version 2.1.9 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,11.0	Flexible

2.1.8

Le tableau suivant répertorie les dépendances pour la version 2.1.8 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0,0 <2,1,0	Flexible

2.1.7

Le tableau suivant répertorie les dépendances pour la version 2.1.7 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,9.0	Flexible

2.1.6

Le tableau suivant répertorie les dépendances pour la version 2.1.6 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,8.0	Flexible

2.1.5

Le tableau suivant répertorie les dépendances pour la version 2.1.5 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,7.0	Flexible

2.1.4

Le tableau suivant répertorie les dépendances pour la version 2.1.4 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,6.0	Flexible

2.1.3

Le tableau suivant répertorie les dépendances pour la version 2.1.3 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,5.0	Flexible

2.1.2

Le tableau suivant répertorie les dépendances pour la version 2.1.2 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,4.0	Flexible

2.1.1

Le tableau suivant répertorie les dépendances pour la version 2.1.1 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,3.0	Flexible

2.1.0

Le tableau suivant répertorie les dépendances pour la version 2.1.0 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,2.0	Flexible

2.0.3

Le tableau suivant répertorie les dépendances pour la version 2.0.3 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0,3 <2,10	Flexible

Pour plus d'informations sur les dépendances des composants, consultez la [référence de la recette des composants](#).

Configuration

Ce composant fournit les paramètres de configuration suivants que vous pouvez personnaliser lorsque vous déployez le composant.

v2.1.x

subscriptions

(Facultatif) Les abonnements à activer sur l'appareil principal. Il s'agit d'un objet, où chaque clé est un identifiant unique, et chaque valeur est un objet qui définit l'abonnement pour ce

connecteur. Vous devez configurer un abonnement lorsque vous déployez un composant du connecteur V1 ou une fonction Lambda qui utilise le SDK AWS IoT Greengrass Core.

Chaque objet d'abonnement contient les informations suivantes :

`id`

L'identifiant unique de cet abonnement. Cet ID doit correspondre à la clé de cet objet d'abonnement.

`source`

Fonction Lambda qui utilise le SDK de AWS IoT Greengrass base pour publier des messages MQTT sur les sujets que vous spécifiez. `subject` Spécifiez l'un des éléments suivants :

- Nom d'un composant de fonction Lambda sur le périphérique principal.
Spécifiez le nom du composant avec le `component :` préfixe, tel que **`component:com.example.HelloWorldLambda`**.
- Le nom de ressource Amazon (ARN) d'une fonction Lambda sur le périphérique principal.

 Important

Si la version de la fonction Lambda change, vous devez configurer l'abonnement avec la nouvelle version de la fonction. Dans le cas contraire, ce composant n'acheminera pas les messages tant que la version ne correspondra pas à l'abonnement.

Vous devez spécifier un Amazon Resource Name (ARN) qui inclut la version de la fonction à importer. Vous ne pouvez pas utiliser des alias de version tels que `$LATEST`.

Pour déployer un abonnement pour un composant de connecteur V1, spécifiez le nom du composant ou l'ARN de la fonction Lambda du composant de connecteur.

`subject`

Rubrique ou filtre de rubrique MQTT sur lequel la source et la cible peuvent publier et recevoir des messages. Cette valeur prend en charge les caractères génériques `+` et les caractères génériques du `#` sujet.

target

La cible qui reçoit les messages MQTT sur les sujets que vous spécifiez dans `subject`. L'abonnement indique que la source fonction publie des messages MQTT vers AWS IoT Core ou vers une fonction Lambda sur le périphérique principal. Spécifiez l'un des éléments suivants :

- `cloud`. La source fonction publie des messages MQTT sur AWS IoT Core.
- Nom d'un composant de fonction Lambda sur le périphérique principal. Spécifiez le nom du composant avec le `component` : préfixe, tel que **`component:com.example.HelloWorldLambda`**.
- Le nom de ressource Amazon (ARN) d'une fonction Lambda sur le périphérique principal.

Important

Si la version de la fonction Lambda change, vous devez configurer l'abonnement avec la nouvelle version de la fonction. Dans le cas contraire, ce composant n'acheminera pas les messages tant que la version ne correspondra pas à l'abonnement.

Vous devez spécifier un Amazon Resource Name (ARN) qui inclut la version de la fonction à importer. Vous ne pouvez pas utiliser des alias de version tels que `$LATEST`.

Par défaut : aucun abonnement

Exemple Exemple de mise à jour de configuration (définition d'un abonnement à AWS IoT Core)

L'exemple suivant indique que le composant de la fonction `com.example.HelloWorldLambda` Lambda publie un message MQTT AWS IoT Core sur le sujet. `hello/world`

```
{
  "subscriptions": {
    "Greengrass_HelloWorld_to_cloud": {
      "id": "Greengrass_HelloWorld_to_cloud",
      "source": "component:com.example.HelloWorldLambda",
      "subject": "hello/world",
      "target": "cloud"
    }
  }
}
```

```
}  
}
```

Exemple Exemple de mise à jour de configuration (définition d'un abonnement à une autre fonction Lambda)

L'exemple suivant indique que le composant de fonction `com.example.HelloWorldLambda` publie des messages MQTT au composant de fonction `com.example.MessageRelay` Lambda sur le sujet `hello/world`

```
{  
  "subscriptions": {  
    "Greengrass_HelloWorld_to_MessageRelay": {  
      "id": "Greengrass_HelloWorld_to_MessageRelay",  
      "source": "component:com.example.HelloWorldLambda",  
      "subject": "hello/world",  
      "target": "component:com.example.MessageRelay"  
    }  
  }  
}
```

v2.0.x

subscriptions

(Facultatif) Les abonnements à activer sur l'appareil principal. Il s'agit d'un objet, où chaque clé est un identifiant unique, et chaque valeur est un objet qui définit l'abonnement pour ce connecteur. Vous devez configurer un abonnement lorsque vous déployez un composant du connecteur V1 ou une fonction Lambda qui utilise le SDK AWS IoT Greengrass Core.

Chaque objet d'abonnement contient les informations suivantes :

id

L'identifiant unique de cet abonnement. Cet ID doit correspondre à la clé de cet objet d'abonnement.

source

Fonction Lambda qui utilise le SDK de AWS IoT Greengrass base pour publier des messages MQTT sur les sujets que vous spécifiez. `subject` Spécifiez les paramètres suivants :

- Le nom de ressource Amazon (ARN) d'une fonction Lambda sur le périphérique principal.

 Important

Si la version de la fonction Lambda change, vous devez configurer l'abonnement avec la nouvelle version de la fonction. Dans le cas contraire, ce composant n'acheminera pas les messages tant que la version ne correspondra pas à l'abonnement.

Vous devez spécifier un Amazon Resource Name (ARN) qui inclut la version de la fonction à importer. Vous ne pouvez pas utiliser des alias de version tels que \$LATEST.

Pour déployer un abonnement pour un composant de connecteur V1, spécifiez l'ARN de la fonction Lambda du composant de connecteur.

subject

Rubrique ou filtre de rubrique MQTT sur lequel la source et la cible peuvent publier et recevoir des messages. Cette valeur prend en charge les caractères génériques + et les caractères génériques du # sujet.

target

La cible qui reçoit les messages MQTT sur les sujets que vous spécifiez dans `subject`. L'abonnement indique que la source fonction publie des messages MQTT vers AWS IoT Core ou vers une fonction Lambda sur le périphérique principal. Spécifiez l'un des éléments suivants :

- `cloud`. La source fonction publie des messages MQTT sur AWS IoT Core.
- Le nom de ressource Amazon (ARN) d'une fonction Lambda sur le périphérique principal.

 Important

Si la version de la fonction Lambda change, vous devez configurer l'abonnement avec la nouvelle version de la fonction. Dans le cas contraire, ce composant n'acheminera pas les messages tant que la version ne correspondra pas à l'abonnement.

Vous devez spécifier un Amazon Resource Name (ARN) qui inclut la version de la fonction à importer. Vous ne pouvez pas utiliser des alias de version tels que \$LATEST.

Par défaut : aucun abonnement

Exemple Exemple de mise à jour de configuration (définition d'un abonnement à AWS IoT Core)

L'exemple suivant indique que la Greengrass_HelloWorld fonction publie un message MQTT AWS IoT Core sur le hello/world sujet.

```
"subscriptions": {
  "Greengrass_HelloWorld_to_cloud": {
    "id": "Greengrass_HelloWorld_to_cloud",
    "source": "arn:aws:lambda:us-west-2:123456789012:function:Greengrass_HelloWorld:5",
    "subject": "hello/world",
    "target": "cloud"
  }
}
```

Exemple Exemple de mise à jour de configuration (définition d'un abonnement à une autre fonction Lambda)

L'exemple suivant indique que la Greengrass_HelloWorld fonction publie des messages MQTT Greengrass_MessageRelay sur le hello/world sujet.

```
"subscriptions": {
  "Greengrass_HelloWorld_to_MessageRelay": {
    "id": "Greengrass_HelloWorld_to_MessageRelay",
    "source": "arn:aws:lambda:us-west-2:123456789012:function:Greengrass_HelloWorld:5",
    "subject": "hello/world",
    "target": "arn:aws:lambda:us-west-2:123456789012:function:Greengrass_MessageRelay:5"
  }
}
```

Fichier journal local

Ce composant ne génère pas de journaux.

Journal des modifications

Le tableau suivant décrit les modifications apportées à chaque version du composant.

Version	Modifications
2.1.15	Version mise à jour pour la version 2.16.0 de Greengrass Nucleus.
2.1.14	Version mise à jour pour la version 2.15.0 de Greengrass Nucleus.
2.1.13	Version mise à jour pour la version 2.14.0 de Greengrass Nucleus.
2.1.12	Version mise à jour pour la version 2.13.0 de Greengrass Nucleus.
2.1.11	Version mise à jour pour la version 2.12.0 de Greengrass Nucleus.
2.1.10	Version mise à jour pour la version 2.11.0 de Greengrass Nucleus.
2.1.9	Version mise à jour pour la version 2.10.0 de Greengrass Nucleus.
2.1.8	Version mise à jour pour la version 2.9.0 de Greengrass Nucleus.
2.1.7	Version mise à jour pour la version 2.8.0 de Greengrass Nucleus.
2.1.6	Version mise à jour pour la version 2.7.0 de Greengrass Nucleus.
2.1.5	Version mise à jour pour la version 2.6.0 de Greengrass Nucleus.
2.1.4	Version mise à jour pour la version 2.5.0 de Greengrass Nucleus.
2.1.3	Version mise à jour pour la version 2.4.0 de Greengrass Nucleus.
2.1.2	Version mise à jour pour la version 2.3.0 de Greengrass Nucleus.
2.1.1	Version mise à jour pour la version 2.2.0 de Greengrass Nucleus.

Version	Modifications
2.1.0	Corrections de bugs et améliorations Permet de spécifier les noms des composants au lieu de ARNs for source et target. Si vous spécifiez un nom de composant pour un abonnement, vous n'avez pas besoin de reconfigurer l'abonnement chaque fois que la version de la fonction Lambda change.
2.0.3	Première version.

Console de débogage locale

Le composant de console de débogage local (`aws.greengrass.LocalDebugConsole`) fournit un tableau de bord local qui affiche des informations sur vos AWS IoT Greengrass principaux appareils et leurs composants. Vous pouvez utiliser ce tableau de bord pour déboguer votre appareil principal et gérer les composants locaux.

Important

Nous vous recommandons d'utiliser ce composant uniquement dans les environnements de développement, et non dans les environnements de production. Ce composant permet d'accéder à des informations et à des opérations dont vous n'avez généralement pas besoin dans un environnement de production. Respectez le principe du moindre privilège en déployant ce composant uniquement sur les appareils principaux là où vous en avez besoin.

Rubriques

- [Versions](#)
- [Type](#)
- [Système d'exploitation](#)
- [Prérequis](#)
- [Dépendances](#)
- [Configuration](#)
- [Utilisation](#)
- [Fichier journal local](#)

- [Journal des modifications](#)

Versions

Les versions de ce composant sont les suivantes :

- 2.4.x
- 2.3.x
- 2.2.x
- 2,1x
- 2,0.x

Type

Ce composant est un composant de plugin (`aws.greengrass.plugin`). Le [noyau Greengrass](#) exécute ce composant dans la même machine virtuelle Java (JVM) que le noyau. Le noyau redémarre lorsque vous modifiez la version de ce composant sur le périphérique principal.

Ce composant utilise le même fichier journal que le noyau Greengrass. Pour de plus amples informations, veuillez consulter [AWS IoT Greengrass Journaux de surveillance](#).

Pour de plus amples informations, veuillez consulter [Types de composants](#).

Système d'exploitation

Ce composant peut être installé sur les appareils principaux qui exécutent les systèmes d'exploitation suivants :

- Linux
- Windows

Prérequis

Ce composant répond aux exigences suivantes :

- Vous utilisez un nom d'utilisateur et un mot de passe pour vous connecter au tableau de bord.
Le nom d'utilisateur, qui est `debug`, vous est fourni. Vous devez utiliser la AWS IoT Greengrass

CLI pour créer un mot de passe temporaire qui vous authentifie auprès du tableau de bord d'un appareil principal. Vous devez être en mesure d'utiliser la AWS IoT Greengrass CLI pour utiliser la console de débogage locale. Pour plus d'informations, consultez les exigences de [Greengrass CLI](#). Pour plus d'informations sur la façon de générer le mot de passe et de se connecter, consultez la section [Utilisation des composants de la console de débogage locale](#).

- Le composant de console de débogage local peut être exécuté dans un VPC.

Dépendances

Lorsque vous déployez un composant, il déploie AWS IoT Greengrass également des versions compatibles de ses dépendances. Cela signifie que vous devez satisfaire aux exigences relatives au composant et à toutes ses dépendances pour réussir le déploiement du composant. Cette section répertorie les dépendances des [versions publiées](#) de ce composant et les contraintes de version sémantiques qui définissent les versions des composants pour chaque dépendance. Vous pouvez également consulter les dépendances de chaque version du composant dans la [AWS IoT Greengrass console](#). Sur la page de détails du composant, recherchez la liste des dépendances.

2.4.6

Le tableau suivant répertorie les dépendances pour la version 2.4.6 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,1,0$ $< 2,17,0$	Stricte
Greengrass CLI	$\geq 2,1,0$ $< 2,17,0$	Stricte

2.4.5

Le tableau suivant répertorie les dépendances pour la version 2.4.5 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,1,0$ $< 2,16,0$	Stricte
Greengrass CLI	$\geq 2,1,0$ $< 2,16,0$	Stricte

2.4.4

Le tableau suivant répertorie les dépendances pour la version 2.4.4 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,1,0$ $< 2,15,0$	Stricte
Greengrass CLI	$\geq 2,1,0$ $< 2,15,0$	Stricte

2.4.3

Le tableau suivant répertorie les dépendances pour la version 2.4.3 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,1,0$ $< 2,14,0$	Stricte
Greengrass CLI	$\geq 2,1,0$ $< 2,14,0$	Stricte

2.4.1 – 2.4.2

Le tableau suivant répertorie les dépendances pour les versions 2.4.1 à 2.4.2 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,1,0$ $< 2,13,0$	Stricte
Greengrass CLI	$\geq 2,1,0$ $< 2,13,0$	Stricte

2.4.0

Le tableau suivant répertorie les dépendances pour la version 2.4.0 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,1,0$ $< 2,12,0$	Stricte

Dépendance	Versions compatibles	Type de dépendance
Greengrass CLI	>=2,1,0 <2,12,0	Stricte

2.3.0 and 2.3.1

Le tableau suivant répertorie les dépendances pour les versions 2.3.0 et 2.3.1 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,1,0 <2,12,0	Stricte
Greengrass CLI	>=2,1,0 <2,12,0	Stricte

2.2.9

Le tableau suivant répertorie les dépendances pour la version 2.2.9 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,1,0 <2,12,0	Stricte
Greengrass CLI	>=2,1,0 <2,12,0	Stricte

2.2.8

Le tableau suivant répertorie les dépendances pour la version 2.2.8 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,10 <2,11,0	Stricte
Greengrass CLI	>=2,10 <2,11,0	Stricte

2.2.7

Le tableau suivant répertorie les dépendances pour la version 2.2.7 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,10 <2,1,0	Stricte
Greengrass CLI	>=2,10 <2,1,0	Stricte

2.2.6

Le tableau suivant répertorie les dépendances pour la version 2.2.6 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,10 <2,9,0	Stricte
Greengrass CLI	>=2,10 <2,9,0	Stricte

2.2.5

Le tableau suivant répertorie les dépendances pour la version 2.2.5 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,10 <2,8,0	Stricte
Greengrass CLI	>=2,10 <2,8,0	Stricte

2.2.4

Le tableau suivant répertorie les dépendances pour la version 2.2.4 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,10 <2,7,0	Stricte
Greengrass CLI	>=2,10 <2,7,0	Stricte

2.2.3

Le tableau suivant répertorie les dépendances pour la version 2.2.3 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,10 < 2,6,0$	Stricte
Greengrass CLI	$\geq 2,10 < 2,6,0$	Stricte

2.2.2

Le tableau suivant répertorie les dépendances pour la version 2.2.2 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,10 < 2,5,0$	Stricte
Greengrass CLI	$\geq 2,10 < 2,5,0$	Stricte

2.2.1

Le tableau suivant répertorie les dépendances pour la version 2.2.1 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,10 < 2,4,0$	Stricte
Greengrass CLI	$\geq 2,10 < 2,4,0$	Stricte

2.2.0

Le tableau suivant répertorie les dépendances pour la version 2.2.0 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,10 < 2,3,0$	Stricte

Dépendance	Versions compatibles	Type de dépendance
Greengrass CLI	>=2,10 <2,3,0	Stricte

2.1.0

Le tableau suivant répertorie les dépendances pour la version 2.1.0 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,10 <2,2,0	Stricte
Greengrass CLI	>=2,10 <2,2,0	Stricte

2.0.x

Le tableau suivant répertorie les dépendances pour la version 2.0.x de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0,3 <2,10	Flexible
Greengrass CLI	>=2,0,3 <2,10	Flexible

Pour plus d'informations sur les dépendances des composants, consultez la [référence de la recette des composants](#).

Configuration

Ce composant fournit les paramètres de configuration suivants que vous pouvez personnaliser lorsque vous déployez le composant.

v2.1.x - v2.4.x

httpsEnabled

(Facultatif) Vous pouvez activer la communication HTTPS pour la console de débogage locale. Si vous activez la communication HTTPS, la console de débogage locale crée un certificat

auto-signé. Les navigateurs Web affichent des avertissements de sécurité pour les sites Web qui utilisent des certificats auto-signés. Vous devez donc vérifier le certificat manuellement. Ensuite, vous pouvez contourner l'avertissement. Pour de plus amples informations, veuillez consulter [Utilisation](#).

Par défaut: true

port

(Facultatif) Port sur lequel fournir la console de débogage locale.

Par défaut : 1441

websocketPort

(Facultatif) Le port websocket à utiliser pour la console de débogage locale.

Par défaut : 1442

bindHostname

(Facultatif) Le nom d'hôte à utiliser pour la console de débogage locale.

Si vous [exécutez le logiciel AWS IoT Greengrass Core dans un conteneur Docker](#), définissez ce paramètre sur afin de pouvoir ouvrir la console de débogage locale en dehors du conteneur Docker. 0.0.0.0

Par défaut : localhost

Exemple Exemple : mise à jour de la fusion de configurations

L'exemple de configuration suivant indique d'ouvrir la console de débogage locale sur des ports autres que ceux par défaut et de désactiver le protocole HTTPS.

```
{
  "httpsEnabled": false,
  "port": "10441",
  "websocketPort": "10442"
}
```

v2.0.x

port

(Facultatif) Port sur lequel fournir la console de débogage locale.

Par défaut : 1441

websocketPort

(Facultatif) Le port websocket à utiliser pour la console de débogage locale.

Par défaut : 1442

bindHostname

(Facultatif) Le nom d'hôte à utiliser pour la console de débogage locale.

Si vous [exécutez le logiciel AWS IoT Greengrass Core dans un conteneur Docker](#), définissez ce paramètre sur afin de pouvoir ouvrir la console de débogage locale en dehors du conteneur Docker. 0.0.0.0

Par défaut : localhost

Exemple Exemple : mise à jour de la fusion de configurations

L'exemple de configuration suivant indique d'ouvrir la console de débogage locale sur des ports autres que ceux par défaut.

```
{
  "port": "10441",
  "websocketPort": "10442"
}
```

Utilisation

Pour utiliser la console de débogage locale, créez une session à partir de la CLI Greengrass. Lorsque vous créez une session, la CLI Greengrass fournit un nom d'utilisateur et un mot de passe temporaire que vous pouvez utiliser pour vous connecter à la console de débogage locale.

Suivez ces instructions pour ouvrir la console de débogage locale sur votre périphérique principal ou sur votre ordinateur de développement.

v2.1.x - v2.4.x

Dans les versions 2.1.0 et ultérieures, la console de débogage locale utilise HTTPS par défaut. Lorsque le protocole HTTPS est activé, la console de débogage locale crée un certificat auto-

signé pour sécuriser la connexion. Votre navigateur Web affiche un avertissement de sécurité lorsque vous ouvrez la console de débogage locale en raison de ce certificat auto-signé. Lorsque vous créez une session avec la CLI Greengrass, la sortie inclut les empreintes digitales du certificat, ce qui vous permet de vérifier que le certificat est légitime et que la connexion est sécurisée.

Vous pouvez désactiver le protocole HTTPS. Pour plus d'informations, voir [Configuration de la console de débogage locale](#).

Pour ouvrir la console de débogage locale

1. (Facultatif) Pour afficher la console de débogage locale sur votre ordinateur de développement, vous pouvez transférer le port de la console via SSH. Cependant, vous devez d'abord activer l'AllowTcpForwardingoption dans le fichier de configuration SSH de votre appareil principal. Cette option est activée par défaut. Exécutez la commande suivante sur votre ordinateur de développement pour afficher le tableau de bord localhost:1441 sur votre ordinateur de développement.

```
ssh -L 1441:localhost:1441 -L 1442:localhost:1442 username@core-device-ip-address
```

 Note

Vous pouvez modifier les ports par défaut à partir de 1441 et 1442. Pour plus d'informations, voir [Configuration de la console de débogage locale](#).

2. Créez une session pour utiliser la console de débogage locale. Lorsque vous créez une session, vous générez un mot de passe que vous utilisez pour vous authentifier. La console de débogage locale nécessite un mot de passe pour renforcer la sécurité, car vous pouvez utiliser ce composant pour afficher des informations importantes et effectuer des opérations sur le périphérique principal. La console de débogage locale crée également un certificat pour sécuriser la connexion si vous activez le protocole HTTPS dans la configuration du composant. Le protocole HTTPS est activé par défaut.

Utilisez la AWS IoT Greengrass CLI pour créer la session. Cette commande génère un mot de passe aléatoire de 43 caractères qui expire au bout de 8 heures. Remplacez */greengrass/v2* ou *C:\greengrass\v2* par le chemin d'accès au dossier AWS IoT Greengrass V2 racine.

Linux or Unix

```
sudo /greengrass/v2/bin/greengrass-cli get-debug-password
```

Windows

```
C:\greengrass\v2\bin\greengrass-cli get-debug-password
```

La sortie de commande ressemble à l'exemple suivant si vous avez configuré la console de débogage locale pour utiliser le protocole HTTPS. Vous utilisez les empreintes du certificat pour vérifier que la connexion est sécurisée lorsque vous ouvrez la console de débogage locale.

```
Username: debug
Password: bEDp3M0Hdj8ou2w5de_sCBI2XAaguy3a8XxREXAMPLE
Password expires at: 2021-04-01T17:01:43.921999931-07:00
The local debug console is configured to use TLS security. The certificate is
self-signed so you will need to bypass your web browser's security warnings to
open the console.
Before you bypass the security warning, verify that the certificate fingerprint
matches the following fingerprints.
SHA-256: 15 0B 2C E2 54 8B 22 DE 08 46 54 8A B1 2B 25 DE FB 02 7D 01 4E 4A 56 67
96 DA A6 CC B1 D2 C4 1B
SHA-1: BC 3E 16 04 D3 80 70 DA E0 47 25 F9 90 FA D6 02 80 3E B5 C1
```

Le composant debug view crée une session d'une durée de 8 heures. Ensuite, vous devez générer un nouveau mot de passe pour afficher à nouveau la console de débogage locale.

3. Ouvrez le tableau de bord et connectez-vous à celui-ci. Consultez le tableau de bord sur votre appareil principal Greengrass ou sur votre ordinateur de développement si vous transférez le port via SSH. Effectuez l'une des actions suivantes :
 - Si vous avez activé le protocole HTTPS dans la console de débogage locale, qui est le paramètre par défaut, procédez comme suit :
 - a. Ouvrez `https://localhost:1441` sur votre périphérique principal ou sur votre ordinateur de développement si vous avez transféré le port via SSH.

Votre navigateur peut afficher un avertissement de sécurité concernant un certificat de sécurité non valide.

- b. Si votre navigateur affiche un avertissement de sécurité, vérifiez que le certificat est légitime et contournez l'avertissement de sécurité. Procédez comme suit :
 - i. Recherchez l'empreinte SHA-256 ou SHA-1 du certificat et vérifiez qu'elle correspond à l'empreinte SHA-256 ou SHA-1 précédemment imprimée par la commande `get-debug-password`. Votre navigateur peut fournir une ou les deux empreintes digitales. Consultez la documentation de votre navigateur pour consulter le certificat et ses empreintes digitales. Dans certains navigateurs, l'empreinte du certificat est appelée empreinte numérique.

 Note

Si l'empreinte du certificat ne correspond pas, accédez [Step 2](#) à pour créer une nouvelle session. Si l'empreinte du certificat ne correspond toujours pas, votre connexion n'est peut-être pas sécurisée.

- ii. Si l'empreinte du certificat correspond, ignorez l'avertissement de sécurité de votre navigateur pour ouvrir la console de débogage locale. Consultez la documentation de votre navigateur pour contourner l'avertissement de sécurité du navigateur.
- c. Connectez-vous au site Web en utilisant le nom d'utilisateur et le mot de passe que la `get-debug-password` commande a imprimés précédemment.

La console de débogage locale s'ouvre.

- d. Si la console de débogage locale affiche une erreur indiquant qu'elle ne peut pas se connecter au WebSocket en raison d'un échec de connexion TLS, vous devez ignorer l'avertissement de sécurité auto-signé pour l'URL. WebSocket

 Error connecting to WebSocket

The connection was closed due to a failure to perform a TLS handshake

Try opening <https://localhost:1442> and bypass any warnings, then reload this page. The WebSocket connection uses the same certificate as this page.

Procédez comme suit :

- i. Ouvrez `https://localhost:1442` dans le même navigateur que celui dans lequel vous avez ouvert la console de débogage locale.
- ii. Vérifiez le certificat et contournez l'avertissement de sécurité.

Votre navigateur peut afficher une page HTTP 404 une fois que vous avez ignoré l'avertissement.

- iii. Ouvrez `https://localhost:1441` à nouveau.

La console de débogage locale affiche des informations sur le périphérique principal.

- Si vous avez désactivé le protocole HTTPS dans la console de débogage locale, procédez comme suit :
 - a. `http://localhost:1441` Ouvrez-le sur votre périphérique principal ou ouvrez-le sur votre ordinateur de développement si vous avez transféré le port via SSH.
 - b. Connectez-vous au site Web en utilisant le nom d'utilisateur et le mot de passe que la `get-debug-password` commande a précédemment imprimés.

La console de débogage locale s'ouvre.

v2.0.x

Pour ouvrir la console de débogage locale

1. (Facultatif) Pour afficher la console de débogage locale sur votre ordinateur de développement, vous pouvez transférer le port de la console via SSH. Cependant, vous devez d'abord activer l'`AllowTcpForwarding` option dans le fichier de configuration SSH de votre appareil principal. Cette option est activée par défaut. Exécutez la commande suivante sur votre ordinateur de développement pour afficher le tableau de bord `localhost:1441` sur votre ordinateur de développement.

```
ssh -L 1441:localhost:1441 -L 1442:localhost:1442 username@core-device-ip-address
```

 Note

Vous pouvez modifier les ports par défaut à partir de 1441 et 1442. Pour plus d'informations, voir [Configuration de la console de débogage locale](#).

2. Créez une session pour utiliser la console de débogage locale. Lorsque vous créez une session, vous générez un mot de passe que vous utilisez pour vous authentifier. La console de débogage locale nécessite un mot de passe pour renforcer la sécurité, car vous pouvez utiliser ce composant pour afficher des informations importantes et effectuer des opérations sur le périphérique principal.

Utilisez la AWS IoT Greengrass CLI pour créer la session. Cette commande génère un mot de passe aléatoire de 43 caractères qui expire au bout de 8 heures. Remplacez `/greengrass/v2` ou `C:\greengrass\v2` par le chemin d'accès au dossier AWS IoT Greengrass V2 racine.

Linux or Unix

```
sudo /greengrass/v2/bin/greengrass-cli get-debug-password
```

Windows

```
C:\greengrass\v2\bin\greengrass-cli get-debug-password
```

La sortie de commande ressemble à l'exemple suivant.

```
Username: debug
Password: bEDp3M0Hdj8ou2w5de_sCBI2XAaguy3a8XxREXAMPLE
Password will expire at: 2021-04-01T17:01:43.921999931-07:00
```

Le composant d'affichage du débogage crée une session qui dure 4 heures, puis vous devez générer un nouveau mot de passe pour afficher à nouveau la console de débogage locale.

3. `http://localhost:1441` Ouvrez-le sur votre périphérique principal ou ouvrez-le sur votre ordinateur de développement si vous avez transféré le port via SSH.
4. Connectez-vous au site Web en utilisant le nom d'utilisateur et le mot de passe que la `get-debug-password` commande a précédemment imprimés.

La console de débogage locale s'ouvre.

Fichier journal local

Ce composant utilise le même fichier journal que le composant [Greengrass nucleus](#).

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

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

Pour consulter les journaux de ce composant

- Exécutez la commande suivante sur le périphérique principal pour afficher le fichier journal de ce composant en temps réel. Remplacez `/greengrass/v2` ou `C:\greengrass\v2` par le chemin d'accès au dossier AWS IoT Greengrass racine.

Linux

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

Windows (PowerShell)

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

Journal des modifications

Le tableau suivant décrit les modifications apportées à chaque version du composant.

Version	Modifications
2.4.6	Version mise à jour pour la version 2.16.0 de Greengrass Nucleus.
2.4.5	Version mise à jour pour la version 2.15.0 de Greengrass Nucleus.

Version	Modifications
2.4.4	Version mise à jour pour la version 2.14.0 de Greengrass Nucleus.
2.4.3	Version mise à jour pour la version 2.13.0 de Greengrass Nucleus. Corrections de bogues et améliorations • Résout un problème qui entraînait un affichage incorrect <code>STREAM_MANAGER_EXPORTER_MAX_BANDWIDTH</code> en mégabits par seconde (Mbits/s) au lieu d'octets par seconde (Mbits/s).
2.4.2	Corrections de bogues et améliorations • Correction et amélioration de bogues généraux
2.4.1	Version mise à jour pour la version 2.12.0 de Greengrass Nucleus.
2.4.0	Nouvelles fonctionnalités • Ajoute une console de débogage au gestionnaire de flux.
2.3.1	Version mise à jour pour la version 2.11.0 de Greengrass Nucleus.
2.3.0	Version mise à jour pour la version 2.10.0 de Greengrass Nucleus. Nouvelles fonctionnalités • Inclut un PubSub client de débogage AWS IoT Core MQTT.
2.2.7	Version mise à jour pour la version 2.9.0 de Greengrass Nucleus.
2.2.6	Version mise à jour pour la version 2.8.0 de Greengrass Nucleus.
2.2.5	Version mise à jour pour la version 2.7.0 de Greengrass Nucleus.
2.2.4	Version mise à jour pour la version 2.6.0 de Greengrass Nucleus.
2.2.3	Corrections de bogues et améliorations • Résout un problème qui empêchait le démarrage lorsque le composant ne parvenait pas à déchiffrer le keystore contenant la clé privée SSL. • Version mise à jour pour la version 2.5.0 de Greengrass Nucleus.

Version	Modifications
2.2.2	Version mise à jour pour la version 2.4.0 de Greengrass Nucleus.
2.2.1	Version mise à jour pour la version 2.3.0 de Greengrass Nucleus.
2.2.0	Version mise à jour pour la version 2.2.0 de Greengrass Nucleus.
2.1.0	Nouvelles fonctionnalités • Utilise le protocole HTTPS pour sécuriser votre connexion à la console de débogage locale. Le protocole HTTPS est activé par défaut. Corrections de bogues et améliorations • Vous pouvez ignorer les messages de la barre flash dans l'éditeur de configuration.
2.0.3	Première version.

Gestionnaire de journaux

Warning

Nous vous recommandons de passer à Log Manager v2.3.5 ou version ultérieure. La version 2.3.5 optimise les écritures de configuration de Log Manager, en réduisant les opérations d'E/S, en améliorant la vitesse de téléchargement des journaux, les performances globales des appareils et en prolongeant éventuellement leur durée de vie.

Le composant du gestionnaire de journaux (`aws.greengrass.LogManager`) télécharge les journaux des appareils AWS IoT Greengrass principaux vers Amazon CloudWatch Logs. Vous pouvez télécharger des journaux à partir du noyau Greengrass, d'autres composants de Greengrass et d'autres applications et services qui ne sont pas des composants de Greengrass. Pour plus d'informations sur la façon de surveiller les journaux dans CloudWatch les journaux et sur le système de fichiers local, consultez [AWS IoT Greengrass Journaux de surveillance](#).

Les considérations suivantes s'appliquent lorsque vous utilisez le composant du gestionnaire de CloudWatch journaux pour écrire dans Logs :

- Retards de journalisation

Le composant du gestionnaire de journaux version 2.2.8 (et versions antérieures) traite et télécharge les journaux uniquement à partir de fichiers journaux ayant fait l'objet d'une rotation. Par défaut, le logiciel AWS IoT Greengrass Core fait pivoter les fichiers journaux toutes les heures ou une fois qu'ils atteignent 1 024 Ko. Par conséquent, le composant du gestionnaire de journaux ne télécharge les journaux qu'une fois que le logiciel AWS IoT Greengrass Core ou un composant Greengrass a écrit plus de 1 024 Ko de journaux. Vous pouvez configurer une limite de taille de fichier journal inférieure afin de provoquer une rotation plus fréquente des fichiers journaux. Le composant du gestionnaire de journaux télécharge donc les CloudWatch journaux dans Logs plus fréquemment.

Le composant du gestionnaire de journaux version 2.3.0 (et versions ultérieures) traite et télécharge tous les journaux. Lorsque vous rédigez un nouveau journal, la version 2.3.0 (et versions ultérieures) du gestionnaire de journaux traite et télécharge directement le fichier journal actif au lieu d'attendre qu'il soit pivoté. Cela signifie que vous pouvez consulter le nouveau journal en 5 minutes ou moins.

Le composant du gestionnaire de journaux télécharge régulièrement de nouveaux journaux. Par défaut, le composant du gestionnaire de journaux télécharge de nouveaux journaux toutes les 5 minutes. Vous pouvez configurer un intervalle de téléchargement plus court, afin que le composant du gestionnaire de journaux télécharge les CloudWatch journaux vers Logs plus fréquemment en configurant `leperiodicUploadIntervalSec`. Pour plus d'informations sur la configuration de cet intervalle périodique, consultez [la section Configuration](#).

Les journaux peuvent être téléchargés en temps quasi réel à partir du même système de fichiers Greengrass. Si vous devez observer les journaux en temps réel, pensez à utiliser les [journaux du système de fichiers](#).

 Note

Si vous utilisez différents systèmes de fichiers pour y écrire des journaux, le gestionnaire de journaux revient au comportement des composants du gestionnaire de journaux dans les versions 2.2.8 et antérieures. Pour plus d'informations sur l'accès aux journaux du système de fichiers, consultez la section [Accès aux journaux du système de fichiers](#).

- Horloge oblique

Le composant du gestionnaire de journaux utilise le processus de signature standard de Signature version 4 pour créer des demandes d'API destinées à CloudWatch Logs. Si l'heure système d'un appareil principal est désynchronisée de plus de 15 minutes, CloudWatch Logs rejette les demandes. Pour plus d'informations, consultez [Processus de signature Signature Version 4](#) dans le Références générales AWS.

Pour plus d'informations sur les groupes de journaux et les flux de journaux vers lesquels ce composant télécharge les journaux, consultez [Utilisation](#).

Rubriques

- [Versions](#)
- [Type](#)
- [Système d'exploitation](#)
- [Prérequis](#)
- [Dépendances](#)
- [Configuration](#)
- [Utilisation](#)
- [Fichier journal local](#)
- [Journal des modifications](#)

Versions

Les versions de ce composant sont les suivantes :

- 2.3.x
- 2.2.x
- 2,1x
- 2,0.x

Type

Ce composant est un composant de plugin (`aws.greengrass.plugin`). Le [noyau Greengrass](#) exécute ce composant dans la même machine virtuelle Java (JVM) que le noyau. Le noyau redémarre lorsque vous modifiez la version de ce composant sur le périphérique principal.

Ce composant utilise le même fichier journal que le noyau Greengrass. Pour de plus amples informations, veuillez consulter [AWS IoT Greengrass Journaux de surveillance](#).

Pour de plus amples informations, veuillez consulter [Types de composants](#).

Système d'exploitation

Ce composant peut être installé sur les appareils principaux qui exécutent les systèmes d'exploitation suivants :

- Linux
- Windows

Prérequis

Ce composant répond aux exigences suivantes :

- Le [rôle d'appareil Greengrass](#) doit autoriser les `logs:DescribeLogStreams`, `logs:CreateLogGroup`, `logs:CreateLogStream`, et `logs:PutLogEvents`, comme indiqué dans l'exemple de politique IAM suivant.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:logs:*:*:*"
    }
  ]
}
```

Note

Le [rôle d'appareil Greengrass](#) que vous créez lorsque vous installez le logiciel AWS IoT Greengrass Core inclut par défaut les autorisations indiquées dans cet exemple de politique.

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 Logs.

- Le composant du gestionnaire de journaux peut être exécuté dans un VPC. Pour déployer ce composant dans un VPC, les éléments suivants sont requis.
 - Le composant du gestionnaire de journaux doit disposer d'une connectivité `logs.region.amazonaws.com` dont le point de terminaison VPC est de `com.amazonaws.us-east-1.logs`

Points de terminaison et ports

Ce composant doit être capable d'effectuer des demandes sortantes vers les points de terminaison et les ports suivants, en plus des points de terminaison et des ports requis pour le fonctionnement de base. Pour de plus amples informations, veuillez consulter [Autoriser le trafic des appareils via un proxy ou un pare-feu](#).

Endpoint	Port	Obligatoire	Description
<code>logs.<i>region</i>.amazonaws.com</code>	443	Non	Obligatoire si vous écrivez des journaux dans CloudWatch Logs.

Dépendances

Lorsque vous déployez un composant, il déploie AWS IoT Greengrass également des versions compatibles de ses dépendances. Cela signifie que vous devez satisfaire aux exigences relatives au composant et à toutes ses dépendances pour réussir le déploiement du composant. Cette section répertorie les dépendances des [versions publiées](#) de ce composant et les contraintes de version sémantiques qui définissent les versions des composants pour chaque dépendance. Vous pouvez également consulter les dépendances de chaque version du composant dans la [AWS IoT Greengrass console](#). Sur la page de détails du composant, recherchez la liste des dépendances.

2.3.11

Le tableau suivant répertorie les dépendances pour la version 2.3.11 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,1,0$ $< 2,17,0$	Flexible

2.3.10

Le tableau suivant répertorie les dépendances pour la version 2.3.10 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,10$ $< 2,16,0$	Flexible

2.3.9

Le tableau suivant répertorie les dépendances pour la version 2.3.9 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,10$ $< 2,15,0$	Flexible

2.3.8

Le tableau suivant répertorie les dépendances pour la version 2.3.8 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,10 <2,14.0	Flexible

2.3.7

Le tableau suivant répertorie les dépendances pour la version 2.3.7 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,1,0 <2,13,0	Flexible

2.3.5 and 2.3.6

Le tableau suivant répertorie les dépendances pour les versions 2.3.5 et 2.3.6 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,1,0 <2,12,0	Flexible

2.3.3 – 2.3.4

Le tableau suivant répertorie les dépendances pour les versions 2.3.3 à 2.3.4 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,1,0 <2,11,0	Flexible

2.2.8 – 2.3.2

Le tableau suivant répertorie les dépendances pour les versions 2.2.8 à 2.3.2 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,10 <2,1,0	Flexible

2.2.7

Le tableau suivant répertorie les dépendances pour la version 2.2.7 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,10 < 2,9,0$	Flexible

2.2.6

Le tableau suivant répertorie les dépendances pour la version 2.2.6 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,10 < 2,8,0$	Flexible

2.2.5

Le tableau suivant répertorie les dépendances pour la version 2.2.5 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,10 < 2,7,0$	Flexible

2.2.1 - 2.2.4

Le tableau suivant répertorie les dépendances pour les versions 2.2.1 à 2.2.4 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,10 < 2,6,0$	Flexible

2.1.3 and 2.2.0

Le tableau suivant répertorie les dépendances pour les versions 2.1.3 et 2.2.0 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,10 <2,5,0	Flexible

2.1.2

Le tableau suivant répertorie les dépendances pour la version 2.1.2 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,10 <2,4,0	Flexible

2.1.1

Le tableau suivant répertorie les dépendances pour la version 2.1.1 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,10 <2,3,0	Flexible

2.1.0

Le tableau suivant répertorie les dépendances pour la version 2.1.0 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,10 <2,2,0	Flexible

2.0.x

Le tableau suivant répertorie les dépendances pour la version 2.0.x de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0,3 <2,10	Flexible

Pour plus d'informations sur les dépendances des composants, consultez la [référence de la recette des composants](#).

Configuration

Ce composant fournit les paramètres de configuration suivants que vous pouvez personnaliser lorsque vous déployez le composant.

v2.3.10

logsUploaderConfiguration

(Facultatif) Configuration des journaux que le composant du gestionnaire de journaux télécharge. Cet objet contient les informations suivantes :

systemLogsConfiguration

(Facultatif) Configuration des journaux du système du logiciel AWS IoT Greengrass Core, qui incluent les journaux du [noyau Greengrass](#) et des composants du [plugin](#). Spécifiez cette configuration pour permettre au composant du gestionnaire de journaux de gérer les journaux système. Cet objet contient les informations suivantes :

uploadToCloudWatch

(Facultatif) Vous pouvez télécharger les journaux du système dans CloudWatch Logs.

Par défaut : `false`

minimumLogLevel

(Facultatif) Le niveau minimum de messages de journal à télécharger. Ce niveau minimum s'applique uniquement si vous configurez le [composant Greengrass noyau pour générer des journaux](#) au format JSON. Pour activer les journaux au format JSON, spécifiez JSON le paramètre de [format de journalisation](#) (`logging.format`).

Choisissez parmi les niveaux de journalisation suivants, listés ici par ordre de niveau :

- DEBUG
- INFO
- WARN
- ERROR

Par défaut : `INFO`

diskSpaceLimit

(Facultatif) Taille totale maximale des fichiers journaux du système Greengrass, dans l'unité que vous spécifiez. `diskSpaceLimitUnit` Lorsque la taille totale des fichiers journaux du système Greengrass dépasse cette taille totale maximale, le logiciel AWS IoT Greengrass Core supprime les plus anciens fichiers journaux du système Greengrass.

Ce paramètre est équivalent au paramètre de [limite de taille logarithmique](#) (`totalLogsSizeKB`) du composant du [noyau Greengrass](#). Le logiciel AWS IoT Greengrass Core utilise le minimum des deux valeurs comme taille maximale totale du journal du système Greengrass.

diskSpaceLimitUnit

(Facultatif) L'unité pour `diskSpaceLimit`. Sélectionnez parmi les options suivantes :

- KB— kilo-octets
- MB— méga-octets
- GB— giga-octets

Par défaut : KB

deleteLogFileAfterCloudUpload

(Facultatif) Vous pouvez supprimer un fichier journal une fois que le composant du gestionnaire de journaux a chargé les journaux dans CloudWatch Logs.

Par défaut : `false`

componentLogsConfigurationMap

(Facultatif) Carte des configurations de journal pour les composants du périphérique principal. Chaque `componentName` objet de cette carte définit la configuration du journal pour le composant ou l'application. Le composant du gestionnaire de journaux télécharge les journaux de ces composants dans CloudWatch Logs.

Important

Nous vous recommandons vivement d'utiliser une seule clé de configuration par composant. Vous ne devez cibler qu'un groupe de fichiers dont un seul fichier journal est activement écrit lorsque vous utilisez `logFileRegex`. Le non-respect

de cette recommandation peut entraîner le téléchargement de journaux dupliqués vers CloudWatch. [Si vous ciblez plusieurs fichiers journaux actifs avec une seule expression régulière, nous vous recommandons de passer à la version 2.3.1 ou ultérieure du gestionnaire de journaux et d'envisager de modifier votre configuration à l'aide de l'exemple de configuration.](#)

 Note

Si vous effectuez une mise à niveau depuis une version du gestionnaire de journaux antérieure à la version 2.2.0, vous pouvez continuer à utiliser la `componentLogsConfiguration` liste au lieu de `componentLogsConfigurationMap`. Toutefois, nous vous recommandons vivement d'utiliser le format cartographique afin de pouvoir utiliser les mises à jour de fusion et de réinitialisation pour modifier les configurations de composants spécifiques. Pour plus d'informations sur le `componentLogsConfiguration` paramètre, consultez les paramètres de configuration pour la version 2.1.x de ce composant.

componentName

Configuration du journal pour le *componentName* composant ou l'application pour cette configuration de journal. Vous pouvez spécifier le nom d'un composant Greengrass ou une autre valeur pour identifier ce groupe de logs.

Chaque objet contient les informations suivantes :

`minimumLogLevel`

(Facultatif) Le niveau minimum de messages de journal à télécharger. Ce niveau minimum s'applique uniquement si les journaux de ce composant utilisent un format JSON spécifique, que vous pouvez trouver dans le référentiel du [module de AWS IoT Greengrass journalisation](#) sur GitHub.

Choisissez parmi les niveaux de journalisation suivants, listés ici par ordre de niveau :

- DEBUG
- INFO

- WARN
- ERROR

Par défaut : INFO

`diskSpaceLimit`

(Facultatif) Taille totale maximale de tous les fichiers journaux pour ce composant, dans l'unité que vous spécifiez `diskSpaceLimitUnit`. Lorsque la taille totale des fichiers journaux de ce composant dépasse cette taille totale maximale, le logiciel AWS IoT Greengrass Core supprime les fichiers journaux les plus anciens de ce composant.

Ce paramètre est lié au paramètre de [limite de taille logarithmique](#) (`totalLogsSizeKB`) du composant du [noyau Greengrass](#). Le logiciel AWS IoT Greengrass Core utilise le minimum des deux valeurs comme taille maximale totale du journal pour ce composant.

`diskSpaceLimitUnit`

(Facultatif) L'unité pour `diskSpaceLimit`. Sélectionnez parmi les options suivantes :

- KB— kilo-octets
- MB— mégaoctets
- GB— gigaoctets

Par défaut : KB

`logFileDirectoryPath`

(Facultatif) Le chemin d'accès au dossier contenant les fichiers journaux de ce composant.

Il n'est pas nécessaire de spécifier ce paramètre pour les composants Greengrass qui impriment en sortie standard (`stdout`) et en erreur standard (`stderr`).

Par défaut : `/greengrass/v2/logs`.

`logFileRegex`

(Facultatif) Expression régulière qui spécifie le format de nom de fichier journal utilisé par le composant ou l'application. Le composant du gestionnaire de journaux

utilise cette expression régulière pour identifier les fichiers journaux dans le dossier situé à `logFileDirectoryPath`.

Il n'est pas nécessaire de spécifier ce paramètre pour les composants Greengrass qui impriment en sortie standard (stdout) et en erreur standard (stderr).

Si votre composant ou application fait pivoter les fichiers journaux, spécifiez une expression régulière correspondant aux noms des fichiers journaux pivotés. Par exemple, vous pouvez spécifier **hello_world\\\\w*.log** de télécharger les journaux d'une application Hello World. Le `\\\\w*` modèle correspond à zéro ou plusieurs caractères de mot, y compris les caractères alphanumériques et les traits de soulignement. Cette expression régulière fait correspondre les fichiers journaux avec et sans horodatage dans leur nom. Dans cet exemple, le gestionnaire de journaux télécharge les fichiers journaux suivants :

- `hello_world.log`— Le fichier journal le plus récent de l'application Hello World.
- `hello_world_2020_12_15_17_0.log`— Un ancien fichier journal de l'application Hello World.

Par défaut : `componentName\\\\w*.log`, où *componentName* est le nom du composant pour cette configuration de journal.

`deleteLogFileAfterCloudUpload`

(Facultatif) Vous pouvez supprimer un fichier journal une fois que le composant du gestionnaire de journaux a chargé les journaux dans CloudWatch Logs.

Par défaut : `false`

`multiLineStartPattern`

(Facultatif) Expression régulière qui identifie le moment où un message de journal sur une nouvelle ligne est un nouveau message de journal. Si l'expression régulière ne correspond pas à la nouvelle ligne, le composant du gestionnaire de journaux ajoute la nouvelle ligne au message de journal de la ligne précédente.

Par défaut, le composant du gestionnaire de journaux vérifie si la ligne commence par un caractère d'espace, tel qu'un onglet ou un espace. Si ce n'est pas le cas, le gestionnaire de journaux traite cette ligne comme un nouveau message de journal. Sinon, il ajoute cette ligne au message du journal actuel. Ce comportement garantit

que le composant du gestionnaire de journaux ne divise pas les messages qui s'étendent sur plusieurs lignes, tels que les traces de pile.

`periodicUploadIntervalSec`

(Facultatif) Période en secondes pendant laquelle le composant du gestionnaire de journaux vérifie la présence de nouveaux fichiers journaux à télécharger.

Par défaut : 300 (5 minutes)

Minimum : 0.000001 (1 microseconde)

`updateToTlogIntervalSec`

(Facultatif) Période en secondes pendant laquelle le noyau écrit les détails des événements de Amazon CloudWatch Events téléchargement du journal dans le journal des transactions local `()config.tlog`. La valeur par défaut est spécifiée dans `periodicUploadIntervalSec`. Vous pouvez modifier ce paramètre pour augmenter l'intervalle d'écriture.

Par défaut : `periodicUploadIntervalSec`

Minimum : `periodicUploadIntervalSec`

`deprecatedVersionSupport`

Indique si le gestionnaire de journaux doit utiliser les améliorations de vitesse de journalisation introduites dans le gestionnaire de journaux v2.3.5. Définissez la valeur sur `false` pour utiliser les améliorations.

Si vous définissez cette valeur sur `false` lors de la mise à niveau depuis la version 2.3.1 ou une version antérieure du gestionnaire de journaux, des entrées de journal dupliquées peuvent être téléchargées.

La valeur par défaut est `true`.

Exemple Exemple : mise à jour de la fusion de configurations

L'exemple de configuration suivant indique de télécharger les journaux du système et les journaux des `com.example.HelloWorld` composants dans CloudWatch Logs.

```
{
  "logsUploaderConfiguration": {
```



```

"systemLogsConfiguration": {
  "uploadToCloudWatch": "true",
  "minimumLogLevel": "INFO",
  "diskSpaceLimit": "10",
  "diskSpaceLimitUnit": "MB",
  "deleteLogFileAfterCloudUpload": "false"
},
"componentLogsConfigurationMap": {
  "com.example.HelloWorld": {
    "minimumLogLevel": "INFO",
    "diskSpaceLimit": "20",
    "diskSpaceLimitUnit": "MB",
    "deleteLogFileAfterCloudUpload": "false"
  }
}
},
"periodicUploadIntervalSec": "300",
"deprecatedVersionSupport": "false"
}

```

Exemple Exemple : Configuration pour télécharger plusieurs fichiers journaux actifs à l'aide du gestionnaire de journaux v2.3.1

L'exemple de configuration suivant est recommandé si vous souhaitez cibler plusieurs fichiers journaux actifs. Cet exemple de configuration indique dans quels fichiers journaux actifs vous souhaitez télécharger CloudWatch. À l'aide de cet exemple de configuration, la configuration téléchargera également tous les fichiers pivotés correspondant au `logFileRegex`. Cet exemple de configuration est pris en charge sur le gestionnaire de journaux v2.3.1.

```

{
  "logsUploaderConfiguration": {
    "componentLogsConfigurationMap": {
      "com.example.A": {
        "logFileRegex": "com.example.A\\w*.log",
        "deleteLogFileAfterCloudUpload": "false"
      }
      "com.example.B": {
        "logFileRegex": "com.example.B\\w*.log",
        "deleteLogFileAfterCloudUpload": "false"
      }
    }
  },

```

```
"periodicUploadIntervalSec": "10"
}
```

v2.3.6 – v2.3.9

logsUploaderConfiguration

(Facultatif) Configuration des journaux que le composant du gestionnaire de journaux télécharge. Cet objet contient les informations suivantes :

systemLogsConfiguration

(Facultatif) Configuration des journaux du système du logiciel AWS IoT Greengrass Core, qui incluent les journaux du [noyau Greengrass](#) et des composants du [plugin](#). Spécifiez cette configuration pour permettre au composant du gestionnaire de journaux de gérer les journaux système. Cet objet contient les informations suivantes :

uploadToCloudWatch

(Facultatif) Vous pouvez télécharger les journaux du système dans CloudWatch Logs.

Par défaut : `false`

minimumLogLevel

(Facultatif) Le niveau minimum de messages de journal à télécharger. Ce niveau minimum s'applique uniquement si vous configurez le [composant Greengrass noyau pour générer des journaux](#) au format JSON. Pour activer les journaux au format JSON, spécifiez JSON le paramètre de [format de journalisation](#) (`logging.format`).

Choisissez parmi les niveaux de journalisation suivants, listés ici par ordre de niveau :

- DEBUG
- INFO
- WARN
- ERROR

Par défaut : `INFO`

diskSpaceLimit

(Facultatif) Taille totale maximale des fichiers journaux du système Greengrass, dans l'unité que vous spécifiez. `diskSpaceLimitUnit` Lorsque la taille totale des fichiers

journaux du système Greengrass dépasse cette taille totale maximale, le logiciel AWS IoT Greengrass Core supprime les plus anciens fichiers journaux du système Greengrass.

Ce paramètre est équivalent au paramètre de [limite de taille logarithmique](#) (`totalLogsSizeKB`) du composant du [noyau Greengrass](#). Le logiciel AWS IoT Greengrass Core utilise le minimum des deux valeurs comme taille maximale totale du journal du système Greengrass.

`diskSpaceLimitUnit`

(Facultatif) L'unité pour `diskSpaceLimit`. Sélectionnez parmi les options suivantes :

- KB— kilo-octets
- MB— méga-octets
- GB— giga-octets

Par défaut : KB

`deleteLogFileAfterCloudUpload`

(Facultatif) Vous pouvez supprimer un fichier journal une fois que le composant du gestionnaire de journaux a chargé les journaux dans CloudWatch Logs.

Par défaut : `false`

`componentLogsConfigurationMap`

(Facultatif) Carte des configurations de journal pour les composants du périphérique principal. Chaque `componentName` objet de cette carte définit la configuration du journal pour le composant ou l'application. Le composant du gestionnaire de journaux télécharge les journaux de ces composants dans CloudWatch Logs.

Important

Nous vous recommandons vivement d'utiliser une seule clé de configuration par composant. Vous ne devez cibler qu'un groupe de fichiers dont un seul fichier journal est activement écrit lorsque vous utilisez `logFileRegex`. Le non-respect de cette recommandation peut entraîner le téléchargement de journaux dupliqués vers CloudWatch. [Si vous ciblez plusieurs fichiers journaux actifs avec une seule expression régulière, nous vous recommandons de passer à la version 2.3.1 ou](#)

[ultérieure du gestionnaire de journaux et d'envisager de modifier votre configuration à l'aide de l'exemple de configuration.](#)

 Note

Si vous effectuez une mise à niveau depuis une version du gestionnaire de journaux antérieure à la version 2.2.0, vous pouvez continuer à utiliser la `componentLogsConfiguration` liste au lieu de `componentLogsConfigurationMap`. Toutefois, nous vous recommandons vivement d'utiliser le format cartographique afin de pouvoir utiliser les mises à jour de fusion et de réinitialisation pour modifier les configurations de composants spécifiques. Pour plus d'informations sur le `componentLogsConfiguration` paramètre, consultez les paramètres de configuration pour la version 2.1.x de ce composant.

componentName

Configuration du journal pour le *componentName* composant ou l'application pour cette configuration de journal. Vous pouvez spécifier le nom d'un composant Greengrass ou une autre valeur pour identifier ce groupe de logs.

Chaque objet contient les informations suivantes :

`minimumLogLevel`

(Facultatif) Le niveau minimum de messages de journal à télécharger. Ce niveau minimum s'applique uniquement si les journaux de ce composant utilisent un format JSON spécifique, que vous pouvez trouver dans le référentiel du [module de AWS IoT Greengrass journalisation](#) sur GitHub.

Choisissez parmi les niveaux de journalisation suivants, listés ici par ordre de niveau :

- DEBUG
- INFO
- WARN
- ERROR

Par défaut : INFO

diskSpaceLimit

(Facultatif) Taille totale maximale de tous les fichiers journaux pour ce composant, dans l'unité que vous spécifiez `diskSpaceLimitUnit`. Lorsque la taille totale des fichiers journaux de ce composant dépasse cette taille totale maximale, le logiciel AWS IoT Greengrass Core supprime les fichiers journaux les plus anciens de ce composant.

Ce paramètre est lié au paramètre de [limite de taille logarithmique](#) (`totalLogsSizeKB`) du composant du [noyau Greengrass](#). Le logiciel AWS IoT Greengrass Core utilise le minimum des deux valeurs comme taille maximale totale du journal pour ce composant.

diskSpaceLimitUnit

(Facultatif) L'unité pour `diskSpaceLimit`. Sélectionnez parmi les options suivantes :

- KB— kilo-octets
- MB— mégaoctets
- GB— gigaoctets

Par défaut : KB

logFileDirectoryPath

(Facultatif) Le chemin d'accès au dossier contenant les fichiers journaux de ce composant.

Il n'est pas nécessaire de spécifier ce paramètre pour les composants Greengrass qui impriment en sortie standard (stdout) et en erreur standard (stderr).

Par défaut : `/greengrass/v2/logs`.

logFileRegex

(Facultatif) Expression régulière qui spécifie le format de nom de fichier journal utilisé par le composant ou l'application. Le composant du gestionnaire de journaux utilise cette expression régulière pour identifier les fichiers journaux dans le dossier situé à `logFileDirectoryPath`.

Il n'est pas nécessaire de spécifier ce paramètre pour les composants Greengrass qui impriment en sortie standard (stdout) et en erreur standard (stderr).

Si votre composant ou application fait pivoter les fichiers journaux, spécifiez une expression régulière correspondant aux noms des fichiers journaux pivotés. Par exemple, vous pouvez spécifier **hello_world\\\\w*.log** de télécharger les journaux d'une application Hello World. Le **\\\\w*** modèle correspond à zéro ou plusieurs caractères de mot, y compris les caractères alphanumériques et les traits de soulignement. Cette expression régulière fait correspondre les fichiers journaux avec et sans horodatage dans leur nom. Dans cet exemple, le gestionnaire de journaux télécharge les fichiers journaux suivants :

- **hello_world.log**— Le fichier journal le plus récent de l'application Hello World.
- **hello_world_2020_12_15_17_0.log**— Un ancien fichier journal de l'application Hello World.

Par défaut : **componentName\\\\w*.log**, où **componentName** est le nom du composant pour cette configuration de journal.

`deleteLogFileAfterCloudUpload`

(Facultatif) Vous pouvez supprimer un fichier journal une fois que le composant du gestionnaire de journaux a chargé les journaux dans CloudWatch Logs.

Par défaut : `false`

`multilineStartPattern`

(Facultatif) Expression régulière qui identifie le moment où un message de journal sur une nouvelle ligne est un nouveau message de journal. Si l'expression régulière ne correspond pas à la nouvelle ligne, le composant du gestionnaire de journaux ajoute la nouvelle ligne au message de journal de la ligne précédente.

Par défaut, le composant du gestionnaire de journaux vérifie si la ligne commence par un caractère d'espace, tel qu'un onglet ou un espace. Si ce n'est pas le cas, le gestionnaire de journaux traite cette ligne comme un nouveau message de journal. Sinon, il ajoute cette ligne au message du journal actuel. Ce comportement garantit que le composant du gestionnaire de journaux ne divise pas les messages qui s'étendent sur plusieurs lignes, tels que les traces de pile.

periodicUploadIntervalSec

(Facultatif) Période en secondes pendant laquelle le composant du gestionnaire de journaux vérifie la présence de nouveaux fichiers journaux à télécharger.

Par défaut : 300 (5 minutes)

Minimum : 0.000001 (1 microseconde)

deprecatedVersionSupport

Indique si le gestionnaire de journaux doit utiliser les améliorations de vitesse de journalisation introduites dans le gestionnaire de journaux v2.3.5. Définissez la valeur sur `false` pour utiliser les améliorations.

Si vous définissez cette valeur sur `false` lors de la mise à niveau depuis la version 2.3.1 ou une version antérieure du gestionnaire de journaux, des entrées de journal dupliquées peuvent être téléchargées.

La valeur par défaut est `true`.

Exemple Exemple : mise à jour de la fusion de configurations

L'exemple de configuration suivant indique de télécharger les journaux du système et les journaux des `com.example.HelloWorld` composants dans CloudWatch Logs.

```
{
  "logsUploaderConfiguration": {
    "systemLogsConfiguration": {
      "uploadToCloudWatch": "true",
      "minimumLogLevel": "INFO",
      "diskSpaceLimit": "10",
      "diskSpaceLimitUnit": "MB",
      "deleteLogFileAfterCloudUpload": "false"
    },
    "componentLogsConfigurationMap": {
      "com.example.HelloWorld": {
        "minimumLogLevel": "INFO",
        "diskSpaceLimit": "20",
        "diskSpaceLimitUnit": "MB",
        "deleteLogFileAfterCloudUpload": "false"
      }
    }
  }
}
```

```

    }
  },
  "periodicUploadIntervalSec": "300",
  "deprecatedVersionSupport": "false"
}

```

Exemple Exemple : Configuration pour télécharger plusieurs fichiers journaux actifs à l'aide du gestionnaire de journaux v2.3.1

L'exemple de configuration suivant est recommandé si vous souhaitez cibler plusieurs fichiers journaux actifs. Cet exemple de configuration indique dans quels fichiers journaux actifs vous souhaitez télécharger CloudWatch. À l'aide de cet exemple de configuration, la configuration téléchargera également tous les fichiers pivotés correspondant au `logFileRegex`. Cet exemple de configuration est pris en charge sur le gestionnaire de journaux v2.3.1.

```

{
  "logsUploaderConfiguration": {
    "componentLogsConfigurationMap": {
      "com.example.A": {
        "logFileRegex": "com.example.A\\w*.log",
        "deleteLogFileAfterCloudUpload": "false"
      }
      "com.example.B": {
        "logFileRegex": "com.example.B\\w*.log",
        "deleteLogFileAfterCloudUpload": "false"
      }
    }
  },
  "periodicUploadIntervalSec": "10"
}

```

v2.3.0 – 2.3.5

logsUploaderConfiguration

(Facultatif) Configuration des journaux que le composant du gestionnaire de journaux télécharge. Cet objet contient les informations suivantes :

systemLogsConfiguration

(Facultatif) Configuration des journaux du système du logiciel AWS IoT Greengrass Core, qui incluent les journaux du [noyau Greengrass](#) et des composants du [plugin](#). Spécifiez

cette configuration pour permettre au composant du gestionnaire de journaux de gérer les journaux système. Cet objet contient les informations suivantes :

`uploadToCloudWatch`

(Facultatif) Vous pouvez télécharger les journaux du système dans CloudWatch Logs.

Par défaut : `false`

`minimumLogLevel`

(Facultatif) Le niveau minimum de messages de journal à télécharger. Ce niveau minimum s'applique uniquement si vous configurez le [composant Greengrass noyau pour générer des journaux](#) au format JSON. Pour activer les journaux au format JSON, spécifiez JSON le paramètre de [format de journalisation](#) (`logging.format`).

Choisissez parmi les niveaux de journalisation suivants, listés ici par ordre de niveau :

- DEBUG
- INFO
- WARN
- ERROR

Par défaut : `INFO`

`diskSpaceLimit`

(Facultatif) Taille totale maximale des fichiers journaux du système Greengrass, dans l'unité que vous spécifiez. `diskSpaceLimitUnit` Lorsque la taille totale des fichiers journaux du système Greengrass dépasse cette taille totale maximale, le logiciel AWS IoT Greengrass Core supprime les plus anciens fichiers journaux du système Greengrass.

Ce paramètre est équivalent au paramètre de [limite de taille logarithmique](#) (`totalLogsSizeKB`) du composant du [noyau Greengrass](#). Le logiciel AWS IoT Greengrass Core utilise le minimum des deux valeurs comme taille maximale totale du journal du système Greengrass.

`diskSpaceLimitUnit`

(Facultatif) L'unité pour `diskSpaceLimit`. Sélectionnez parmi les options suivantes :

- KB— kilo-octets

- MB— mégaoctets
- GB— gigaoctets

Par défaut : KB

`deleteLogFileAfterCloudUpload`

(Facultatif) Vous pouvez supprimer un fichier journal une fois que le composant du gestionnaire de journaux a chargé les journaux dans CloudWatch Logs.

Par défaut : `false`

`componentLogsConfigurationMap`

(Facultatif) Carte des configurations de journal pour les composants du périphérique principal. Chaque `componentName` objet de cette carte définit la configuration du journal pour le composant ou l'application. Le composant du gestionnaire de journaux télécharge les journaux de ces composants dans CloudWatch Logs.

Important

Nous vous recommandons vivement d'utiliser une seule clé de configuration par composant. Vous ne devez cibler qu'un groupe de fichiers dont un seul fichier journal est activement écrit lorsque vous utilisez `logFileRegex`. Le non-respect de cette recommandation peut entraîner le téléchargement de journaux dupliqués vers CloudWatch. [Si vous ciblez plusieurs fichiers journaux actifs avec une seule expression régulière, nous vous recommandons de passer à la version 2.3.1 du gestionnaire de journaux et d'envisager de modifier votre configuration à l'aide de l'exemple de configuration.](#)

Note

Si vous effectuez une mise à niveau depuis une version du gestionnaire de journaux antérieure à la version 2.2.0, vous pouvez continuer à utiliser la `componentLogsConfiguration` liste au lieu de `componentLogsConfigurationMap`. Toutefois, nous vous recommandons vivement d'utiliser le format cartographique afin de pouvoir utiliser les mises à jour de fusion et de réinitialisation pour modifier les configurations de composants.

spécifiques. Pour plus d'informations sur le `componentLogsConfiguration` paramètre, consultez les paramètres de configuration pour la version 2.1.x de ce composant.

componentName

Configuration du journal pour le *componentName* composant ou l'application pour cette configuration de journal. Vous pouvez spécifier le nom d'un composant Greengrass ou une autre valeur pour identifier ce groupe de logs.

Chaque objet contient les informations suivantes :

`minimumLogLevel`

(Facultatif) Le niveau minimum de messages de journal à télécharger. Ce niveau minimum s'applique uniquement si les journaux de ce composant utilisent un format JSON spécifique, que vous pouvez trouver dans le référentiel du [module de AWS IoT Greengrass journalisation](#) sur GitHub.

Choisissez parmi les niveaux de journalisation suivants, listés ici par ordre de niveau :

- DEBUG
- INFO
- WARN
- ERROR

Par défaut : INFO

`diskSpaceLimit`

(Facultatif) Taille totale maximale de tous les fichiers journaux pour ce composant, dans l'unité que vous spécifiez `diskSpaceLimitUnit`. Lorsque la taille totale des fichiers journaux de ce composant dépasse cette taille totale maximale, le logiciel AWS IoT Greengrass Core supprime les fichiers journaux les plus anciens de ce composant.

Ce paramètre est lié au paramètre de [limite de taille logarithmique](#) (`totalLogsSizeKB`) du composant du [noyau Greengrass](#). Le logiciel AWS IoT

Greengrass Core utilise le minimum des deux valeurs comme taille maximale totale du journal pour ce composant.

`diskSpaceLimitUnit`

(Facultatif) L'unité pour `diskSpaceLimit`. Sélectionnez parmi les options suivantes :

- KB— kilo-octets
- MB— méga-octets
- GB— giga-octets

Par défaut : KB

`logFileDirectoryPath`

(Facultatif) Le chemin d'accès au dossier contenant les fichiers journaux de ce composant.

Il n'est pas nécessaire de spécifier ce paramètre pour les composants Greengrass qui impriment en sortie standard (stdout) et en erreur standard (stderr).

Par défaut : `/greengrass/v2/logs`.

`logFileRegex`

(Facultatif) Expression régulière qui spécifie le format de nom de fichier journal utilisé par le composant ou l'application. Le composant du gestionnaire de journaux utilise cette expression régulière pour identifier les fichiers journaux dans le dossier situé à `logFileDirectoryPath`.

Il n'est pas nécessaire de spécifier ce paramètre pour les composants Greengrass qui impriment en sortie standard (stdout) et en erreur standard (stderr).

Si votre composant ou application fait pivoter les fichiers journaux, spécifiez une expression régulière correspondant aux noms des fichiers journaux pivotés. Par exemple, vous pouvez spécifier **`hello_world\\\\w*.log`** de télécharger les journaux d'une application Hello World. Le `\\\\w*` modèle correspond à zéro ou plusieurs caractères de mot, y compris les caractères alphanumériques et les traits de soulignement. Cette expression régulière fait correspondre les fichiers journaux avec et sans horodatage dans leur nom. Dans cet exemple, le gestionnaire de journaux télécharge les fichiers journaux suivants :

- `hello_world.log`— Le fichier journal le plus récent de l'application Hello World.
- `hello_world_2020_12_15_17_0.log`— Un ancien fichier journal de l'application Hello World.

Par défaut : `componentName\\\\w*.log`, où `componentName` est le nom du composant pour cette configuration de journal.

`deleteLogFileAfterCloudUpload`

(Facultatif) Vous pouvez supprimer un fichier journal une fois que le composant du gestionnaire de journaux a chargé les journaux dans CloudWatch Logs.

Par défaut : `false`

`multiLineStartPattern`

(Facultatif) Expression régulière qui identifie le moment où un message de journal sur une nouvelle ligne est un nouveau message de journal. Si l'expression régulière ne correspond pas à la nouvelle ligne, le composant du gestionnaire de journaux ajoute la nouvelle ligne au message de journal de la ligne précédente.

Par défaut, le composant du gestionnaire de journaux vérifie si la ligne commence par un caractère d'espace, tel qu'un onglet ou un espace. Si ce n'est pas le cas, le gestionnaire de journaux traite cette ligne comme un nouveau message de journal. Sinon, il ajoute cette ligne au message du journal actuel. Ce comportement garantit que le composant du gestionnaire de journaux ne divise pas les messages qui s'étendent sur plusieurs lignes, tels que les traces de pile.

`periodicUploadIntervalSec`

(Facultatif) Période en secondes pendant laquelle le composant du gestionnaire de journaux vérifie la présence de nouveaux fichiers journaux à télécharger.

Par défaut : `300` (5 minutes)

Minimum : `0.000001` (1 microseconde)

Exemple Exemple : mise à jour de la fusion de configurations

L'exemple de configuration suivant indique de télécharger les journaux du système et les journaux des `com.example.HelloWorld` composants dans CloudWatch Logs.

```
{
  "logsUploaderConfiguration": {
    "systemLogsConfiguration": {
      "uploadToCloudWatch": "true",
      "minimumLogLevel": "INFO",
      "diskSpaceLimit": "10",
      "diskSpaceLimitUnit": "MB",
      "deleteLogFileAfterCloudUpload": "false"
    },
    "componentLogsConfigurationMap": {
      "com.example.HelloWorld": {
        "minimumLogLevel": "INFO",
        "diskSpaceLimit": "20",
        "diskSpaceLimitUnit": "MB",
        "deleteLogFileAfterCloudUpload": "false"
      }
    }
  },
  "periodicUploadIntervalSec": "300"
}
```

Exemple Exemple : Configuration pour télécharger plusieurs fichiers journaux actifs à l'aide du gestionnaire de journaux v2.3.1

L'exemple de configuration suivant est recommandé si vous souhaitez cibler plusieurs fichiers journaux actifs. Cet exemple de configuration indique dans quels fichiers journaux actifs vous souhaitez télécharger CloudWatch. À l'aide de cet exemple de configuration, la configuration téléchargera également tous les fichiers pivotés correspondant au `logFileRegex`. Cet exemple de configuration est pris en charge sur le gestionnaire de journaux v2.3.1.

```
{
  "logsUploaderConfiguration": {
    "componentLogsConfigurationMap": {
      "com.example.A": {
        "logFileRegex": "com.example.A\\w*.log",
        "deleteLogFileAfterCloudUpload": "false"
      }
      "com.example.B": {
        "logFileRegex": "com.example.B\\w*.log",
        "deleteLogFileAfterCloudUpload": "false"
      }
    }
  }
}
```

```
},  
"periodicUploadIntervalSec": "10"  
}
```

v2.2.x

logsUploaderConfiguration

(Facultatif) Configuration des journaux que le composant du gestionnaire de journaux télécharge. Cet objet contient les informations suivantes :

systemLogsConfiguration

(Facultatif) Configuration des journaux du système du logiciel AWS IoT Greengrass Core, qui incluent les journaux du [noyau Greengrass](#) et des composants du [plugin](#). Spécifiez cette configuration pour permettre au composant du gestionnaire de journaux de gérer les journaux système. Cet objet contient les informations suivantes :

uploadToCloudWatch

(Facultatif) Vous pouvez télécharger les journaux du système dans CloudWatch Logs.

Par défaut : `false`

minimumLogLevel

(Facultatif) Le niveau minimum de messages de journal à télécharger. Ce niveau minimum s'applique uniquement si vous configurez le [composant Greengrass noyau pour générer des journaux](#) au format JSON. Pour activer les journaux au format JSON, spécifiez JSON le paramètre de [format de journalisation](#) (`logging.format`).

Choisissez parmi les niveaux de journalisation suivants, listés ici par ordre de niveau :

- DEBUG
- INFO
- WARN
- ERROR

Par défaut : `INFO`

diskSpaceLimit

(Facultatif) Taille totale maximale des fichiers journaux du système Greengrass, dans l'unité que vous spécifiez. `diskSpaceLimitUnit` Lorsque la taille totale des fichiers

journaux du système Greengrass dépasse cette taille totale maximale, le logiciel AWS IoT Greengrass Core supprime les plus anciens fichiers journaux du système Greengrass.

Ce paramètre est équivalent au paramètre de [limite de taille logarithmique](#) (`totalLogsSizeKB`) du composant du [noyau Greengrass](#). Le logiciel AWS IoT Greengrass Core utilise le minimum des deux valeurs comme taille maximale totale du journal du système Greengrass.

`diskSpaceLimitUnit`

(Facultatif) L'unité pour `diskSpaceLimit`. Sélectionnez parmi les options suivantes :

- KB— kilo-octets
- MB— méga-octets
- GB— giga-octets

Par défaut : KB

`deleteLogFileAfterCloudUpload`

(Facultatif) Vous pouvez supprimer un fichier journal une fois que le composant du gestionnaire de journaux a chargé les journaux dans CloudWatch Logs.

Par défaut : `false`

`componentLogsConfigurationMap`

(Facultatif) Carte des configurations de journal pour les composants du périphérique principal. Chaque `componentName` objet de cette carte définit la configuration du journal pour le composant ou l'application. Le composant du gestionnaire de journaux télécharge les journaux de ces composants dans CloudWatch Logs.

Note

Si vous effectuez une mise à niveau depuis une version du gestionnaire de journaux antérieure à la version 2.2.0, vous pouvez continuer à utiliser la `componentLogsConfiguration` liste au lieu de `componentLogsConfigurationMap`. Toutefois, nous vous recommandons vivement d'utiliser le format cartographique afin de pouvoir utiliser les mises à jour de fusion et de réinitialisation pour modifier les configurations de composants spécifiques. Pour plus d'informations sur le `componentLogsConfiguration`

paramètre, consultez les paramètres de configuration pour la version 2.1.x de ce composant.

componentName

Configuration du journal pour le *componentName* composant ou l'application pour cette configuration de journal. Vous pouvez spécifier le nom d'un composant Greengrass ou une autre valeur pour identifier ce groupe de logs.

Chaque objet contient les informations suivantes :

`minimumLogLevel`

(Facultatif) Le niveau minimum de messages de journal à télécharger. Ce niveau minimum s'applique uniquement si les journaux de ce composant utilisent un format JSON spécifique, que vous pouvez trouver dans le référentiel du [module de AWS IoT Greengrass journalisation](#) sur GitHub.

Choisissez parmi les niveaux de journalisation suivants, listés ici par ordre de niveau :

- DEBUG
- INFO
- WARN
- ERROR

Par défaut : INFO

`diskSpaceLimit`

(Facultatif) Taille totale maximale de tous les fichiers journaux pour ce composant, dans l'unité que vous spécifiez `diskSpaceLimitUnit`. Lorsque la taille totale des fichiers journaux de ce composant dépasse cette taille totale maximale, le logiciel AWS IoT Greengrass Core supprime les fichiers journaux les plus anciens de ce composant.

Ce paramètre est lié au paramètre de [limite de taille logarithmique](#) (`totalLogsSizeKB`) du composant du [noyau Greengrass](#). Le logiciel AWS IoT Greengrass Core utilise le minimum des deux valeurs comme taille maximale totale du journal pour ce composant.

diskSpaceLimitUnit

(Facultatif) L'unité pour le `diskSpaceLimit`. Sélectionnez parmi les options suivantes :

- KB— kilo-octets
- MB— méga-octets
- GB— giga-octets

Par défaut : KB

logFileDirectoryPath

(Facultatif) Le chemin d'accès au dossier contenant les fichiers journaux de ce composant.

Il n'est pas nécessaire de spécifier ce paramètre pour les composants Greengrass qui impriment en sortie standard (stdout) et en erreur standard (stderr).

Par défaut : `/greengrass/v2/logs`.

logFileRegex

(Facultatif) Expression régulière qui spécifie le format de nom de fichier journal utilisé par le composant ou l'application. Le composant du gestionnaire de journaux utilise cette expression régulière pour identifier les fichiers journaux dans le dossier situé à `logFileDirectoryPath`.

Il n'est pas nécessaire de spécifier ce paramètre pour les composants Greengrass qui impriment en sortie standard (stdout) et en erreur standard (stderr).

Si votre composant ou application fait pivoter les fichiers journaux, spécifiez une expression régulière correspondant aux noms des fichiers journaux pivotés. Par exemple, vous pouvez spécifier `hello_world\\\\w*.log` de télécharger les journaux d'une application Hello World. Le `\\\\w*` modèle correspond à zéro ou plusieurs caractères de mot, y compris les caractères alphanumériques et les traits de soulignement. Cette expression régulière fait correspondre les fichiers journaux avec et sans horodatage dans leur nom. Dans cet exemple, le gestionnaire de journaux télécharge les fichiers journaux suivants :

- `hello_world.log`— Le fichier journal le plus récent de l'application Hello World.

- `hello_world_2020_12_15_17_0.log`— Un ancien fichier journal de l'application Hello World.

Par défaut : `componentName\\\\w*.log`, où `componentName` est le nom du composant pour cette configuration de journal.

`deleteLogFileAfterCloudUpload`

(Facultatif) Vous pouvez supprimer un fichier journal une fois que le composant du gestionnaire de journaux a chargé les journaux dans CloudWatch Logs.

Par défaut : `false`

`multiLineStartPattern`

(Facultatif) Expression régulière qui identifie le moment où un message de journal sur une nouvelle ligne est un nouveau message de journal. Si l'expression régulière ne correspond pas à la nouvelle ligne, le composant du gestionnaire de journaux ajoute la nouvelle ligne au message de journal de la ligne précédente.

Par défaut, le composant du gestionnaire de journaux vérifie si la ligne commence par un caractère d'espace, tel qu'un onglet ou un espace. Si ce n'est pas le cas, le gestionnaire de journaux traite cette ligne comme un nouveau message de journal. Dans le cas contraire, il ajoute cette ligne au message du journal actuel. Ce comportement garantit que le composant du gestionnaire de journaux ne divise pas les messages qui s'étendent sur plusieurs lignes, tels que les traces de pile.

`periodicUploadIntervalSec`

(Facultatif) Période en secondes pendant laquelle le composant du gestionnaire de journaux vérifie la présence de nouveaux fichiers journaux à télécharger.

Par défaut : `300` (5 minutes)

Minimum : `0.000001` (1 microseconde)

Exemple Exemple : mise à jour de la fusion de configurations

L'exemple de configuration suivant indique de télécharger les journaux du système et les journaux des `com.example>HelloWorld` composants dans CloudWatch Logs.

```
{
```

```
"logsUploaderConfiguration": {
  "systemLogsConfiguration": {
    "uploadToCloudWatch": "true",
    "minimumLogLevel": "INFO",
    "diskSpaceLimit": "10",
    "diskSpaceLimitUnit": "MB",
    "deleteLogFileAfterCloudUpload": "false"
  },
  "componentLogsConfigurationMap": {
    "com.example.HelloWorld": {
      "minimumLogLevel": "INFO",
      "diskSpaceLimit": "20",
      "diskSpaceLimitUnit": "MB",
      "deleteLogFileAfterCloudUpload": "false"
    }
  },
  "periodicUploadIntervalSec": "300"
}
```

v2.1.x

logsUploaderConfiguration

(Facultatif) Configuration des journaux que le composant du gestionnaire de journaux télécharge. Cet objet contient les informations suivantes :

systemLogsConfiguration

(Facultatif) Configuration des journaux du système du logiciel AWS IoT Greengrass Core, qui incluent les journaux du [noyau Greengrass](#) et des composants du [plugin](#). Spécifiez cette configuration pour permettre au composant du gestionnaire de journaux de gérer les journaux système. Cet objet contient les informations suivantes :

uploadToCloudWatch

(Facultatif) Vous pouvez télécharger les journaux du système dans CloudWatch Logs.

Par défaut : false

minimumLogLevel

(Facultatif) Le niveau minimum de messages de journal à télécharger. Ce niveau minimum s'applique uniquement si vous configurez le [composant Greengrass noyau](#)

[pour générer des journaux](#) au format JSON. Pour activer les journaux au format JSON, spécifiez JSON le paramètre de [format de journalisation](#) (`logging.format`).

Choisissez parmi les niveaux de journalisation suivants, listés ici par ordre de niveau :

- DEBUG
- INFO
- WARN
- ERROR

Par défaut : INFO

`diskSpaceLimit`

(Facultatif) Taille totale maximale des fichiers journaux du système Greengrass, dans l'unité que vous spécifiez. `diskSpaceLimitUnit` Lorsque la taille totale des fichiers journaux du système Greengrass dépasse cette taille totale maximale, le logiciel AWS IoT Greengrass Core supprime les plus anciens fichiers journaux du système Greengrass.

Ce paramètre est équivalent au paramètre de [limite de taille logarithmique](#) (`totalLogsSizeKB`) du composant du [noyau Greengrass](#). Le logiciel AWS IoT Greengrass Core utilise le minimum des deux valeurs comme taille maximale totale du journal du système Greengrass.

`diskSpaceLimitUnit`

(Facultatif) L'unité pour `diskSpaceLimit`. Sélectionnez parmi les options suivantes :

- KB— kilo-octets
- MB— méga-octets
- GB— giga-octets

Par défaut : KB

`deleteLogFileAfterCloudUpload`

(Facultatif) Vous pouvez supprimer un fichier journal une fois que le composant du gestionnaire de journaux a chargé les journaux dans CloudWatch Logs.

Par défaut : `false`

componentLogsConfiguration

(Facultatif) Liste des configurations de journal pour les composants du périphérique principal. Chaque configuration de cette liste définit la configuration du journal pour un composant ou une application. Le composant du gestionnaire de journaux télécharge les journaux de ces composants dans Logs. CloudWatch

Chaque objet contient les informations suivantes :

componentName

Nom du composant ou de l'application pour cette configuration de journal. Vous pouvez spécifier le nom d'un composant Greengrass ou une autre valeur pour identifier ce groupe de logs.

minimumLogLevel

(Facultatif) Le niveau minimum de messages de journal à télécharger. Ce niveau minimum s'applique uniquement si les journaux de ce composant utilisent un format JSON spécifique, que vous pouvez trouver dans le référentiel du [module de AWS IoT Greengrass journalisation](#) sur GitHub.

Choisissez parmi les niveaux de journalisation suivants, listés ici par ordre de niveau :

- DEBUG
- INFO
- WARN
- ERROR

Par défaut : INFO

diskSpaceLimit

(Facultatif) Taille totale maximale de tous les fichiers journaux pour ce composant, dans l'unité que vous spécifiez `diskSpaceLimitUnit`. Lorsque la taille totale des fichiers journaux de ce composant dépasse cette taille totale maximale, le logiciel AWS IoT Greengrass Core supprime les fichiers journaux les plus anciens de ce composant.

Ce paramètre est lié au paramètre de [limite de taille logarithmique](#) (`totalLogsSizeKB`) du composant du [noyau Greengrass](#). Le logiciel AWS IoT Greengrass Core utilise le minimum des deux valeurs comme taille maximale totale du journal pour ce composant.

diskSpaceLimitUnit

(Facultatif) L'unité pour le `diskSpaceLimit`. Sélectionnez parmi les options suivantes :

- KB— kilo-octets
- MB— méga-octets
- GB— giga-octets

Par défaut : KB

logFileDirectoryPath

(Facultatif) Le chemin d'accès au dossier contenant les fichiers journaux de ce composant.

Il n'est pas nécessaire de spécifier ce paramètre pour les composants Greengrass qui impriment en sortie standard (stdout) et en erreur standard (stderr).

Par défaut : `/greengrass/v2/logs`.

logFileRegex

(Facultatif) Expression régulière qui spécifie le format de nom de fichier journal utilisé par le composant ou l'application. Le composant du gestionnaire de journaux utilise cette expression régulière pour identifier les fichiers journaux dans le dossier situé à `logFileDirectoryPath`.

Il n'est pas nécessaire de spécifier ce paramètre pour les composants Greengrass qui impriment en sortie standard (stdout) et en erreur standard (stderr).

Si votre composant ou application fait pivoter les fichiers journaux, spécifiez une expression régulière correspondant aux noms des fichiers journaux pivotés. Par exemple, vous pouvez spécifier `hello_world\\\\w*.log` de télécharger les journaux d'une application Hello World. Le `\\\\w*` modèle correspond à zéro ou plusieurs caractères de mot, y compris les caractères alphanumériques et les traits de soulignement. Cette expression régulière fait correspondre les fichiers journaux avec et sans horodatage dans leur nom. Dans cet exemple, le gestionnaire de journaux télécharge les fichiers journaux suivants :

- `hello_world.log`— Le fichier journal le plus récent de l'application Hello World.
- `hello_world_2020_12_15_17_0.log`— Un ancien fichier journal de l'application Hello World.

Par défaut : `componentName\\\\w*.log`, où `componentName` est le nom du composant pour cette configuration de journal.

`deleteLogFileAfterCloudUpload`

(Facultatif) Vous pouvez supprimer un fichier journal une fois que le composant du gestionnaire de journaux a chargé les journaux dans CloudWatch Logs.

Par défaut : `false`

`multiLineStartPattern`

(Facultatif) Expression régulière qui identifie le moment où un message de journal sur une nouvelle ligne est un nouveau message de journal. Si l'expression régulière ne correspond pas à la nouvelle ligne, le composant du gestionnaire de journaux ajoute la nouvelle ligne au message de journal de la ligne précédente.

Par défaut, le composant du gestionnaire de journaux vérifie si la ligne commence par un caractère d'espace, tel qu'un onglet ou un espace. Si ce n'est pas le cas, le gestionnaire de journaux traite cette ligne comme un nouveau message de journal. Dans le cas contraire, il ajoute cette ligne au message du journal actuel. Ce comportement garantit que le composant du gestionnaire de journaux ne divise pas les messages qui s'étendent sur plusieurs lignes, tels que les traces de pile.

`periodicUploadIntervalSec`

(Facultatif) Période en secondes pendant laquelle le composant du gestionnaire de journaux vérifie la présence de nouveaux fichiers journaux à télécharger.

Par défaut : `300` (5 minutes)

Minimum : `0.000001` (1 microseconde)

Exemple Exemple : mise à jour de la fusion de configurations

L'exemple de configuration suivant indique de télécharger les journaux du système et les journaux des `com.example.HelloWorld` composants dans CloudWatch Logs.

```
{
  "logsUploaderConfiguration": {
    "systemLogsConfiguration": {
      "uploadToCloudWatch": "true",
      "minimumLogLevel": "INFO",
```



```

    "diskSpaceLimit": "10",
    "diskSpaceLimitUnit": "MB",
    "deleteLogFileAfterCloudUpload": "false"
  },
  "componentLogsConfiguration": [
    {
      "componentName": "com.example.HelloWorld",
      "minimumLogLevel": "INFO",
      "diskSpaceLimit": "20",
      "diskSpaceLimitUnit": "MB",
      "deleteLogFileAfterCloudUpload": "false"
    }
  ]
},
"periodicUploadIntervalSec": "300"
}

```

v2.0.x

logsUploaderConfiguration

(Facultatif) Configuration des journaux que le composant du gestionnaire de journaux télécharge. Cet objet contient les informations suivantes :

systemLogsConfiguration

(Facultatif) Configuration des journaux du système du logiciel AWS IoT Greengrass Core. Spécifiez cette configuration pour permettre au composant du gestionnaire de journaux de gérer les journaux système. Cet objet contient les informations suivantes :

uploadToCloudWatch

(Facultatif) Vous pouvez télécharger les journaux du système dans CloudWatch Logs.

Par défaut : `false`

minimumLogLevel

(Facultatif) Le niveau minimum de messages de journal à télécharger. Ce niveau minimum s'applique uniquement si vous configurez le [composant Greengrass noyau pour générer des journaux](#) au format JSON. Pour activer les journaux au format JSON, spécifiez JSON le paramètre de [format de journalisation](#) (`logging.format`).

Choisissez parmi les niveaux de journalisation suivants, listés ici par ordre de niveau :

- DEBUG
- INFO
- WARN
- ERROR

Par défaut : INFO

`diskSpaceLimit`

(Facultatif) Taille totale maximale des fichiers journaux du système Greengrass, dans l'unité que vous spécifiez. `diskSpaceLimitUnit` Lorsque la taille totale des fichiers journaux du système Greengrass dépasse cette taille totale maximale, le logiciel AWS IoT Greengrass Core supprime les plus anciens fichiers journaux du système Greengrass.

Ce paramètre est équivalent au paramètre de [limite de taille logarithmique](#) (`totalLogsSizeKB`) du composant du [noyau Greengrass](#). Le logiciel AWS IoT Greengrass Core utilise le minimum des deux valeurs comme taille maximale totale du journal du système Greengrass.

`diskSpaceLimitUnit`

(Facultatif) L'unité pour `diskSpaceLimit`. Sélectionnez parmi les options suivantes :

- KB— kilo-octets
- MB— méga-octets
- GB— giga-octets

Par défaut : KB

`deleteLogFileAfterCloudUpload`

(Facultatif) Vous pouvez supprimer un fichier journal une fois que le composant du gestionnaire de journaux a chargé les journaux dans CloudWatch Logs.

Par défaut : `false`

`componentLogsConfiguration`

(Facultatif) Liste des configurations de journal pour les composants du périphérique principal. Chaque configuration de cette liste définit la configuration du journal pour un composant ou une application. Le composant du gestionnaire de journaux télécharge les journaux de ces composants dans Logs. CloudWatch

Chaque objet contient les informations suivantes :

`componentName`

Nom du composant ou de l'application pour cette configuration de journal. Vous pouvez spécifier le nom d'un composant Greengrass ou une autre valeur pour identifier ce groupe de logs.

`minimumLogLevel`

(Facultatif) Le niveau minimum de messages de journal à télécharger. Ce niveau minimum s'applique uniquement si les journaux de ce composant utilisent un format JSON spécifique, que vous pouvez trouver dans le référentiel du [module de AWS IoT Greengrass journalisation](#) sur GitHub.

Choisissez parmi les niveaux de journalisation suivants, listés ici par ordre de niveau :

- DEBUG
- INFO
- WARN
- ERROR

Par défaut : INFO

`diskSpaceLimit`

(Facultatif) Taille totale maximale de tous les fichiers journaux pour ce composant, dans l'unité que vous spécifiez `diskSpaceLimitUnit`. Lorsque la taille totale des fichiers journaux de ce composant dépasse cette taille totale maximale, le logiciel AWS IoT Greengrass Core supprime les fichiers journaux les plus anciens de ce composant.

Ce paramètre est lié au paramètre de [limite de taille logarithmique](#) (`totalLogsSizeKB`) du composant du [noyau Greengrass](#). Le logiciel AWS IoT Greengrass Core utilise le minimum des deux valeurs comme taille maximale totale du journal pour ce composant.

`diskSpaceLimitUnit`

(Facultatif) L'unité pour `diskSpaceLimit`. Sélectionnez parmi les options suivantes :

- KB— kilo-octets
- MB— méga-octets
- GB— giga-octets

Par défaut : KB

logFileDirectoryPath

Le chemin d'accès au dossier contenant les fichiers journaux de ce composant.

Pour télécharger les journaux d'un composant Greengrass **/greengrass/v2/logs**, spécifiez-le et remplacez-le par votre dossier **/greengrass/v2** racine Greengrass.

logFileRegex

Expression régulière qui spécifie le format de nom de fichier journal utilisé par le composant ou l'application. Le composant du gestionnaire de journaux utilise cette expression régulière pour identifier les fichiers journaux dans le dossier situé à `logFileDirectoryPath`.

Pour télécharger les journaux d'un composant Greengrass, spécifiez une expression régulière correspondant aux noms des fichiers journaux pivotés. Par exemple, vous pouvez spécifier **com.example.HelloWorld\\w*.log** de télécharger les journaux d'un composant Hello World. Le `\\w*` modèle correspond à zéro ou plusieurs caractères de mot, y compris les caractères alphanumériques et les traits de soulignement. Cette expression régulière fait correspondre les fichiers journaux avec et sans horodatage dans leur nom. Dans cet exemple, le gestionnaire de journaux télécharge les fichiers journaux suivants :

- `com.example.HelloWorld.log`— Le fichier journal le plus récent pour le composant Hello World.
- `com.example.HelloWorld_2020_12_15_17_0.log`— Un ancien fichier journal pour le composant Hello World. Le noyau Greengrass ajoute un horodatage rotatif aux fichiers journaux.

deleteLogFileAfterCloudUpload

(Facultatif) Vous pouvez supprimer un fichier journal une fois que le composant du gestionnaire de journaux a chargé les journaux dans CloudWatch Logs.

Par défaut : false

multiLineStartPattern

(Facultatif) Expression régulière qui identifie le moment où un message de journal sur une nouvelle ligne est un nouveau message de journal. Si l'expression régulière ne

correspond pas à la nouvelle ligne, le composant du gestionnaire de journaux ajoute la nouvelle ligne au message de journal de la ligne précédente.

Par défaut, le composant du gestionnaire de journaux vérifie si la ligne commence par un caractère d'espace, tel qu'un onglet ou un espace. Si ce n'est pas le cas, le gestionnaire de journaux traite cette ligne comme un nouveau message de journal. Dans le cas contraire, il ajoute cette ligne au message du journal actuel. Ce comportement garantit que le composant du gestionnaire de journaux ne divise pas les messages qui s'étendent sur plusieurs lignes, tels que les traces de pile.

`periodicUploadIntervalSec`

(Facultatif) Période en secondes pendant laquelle le composant du gestionnaire de journaux vérifie la présence de nouveaux fichiers journaux à télécharger.

Par défaut : 300 (5 minutes)

Minimum : 0.000001 (1 microseconde)

Exemple Exemple : mise à jour de la fusion de configurations

L'exemple de configuration suivant indique de télécharger les journaux du système et les journaux des `com.example.HelloWorld` composants dans CloudWatch Logs.

```
{
  "logsUploaderConfiguration": {
    "systemLogsConfiguration": {
      "uploadToCloudWatch": "true",
      "minimumLogLevel": "INFO",
      "diskSpaceLimit": "10",
      "diskSpaceLimitUnit": "MB",
      "deleteLogFileAfterCloudUpload": "false"
    },
    "componentLogsConfiguration": [
      {
        "componentName": "com.example.HelloWorld",
        "minimumLogLevel": "INFO",
        "logFileDirectoryPath": "/greengrass/v2/logs",
        "logFileRegex": "com.example.HelloWorld\\w*.log",
        "diskSpaceLimit": "20",
        "diskSpaceLimitUnit": "MB",
```

```
        "deleteLogFileAfterCloudUpload": "false"
      }
    ]
  },
  "periodicUploadIntervalSec": "300"
}
```

Utilisation

Le composant du gestionnaire de journaux est chargé vers les groupes de journaux et les flux de journaux suivants.

2.1.0 and later

Nom du groupe de journaux

```
/aws/greengrass/componentType/region/componentName
```

Le nom du groupe de journaux utilise les variables suivantes :

- **componentType**— Le type du composant, qui peut être l'un des suivants :
 - **GreengrassSystemComponent**— Ce groupe de journaux inclut les journaux des composants du noyau et du plugin, qui s'exécutent dans la même JVM que le noyau Greengrass. Le composant fait partie du noyau de [Greengrass](#).
 - **UserComponent**— Ce groupe de journaux inclut les journaux des composants génériques, des composants Lambda et d'autres applications de l'appareil. Le composant ne fait pas partie du noyau de Greengrass.

Pour de plus amples informations, veuillez consulter [Types de composants](#).

- **region**— La AWS région utilisée par l'appareil principal.
- **componentName**— Le nom du composant. Pour les journaux système, cette valeur est `System`.

Nom du flux de log

```
/date/thing/thingName
```

Le nom du flux de journal utilise les variables suivantes :

- **date**— La date du journal, telle que 2020/12/15. Le composant du gestionnaire de journaux utilise le yyyy/MM/dd format.
- **thingName**— Le nom de l'appareil principal.

 Note

Si le nom d'un objet contient deux points (:), le gestionnaire de journaux remplace les deux points par un signe plus (+).

2.0.x

Nom du groupe de journaux

```
/aws/greengrass/componentType/region/componentName
```

Le nom du groupe de journaux utilise les variables suivantes :

- **componentType**— Le type du composant, qui peut être l'un des suivants :
 - **GreengrassSystemComponent**— Le composant fait partie du noyau de [Greengrass](#).
 - **UserComponent**— Le composant ne fait pas partie du noyau de Greengrass. Le gestionnaire de journaux utilise ce type pour les composants Greengrass et les autres applications de l'appareil.
- **region**— La AWS région utilisée par l'appareil principal.
- **componentName**— Le nom du composant. Pour les journaux système, cette valeur est `System`.

Nom du flux de log

```
/date/deploymentTargets/thingName
```

Le nom du flux de journal utilise les variables suivantes :

- **date**— La date du journal, telle que 2020/12/15. Le composant du gestionnaire de journaux utilise le yyyy/MM/dd format.
- **deploymentTargets**— Les objets dont les déploiements incluent le composant. Le composant du gestionnaire de journaux sépare chaque cible par une barre oblique. Si le composant s'exécute sur le périphérique principal à la suite d'un déploiement local, cette valeur est `LOCAL_DEPLOYMENT`.

Prenons un exemple où vous avez un périphérique principal nommé `MyGreengrassCore` et où le périphérique principal a deux déploiements :

- Un déploiement qui cible le périphérique principal, `MyGreengrassCore`.
- Un déploiement qui cible un groupe d'objets nommé `MyGreengrassCoreGroup`, qui contient le périphérique principal.

Les `deploymentTargets` pour cet appareil principal sont `thing/MyGreengrassCore/thinggroup/MyGreengrassCoreGroup`.

- `thingName`— Le nom de l'appareil principal.

Formats pour les entrées du journal.

Le noyau Greengrass écrit les fichiers journaux au format chaîne ou JSON. Pour les journaux système, vous pouvez contrôler le format en définissant le format champ de l'`loggingentrée`. Vous pouvez trouver l'`loggingentrée` dans le fichier de configuration du composant Greengrass nucleus. Pour plus d'informations, consultez la section Configuration du [noyau de Greengrass](#).

Le format de texte est de forme libre et accepte n'importe quelle chaîne. Le message de service d'état du parc suivant est un exemple de journalisation au format chaîne :

```
2023-03-26T18:18:27.271Z [INFO] (pool-1-thread-2)
com.aws.greengrass.status.FleetStatusService: fss-status-update-published.
Status update published to FSS. {trigger=CADENCE, serviceName=FleetStatusService,
currentState=RUNNING}
```

Vous devez utiliser le format JSON si vous souhaitez afficher les journaux à l'aide de la commande [Greengrass CLI logs](#) ou interagir avec les journaux par programmation. L'exemple suivant décrit la forme JSON :

```
{
  "loggerName": <string>,
  "level": <"DEBUG" | "INFO" | "ERROR" | "TRACE" | "WARN">,
  "eventType": <string, optional>,
  "cause": <string, optional>,
  "contexts": {},
  "thread": <string>,
  "message": <string>,
  "timestamp": <epoch time> # Needs to be epoch time
```



```
}
```

Pour contrôler la sortie des journaux de votre composant, vous pouvez utiliser l'option `minimumLogLevel` de configuration. Pour utiliser cette option, votre composant doit écrire ses entrées de journal au format JSON. Vous devez utiliser le même format que le fichier journal du système.

Fichier journal local

Ce composant utilise le même fichier journal que le composant [Greengrass nucleus](#).

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

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

Pour consulter les journaux de ce composant

- Exécutez la commande suivante sur le périphérique principal pour afficher le fichier journal de ce composant en temps réel. Remplacez `/greengrass/v2` ou `C:\greengrass\v2` par le chemin d'accès au dossier AWS IoT Greengrass racine.

Linux

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

Windows (PowerShell)

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

Journal des modifications

Le tableau suivant décrit les modifications apportées à chaque version du composant.

Version	Modifications
2.3.11	Corrections de bugs et améliorations • Résout un problème en raison duquel la configuration d'exécution de Log Manager augmentait indéfiniment en raison des informations périmées des fichiers journaux téléchargés.
2.3.10	Nouvelles fonctionnalités • Ajoute une nouvelle clé de configuration (<code>updateToTlogIntervalSec</code>) pour contrôler la fréquence à laquelle les détails des événements de téléchargement du journal sont conservés dans le journal des transactions local (<code>config.tlog</code>). Corrections de bugs et améliorations • Améliore le gestionnaire de journaux pour actualiser le client Cloudwatch en cas d'erreur de connexion au socket.
2.3.9	Version mise à jour pour la version 2.14.0 de Greengrass Nucleus.
2.3.8	Version mise à jour pour la version 2.13.0 de Greengrass Nucleus.
2.3.7	Version mise à jour pour la version 2.12.0 de Greengrass Nucleus.
2.3.6	Corrections de bugs et améliorations • Ajuste les niveaux de journalisation pour certaines erreurs.
2.3.5	Améliorations Améliore la vitesse de téléchargement des journaux. Version mise à jour pour la version 2.11.0 de Greengrass Nucleus.
2.3.4	Corrections de bugs et améliorations • Permet de définir le <code>periodicUploadIntervalSec</code> paramètre sur des valeurs fractionnaires. Le minimum est de 1 microseconde. • Résout un problème en raison duquel le gestionnaire de journaux ne respectait pas les <code>CloudWatch putLogEvents</code> limites.

Version	Modifications
2.3.3	Version mise à jour pour la version 2.10.0 de Greengrass Nucleus.
2.3.2	Corrections de bugs et améliorations • Améliore la gestion de l'espace afin que les fichiers journaux ne soient pas supprimés avant leur téléchargement. • Résout les problèmes liés à la gestion du cache. • Corrections de bogues et améliorations mineures supplémentaires.
2.3.1	Corrections de bugs et améliorations • Résout un problème selon lequel les groupes de fichiers cibles contenant plusieurs fichiers journaux actifs téléchargeaient des entrées dupliquées vers CloudWatch. • Corrections de bogues et améliorations mineures supplémentaires.
2.3.0	 Note Nous vous recommandons de passer à Greengrass nucleus 2.9.1 lors de la mise à niveau vers le gestionnaire de journaux 2.3.0. Nouvelles fonctionnalités Réduit les délais de journalisation en traitant et en téléchargeant directement les fichiers journaux actifs au lieu d'attendre la rotation des nouveaux fichiers. Corrections de bugs et améliorations • Améliore la prise en charge de la rotation des journaux lors de la rotation de fichiers portant un nom unique. • Corrections de bogues et améliorations mineures supplémentaires.
2.2.8	Version mise à jour pour la version 2.9.0 de Greengrass Nucleus.
2.2.7	Version mise à jour pour la version 2.8.0 de Greengrass Nucleus.

Version	Modifications
2.2.6	Version mise à jour pour la version 2.7.0 de Greengrass Nucleus.
2.2.5	Version mise à jour pour la version 2.6.0 de Greengrass Nucleus.
2.2.4	Corrections de bugs et améliorations • Améliore la stabilité lors de la gestion de configurations non valides. • Corrections et améliorations mineures supplémentaires.
2.2.3	Corrections de bugs et améliorations • Améliore la stabilité dans certains scénarios où le composant redémarre ou rencontre des erreurs. • Résout les problèmes liés à l'échec du téléchargement de messages et de fichiers journaux volumineux dans certains scénarios. • Résout les problèmes liés à la façon dont ce composant gère les mises à jour de réinitialisation de configuration. • Résout un problème où une valeur null <code>diskSpaceLimit</code> de configuration empêchait le déploiement du composant.
2.2.2	Corrections de bugs et améliorations • Ajoute la prise en charge des messages de journal d'une taille supérieure à 256 kilo-octets. Le composant du gestionnaire de journaux divise ces messages de journal volumineux en plusieurs messages portant le même horodatage des événements de journal.
2.2.1	Version mise à jour pour la version 2.5.0 de Greengrass Nucleus.
2.2.0	Nouvelle fonctionnalité • Ajoute le paramètre <code>componentLogsConfigurationMap</code> de configuration pour prendre en charge un format de carte pour les configurations du journal des composants. Chaque <code>componentName</code> objet de la carte définit la configuration du journal pour un composant ou une application.
2.1.3	Version mise à jour pour la version 2.4.0 de Greengrass Nucleus.

Version	Modifications
2.1.2	Version mise à jour pour la version 2.3.0 de Greengrass Nucleus.
2.1.1	Corrections de bugs et améliorations • Résout un problème en raison duquel la configuration du journal système n'était pas mise à jour dans certains cas.
2.1.0	Corrections de bugs et améliorations • Utilisez des valeurs par défaut pour <code>logFileDirectoryPath</code> et <code>logFileRegex</code> qui fonctionnent pour les composants Greengrass qui impriment en sortie standard (stdout) et en erreur standard (stderr). • Acheminez correctement le trafic via un proxy réseau configuré lors du téléchargement des CloudWatch journaux vers Logs. • Gérez correctement les caractères deux-points (:) dans les noms des flux de log. CloudWatch Les noms des flux de journaux ne prennent pas en charge les deux-points. • Simplifiez les noms des flux de journaux en supprimant les noms de groupes d'objets du flux de journaux. • Supprimez un message du journal des erreurs qui s'imprime lors d'un comportement normal.
2.0.x	Première version.

Composants d'apprentissage automatique

AWS IoT Greengrass fournit les composants d'apprentissage automatique suivants que vous pouvez déployer sur des appareils pris en charge pour [effectuer des inférences d'apprentissage automatique](#) à l'aide de modèles entraînés par Amazon SageMaker AI ou de vos propres modèles préentraînés stockés dans Amazon S3.

AWS fournit les catégories suivantes de composants d'apprentissage automatique :

- Composant du modèle : contient des modèles d'apprentissage automatique sous forme d'artefacts Greengrass.

- Composant d'exécution : contient le script qui installe le framework d'apprentissage automatique et ses dépendances sur le périphérique principal de Greengrass.
- Composant d'inférence : contient le code d'inférence et inclut les dépendances des composants pour installer le framework d'apprentissage automatique et télécharger des modèles d'apprentissage automatique préentraînés.

Vous pouvez utiliser l'exemple de code d'inférence et les modèles préentraînés dans les composants d'apprentissage automatique AWS fournis pour effectuer la classification d'images et la détection d'objets à l'aide de DLR et Lite. TensorFlow Pour effectuer une inférence d'apprentissage automatique personnalisée avec vos propres modèles stockés dans Amazon S3, ou pour utiliser un autre framework d'apprentissage automatique, vous pouvez utiliser les recettes de ces composants publics comme modèles afin de créer des composants d'apprentissage automatique personnalisés. Pour de plus amples informations, veuillez consulter [Personnalisez vos composants d'apprentissage automatique](#).

AWS IoT Greengrass inclut également un composant AWS fourni pour gérer l'installation et le cycle de vie de l'agent SageMaker AI Edge Manager sur les appareils principaux de Greengrass. Avec SageMaker AI Edge Manager, vous pouvez utiliser les modèles compilés par Amazon SageMaker AI Neo directement sur votre appareil principal. Pour de plus amples informations, veuillez consulter [Utiliser Amazon SageMaker AI Edge Manager sur les appareils principaux de Greengrass](#).

Le tableau suivant répertorie les composants d'apprentissage automatique disponibles dans AWS IoT Greengrass.

Note

Plusieurs composants AWS fournis dépendent de versions mineures spécifiques du noyau Greengrass. En raison de cette dépendance, vous devez mettre à jour ces composants lorsque vous mettez à jour le noyau Greengrass vers une nouvelle version mineure. Pour plus d'informations sur les versions spécifiques du noyau dont dépend chaque composant, consultez la rubrique correspondante sur les composants. Pour plus d'informations sur la mise à jour du noyau, consultez [Mettre à jour le logiciel AWS IoT Greengrass principal \(OTA\)](#).

Lorsqu'un composant possède un type de composant à la fois générique et Lambda, la version actuelle du composant est le type générique et une version précédente du composant est le type Lambda.

Composant	Description	Type de composant	Systèmes d'exploitation pris en charge	Open source
SageMaker Gestionnaire AI Edge	Déploie l'agent Amazon SageMaker AI Edge Manager sur l'appareil principal de Greengrass.	Générique	Linux, Windows	Non
Classification des images DLR	Composant d'inférence qui utilise le magasin de modèles de classification d'images DLR et le composant d'exécution DLR comme dépendances pour installer le DLR, télécharger des exemples de modèles de classification d'images et effectuer une inférence de classification d'images sur	Générique	Linux, Windows	Non

Composant	Description	Type de composant	Systèmes d'exploitation pris en charge	Open source
	les appareils pris en charge.			
Détection d'objets DLR	Composant d'inférence qui utilise le model store de détection d'objets DLR et le composant d'exécution DLR comme dépendances pour installer le DLR, télécharger des exemples de modèles de détection d'objets et effectuer une inférence de détection d'objets sur les appareils pris en charge.	Générique	Linux, Windows	Non

Composant	Description	Type de composant	Systèmes d'exploitation pris en charge	Open source
magasin de modèles de classification d'images DLR	Composant de modèle contenant des exemples de ResNet 50 modèles de classification d'images sous forme d'artefacts Greengrass.	Générique	Linux, Windows	Non
Model Store dédié à la détection d'objets DLR	Composant de modèle contenant des exemples de modèles de détection YOLOv3 d'objets sous forme d'artefacts Greengrass.	Générique	Linux, Windows	Non

Composant	Description	Type de composant	Systèmes d'exploitation pris en charge	Open source
Temps d'exécution du DLR	Composant d'exécution contenant un script d'installation utilisé pour installer le DLR et ses dépendances sur le périphérique principal de Greengrass.	Générique	Linux, Windows	Non

Composant	Description	Type de composant	Systèmes d'exploitation pris en charge	Open source
TensorFlow Classification d'images Lite	Composant d'inférence qui utilise le magasin de modèles de classification d'images TensorFlow Lite et le composant d'exécution TensorFlow Lite comme dépendances pour installer TensorFlow Lite, télécharger des exemples de modèles de classification d'images et effectuer une inférence de classification d'images sur les appareils pris en charge.	Générique	Linux, Windows	Non

Composant	Description	Type de composant	Systèmes d'exploitation pris en charge	Open source
TensorFlow Détection d'objets allégée	Composant d'inférence qui utilise le magasin de modèles de détection d'objets TensorFlow Lite et le composant d'exécution TensorFlow Lite comme dépendances pour installer TensorFlow Lite, télécharger des exemples de modèles de détection d'objets et effectuer une inférence de détection d'objets sur les appareils pris en charge.	Générique	Linux, Windows	Non

Composant	Description	Type de composant	Systèmes d'exploitation pris en charge	Open source
TensorFlow Boutique de modèles de classification d'images Lite	Composant de modèle contenant un exemple de modèle MobileNet v1 en tant qu'artefact Greengrass.	Générique	Linux, Windows	Non
TensorFlow Boutique de modèles de détection d'objets Lite	Composant de modèle contenant un exemple de MobileNet modèle de détection par injection unique (SSD) sous forme d'artefact Greengrass.	Générique	Linux, Windows	Non

Composant	Description	Type de composant	Systèmes d'exploitation pris en charge	Open source
TensorFlow Temps d'exécution allégé	Composant d'exécution contenant un script d'installation utilisé pour installer TensorFlow Lite et ses dépendances sur le périphérique principal de Greengrass.	Générique	Linux, Windows	Non

SageMaker Gestionnaire AI Edge

Important

SageMaker AI Edge Manager a été abandonné le 26 avril 2024. Pour plus d'informations sur la poursuite du déploiement de vos modèles sur des appareils Edge, consultez [SageMaker AI Edge Manager end of life](#).

Le composant Amazon SageMaker AI Edge Manager

(`aws.greengrass.SageMakerEdgeManager`) installe le binaire de l'agent SageMaker AI Edge Manager.

SageMaker AI Edge Manager permet de gérer les modèles pour les appareils de périphérie afin que vous puissiez optimiser, sécuriser, surveiller et gérer les modèles d'apprentissage automatique sur des flottes d'appareils de périphérie. Le composant SageMaker AI Edge Manager installe et gère le cycle de vie de l'agent SageMaker AI Edge Manager sur votre appareil principal. Vous pouvez également utiliser SageMaker AI Edge Manager pour emballer et utiliser des modèles

compilés par SageMaker AI Neo en tant que composants de modèle sur les appareils principaux de Greengrass. Pour plus d'informations sur l'utilisation de l'agent SageMaker AI Edge Manager sur votre appareil principal, consultez [Utiliser Amazon SageMaker AI Edge Manager sur les appareils principaux de Greengrass](#).

SageMaker Le composant AI Edge Manager v1.3.x installe le binaire de l'agent Edge Manager v1.20220822.836f3023. Pour plus d'informations sur les versions binaires de l'agent Edge Manager, consultez la section [Agent Edge Manager](#).

Note

Le composant SageMaker AI Edge Manager n'est disponible que dans les versions suivantes Régions AWS :

- USA Est (Ohio)
- USA Est (Virginie du Nord)
- USA Ouest (Oregon)
- UE (Francfort)
- UE (Irlande)
- Asie-Pacifique (Tokyo)

Rubriques

- [Versions](#)
- [Type](#)
- [Système d'exploitation](#)
- [Prérequis](#)
- [Dépendances](#)
- [Configuration](#)
- [Fichier journal local](#)
- [Journal des modifications](#)

Versions

Les versions de ce composant sont les suivantes :

- 1.3.x
- 1,2. x
- 1,1x
- 1,0 x

Type

Ce composant est un composant générique (`aws.greengrass.generic`). Le [noyau Greengrass](#) exécute les scripts de cycle de vie du composant.

Pour de plus amples informations, veuillez consulter [Types de composants](#).

Système d'exploitation

Ce composant peut être installé sur les appareils principaux qui exécutent les systèmes d'exploitation suivants :

- Linux
- Windows

Prérequis

Ce composant répond aux exigences suivantes :

- Un appareil Greengrass principal fonctionnant sous Amazon Linux 2, une plate-forme Linux basée sur Debian (x86_64 ou Armv8) ou Windows (x86_64). Si vous n'en avez pas, veuillez consulter [Tutoriel : Débuter avec AWS IoT Greengrass V2](#).
- [Python](#) 3.6 ou version ultérieure, y compris `pip` pour votre version de Python, installé sur votre appareil principal.
- Le [rôle d'appareil Greengrass](#) est configuré comme suit :
 - Une relation de confiance qui permet `credentials.iot.amazonaws.com` et permet `sagemaker.amazonaws.com` d'assumer le rôle, comme le montre l'exemple de politique IAM suivant.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
```



```

    "Effect": "Allow",
    "Principal": {
      "Service": "credentials.iot.amazonaws.com"
    },
    "Action": "sts:AssumeRole"
  },
  {
    "Effect": "Allow",
    "Principal": {
      "Service": "sagemaker.amazonaws.com"
    },
    "Action": "sts:AssumeRole"
  }
]
}

```

- La politique gérée par [AmazonSageMakerEdgeDeviceFleetPolicy](#) IAM.
- L'`s3:PutObject` action, comme illustré dans l'exemple de politique IAM suivant.

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "s3:PutObject"
      ],
      "Resource": [
        "*"
      ],
      "Effect": "Allow"
    }
  ]
}

```

- Un bucket Amazon S3 créé en même temps Compte AWS et en même temps Région AWS que votre appareil principal Greengrass. SageMaker AI Edge Manager nécessite un compartiment S3 pour créer un parc d'appareils Edge et pour stocker des exemples de données provenant de l'exécution d'inférences sur votre appareil. Pour plus d'informations sur la création de compartiments S3, consultez [Getting started with Amazon S3](#).
- Un parc d'appareils SageMaker AI Edge qui utilise le même alias de AWS IoT rôle que votre appareil principal Greengrass. Pour de plus amples informations, veuillez consulter [Créez un parc d'appareils de pointe](#).

- Votre appareil principal Greengrass est enregistré en tant qu'appareil Edge dans votre parc d'appareils SageMaker AI Edge. Le nom de l'appareil Edge doit correspondre au nom de l'AWS IoT objet de votre appareil principal. Pour de plus amples informations, veuillez consulter [Enregistrez votre appareil Greengrass Core](#).

Endpoints et ports

Ce composant doit être capable d'effectuer des demandes sortantes vers les points de terminaison et les ports suivants, en plus des points de terminaison et des ports requis pour le fonctionnement de base. Pour de plus amples informations, veuillez consulter [Autoriser le trafic des appareils via un proxy ou un pare-feu](#).

Endpoint	Port	Obligatoire	Description
edge.sagemaker. <i>region</i> .amazonaws.com	443	Oui	Vérifiez l'état d'enregistrement de l'appareil et envoyez des métriques à l'SageMaker IA.
*.s3.amazonaws.com	443	Oui	Téléchargez les données de capture dans le compartiment S3 que vous spécifiez.

Endpoint	Port	Obligatoire	Description
			Vous pouvez le * remplacer par le nom de chaque compartiment dans lequel vous chargez des données.

Dépendances

Lorsque vous déployez un composant, il déploie AWS IoT Greengrass également des versions compatibles de ses dépendances. Cela signifie que vous devez satisfaire aux exigences relatives au composant et à toutes ses dépendances pour réussir le déploiement du composant. Cette section répertorie les dépendances des [versions publiées](#) de ce composant et les contraintes de version sémantiques qui définissent les versions des composants pour chaque dépendance. Vous pouvez également consulter les dépendances de chaque version du composant dans la [AWS IoT Greengrass console](#). Sur la page de détails du composant, recherchez la liste des dépendances.

1.3.5 and 1.3.6

Le tableau suivant répertorie les dépendances pour les versions 1.3.5 et 1.3.6 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,13.0	Flexible
Service d'échange de jetons	>=0,0.0	Stricte

1.3.4

Le tableau suivant répertorie les dépendances pour la version 1.3.4 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,12.0	Flexible
Service d'échange de jetons	>=0,0.0	Stricte

1.3.3

Le tableau suivant répertorie les dépendances pour la version 1.3.3 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,11.0	Flexible
Service d'échange de jetons	>=0,0.0	Stricte

1.3.2

Le tableau suivant répertorie les dépendances pour la version 1.3.2 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0,0 <2,1,0	Flexible
Service d'échange de jetons	>=0,0.0	Stricte

1.3.1

Le tableau suivant répertorie les dépendances pour la version 1.3.1 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,9.0	Flexible
Service d'échange de jetons	>=0,0.0	Stricte

1.1.1 - 1.3.0

Le tableau suivant répertorie les dépendances pour les versions 1.1.1 à 1.3.0 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,0.0 < 2,8.0$	Flexible
Service d'échange de jetons	$\geq 0,0.0$	Stricte

1.1.0

Le tableau suivant répertorie les dépendances pour la version 1.1.0 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,0.0 < 2,6.0$	Flexible
Service d'échange de jetons	$\geq 0,0.0$	Stricte

1.0.3

Le tableau suivant répertorie les dépendances pour la version 1.0.3 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,0.0 < 2,5.0$	Flexible
Service d'échange de jetons	$\geq 0,0.0$	Stricte

1.0.1 and 1.0.2

Le tableau suivant répertorie les dépendances pour les versions 1.0.1 et 1.0.2 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,0.0 < 2,4.0$	Flexible

Dépendance	Versions compatibles	Type de dépendance
Service d'échange de jetons	>=0,0.0	Stricte

1.0.0

Le tableau suivant répertorie les dépendances pour la version 1.0.0 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,3.0	Flexible
Service d'échange de jetons	>=0,0.0	Stricte

Pour plus d'informations sur les dépendances des composants, consultez la [référence de la recette des composants](#).

Configuration

Ce composant fournit les paramètres de configuration suivants que vous pouvez personnaliser lorsque vous déployez le composant.

Note

Cette section décrit les paramètres de configuration que vous définissez dans le composant. Pour plus d'informations sur la configuration correspondante d' SageMaker AI Edge Manager, consultez la section [Edge Manager Agent](#) dans le manuel Amazon SageMaker AI Developer Guide.

DeviceFleetName

Le nom du parc d'appareils SageMaker AI Edge Manager qui contient votre appareil principal Greengrass.

Vous devez spécifier une valeur pour ce paramètre dans la mise à jour de configuration lorsque vous déployez ce composant.

BucketName

Nom du compartiment S3 dans lequel vous chargez les données d'inférence capturées. Le nom du bucket doit contenir la chaîne `sagemaker`.

Si vous définissez `CaptureDataDestinationCloud`, ou si vous définissez `CaptureDataPeriodicUpload` sur `true`, vous devez spécifier une valeur pour ce paramètre dans la mise à jour de configuration lorsque vous déployez ce composant.

Note

La capture de données est une fonctionnalité d' `SageMaker` intelligence artificielle que vous utilisez pour télécharger des entrées d'inférence, des résultats d'inférence et des données d'inférence supplémentaires dans un compartiment S3 ou un répertoire local en vue d'une analyse future. Pour plus d'informations sur l'utilisation des données de capture avec `SageMaker AI Edge Manager`, consultez [Manage Model](#) dans le manuel `Amazon SageMaker AI Developer Guide`.

CaptureDataBatchSize

(Facultatif) Taille d'un lot de demandes de données de capture traitées par l'agent. Cette valeur doit être inférieure à la taille de la mémoire tampon que vous spécifiez dans `CaptureDataBufferSize`. Nous vous recommandons de ne pas dépasser la moitié de la taille de la mémoire tampon.

L'agent gère un lot de demandes lorsque le nombre de demandes dans la mémoire tampon correspond à ce `CaptureDataBatchSize` nombre, ou lorsque l'`CaptureDataPushPeriodSeconds` intervalle est écoulé, selon la première éventualité.

Par défaut : 10

CaptureDataBufferSize

(Facultatif) Le nombre maximum de demandes de données de capture stockées dans la mémoire tampon.

Par défaut : 30

CaptureDataDestination

(Facultatif) La destination où vous stockez les données capturées. Ce paramètre peut présenter les valeurs suivantes :

- **Cloud**—Télécharge les données capturées dans le compartiment S3 que vous spécifiez.
`BucketName`
- **Disk**: écrit les données capturées dans le répertoire de travail du composant.

Si vous le spécifiez `Disk`, vous pouvez également choisir de télécharger régulièrement les données capturées dans votre compartiment S3 en réglant `CaptureDataPeriodicUpload` sur `true`.

Par défaut : `Cloud`

CaptureDataPeriodicUpload

(Facultatif) Valeur de chaîne qui indique s'il faut télécharger régulièrement les données capturées. Les valeurs prises en charge sont `true` et `false`.

Définissez ce paramètre sur `true` si vous le définissez `CaptureDataDestination` sur `Disk`, et vous souhaitez également que l'agent télécharge régulièrement les données capturées dans votre compartiment S3.

Par défaut : `false`

CaptureDataPeriodicUploadPeriodSeconds

(Facultatif) Intervalle en secondes pendant lequel l'agent SageMaker AI Edge Manager télécharge les données capturées dans le compartiment S3. Utilisez ce paramètre si vous définissez `CaptureDataPeriodicUpload` sur `true`.

Par défaut : 8

CaptureDataPushPeriodSeconds

(Facultatif) Intervalle en secondes pendant lequel l'agent SageMaker AI Edge Manager traite un lot de demandes de données de capture provenant de la mémoire tampon.

L'agent gère un lot de demandes lorsque le nombre de demandes dans la mémoire tampon correspond à ce `CaptureDataBatchSize` nombre, ou lorsque l'`CaptureDataPushPeriodSeconds` intervalle est écoulé, selon la première éventualité.

Par défaut : 4

CaptureDataBase64EmbedLimit

(Facultatif) Taille maximale en octets des données capturées que l'agent SageMaker AI Edge Manager télécharge.

Par défaut : 3072

FolderPrefix

(Facultatif) Nom du dossier dans lequel l'agent écrit les données capturées. Si vous définissez CaptureDataDestination cette Disk option, l'agent crée le dossier dans le répertoire spécifié par CaptureDataDiskPath. Si vous avez défini CaptureDataDestination ou défini sur true, CaptureDataPeriodicUpload l'agent crée le dossier dans votre compartiment S3. Cloud

Par défaut : sme-capture

CaptureDataDiskPath

Cette fonctionnalité est disponible dans les versions v1.1.0 et ultérieures du composant SageMaker AI Edge Manager.

(Facultatif) Le chemin d'accès au dossier dans lequel l'agent crée le dossier de données capturées. Si vous définissez cette CaptureDataDestination option Disk, l'agent crée le dossier de données capturées dans ce répertoire. Si vous ne spécifiez pas cette valeur, l'agent crée le dossier de données capturées dans le répertoire de travail du composant. Utilisez le FolderPrefix paramètre pour spécifier le nom du dossier de données capturé.

Par défaut : `/greengrass/v2/work/aws.greengrass.SageMakerEdgeManager/capture`

LocalDataRootPath

Cette fonctionnalité est disponible dans les versions v1.2.0 et ultérieures du composant SageMaker AI Edge Manager.

(Facultatif) Le chemin où ce composant stocke les données suivantes sur le périphérique principal :

- La base de données locale pour les données d'exécution lorsque vous définissez DbEnable sur true.
- SageMaker Modèles compilés par AI Neo que ce composant télécharge automatiquement lorsque vous le configurez DeploymentEnable. true

Par défaut : `/greengrass/v2/work/aws.greengrass.SageMakerEdgeManager`

DbEnable

(Facultatif) Vous pouvez activer ce composant pour stocker les données d'exécution dans une base de données locale afin de préserver les données, en cas de panne du composant ou de coupure d'alimentation du périphérique.

Cette base de données nécessite 5 Mo de stockage sur le système de fichiers de l'appareil principal.

Par défaut : `false`

DeploymentEnable

Cette fonctionnalité est disponible dans les versions v1.2.0 et ultérieures du composant SageMaker AI Edge Manager.

(Facultatif) Vous pouvez activer ce composant pour récupérer automatiquement les modèles compilés par SageMaker AI Neo à partir desquels vous les téléchargez sur Amazon S3. Après avoir chargé un nouveau modèle sur Amazon S3, utilisez SageMaker AI Studio ou l'API SageMaker AI pour déployer le nouveau modèle sur cet appareil principal. Lorsque vous activez cette fonctionnalité, vous pouvez déployer de nouveaux modèles sur les appareils principaux sans avoir à créer de AWS IoT Greengrass déploiement.

Important

Pour utiliser cette fonctionnalité, vous devez `DbEnable` définir sur `true`. Cette fonctionnalité utilise la base de données locale pour suivre les modèles qu'elle extrait du AWS Cloud.

Par défaut : `false`

DeploymentPollInterval

Cette fonctionnalité est disponible dans les versions v1.2.0 et ultérieures du composant SageMaker AI Edge Manager.

(Facultatif) Durée (en minutes) entre laquelle ce composant vérifie la présence de nouveaux modèles à télécharger. Cette option s'applique lorsque vous définissez `DeploymentEnable` sur `true`.

Par défaut : 1440 (1 jour)

DLRBackendOptions

Cette fonctionnalité est disponible dans les versions v1.2.0 et ultérieures du composant SageMaker AI Edge Manager.

(Facultatif) Les indicateurs d'exécution du DLR à définir dans le moteur d'exécution du DLR utilisé par ce composant. Vous pouvez définir le drapeau suivant :

- `TVM_TENSORRT_CACHE_DIR`— Active la mise en cache du modèle TensorRT. Spécifiez un chemin absolu vers un dossier existant doté d'autorisations de lecture/écriture.
- `TVM_TENSORRT_CACHE_DISK_SIZE_MB`— Attribue la limite supérieure du dossier de cache du modèle TensorRT. Lorsque la taille du répertoire dépasse cette limite, les moteurs mis en cache les moins utilisés sont supprimés. La valeur par défaut est de 512 Mo.

Par exemple, vous pouvez définir ce paramètre sur la valeur suivante pour activer la mise en cache du modèle TensorRT et limiter la taille du cache à 800 Mo.

```
TVM_TENSORRT_CACHE_DIR=/data/secured_folder/trt/cache;  
TVM_TENSORRT_CACHE_DISK_SIZE_MB=800
```

SagemakerEdgeLogVerbose

(Facultatif) Valeur de chaîne qui indique s'il faut activer la journalisation du débogage. Les valeurs prises en charge sont `true` et `false`.

Par défaut : `false`

UnixSocketName

(Facultatif) Emplacement du descripteur de fichier de socket SageMaker AI Edge Manager sur le périphérique principal.

Par défaut : `/tmp/aws.greengrass.SageMakerEdgeManager.sock`

Exemple Exemple : mise à jour de la fusion de configurations

L'exemple de configuration suivant indique que le périphérique principal fait partie du *MyEdgeDeviceFleet* et que l'agent écrit les données de capture à la fois sur le périphérique et dans un compartiment S3. Cette configuration permet également la journalisation du débogage.

```
{
```

```
"DeviceFleetName": "MyEdgeDeviceFleet",  
"BucketName": "amzn-s3-demo-bucket",  
"CaptureDataDestination": "Disk",  
"CaptureDataPeriodicUpload": "true",  
"SagemakerEdgeLogVerbose": "true"  
}
```

Fichier journal local

Ce composant utilise le fichier journal suivant.

Linux

```
/greengrass/v2/logs/aws.greengrass.SageMakerEdgeManager.log
```

Windows

```
C:\greengrass\v2\logs\aws.greengrass.SageMakerEdgeManager.log
```

Pour consulter les journaux de ce composant

- Exécutez la commande suivante sur le périphérique principal pour afficher le fichier journal de ce composant en temps réel. Remplacez `/greengrass/v2` ou `C:\greengrass\v2` par le chemin d'accès au dossier AWS IoT Greengrass racine.

Linux

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

Windows (PowerShell)

```
Get-Content C:\greengrass\v2\logs\aws.greengrass.SageMakerEdgeManager.log -Tail  
10 -Wait
```

Journal des modifications

Le tableau suivant décrit les modifications apportées à chaque version du composant.

Version	Modifications
1.3.6	Version mise à jour pour la version 2.12.5 de Greengrass Nucleus.
1.3.5	Version mise à jour pour la version 2.12.0 de Greengrass Nucleus.
1.3.4	Version mise à jour pour la version 2.11.0 de Greengrass Nucleus.
1.3.3	Version mise à jour pour la version 2.10.0 de Greengrass Nucleus.
1.3.2	Version mise à jour pour la version 2.9.0 de Greengrass Nucleus.
1.3.1	Version mise à jour pour la version 2.8.0 de Greengrass Nucleus.
1.3.0	Nouvelles fonctionnalités • Ajoute la prise en charge de la gestion de la taille du disque de cache TensorRT. • Ajoute l'<code>TVM_TENSORRT_CACHE_DISK_SIZE_MB</code> indicateur facultatif au paramètre DLRBackend Options pour définir la limite de taille des modèles mis en cache sur le disque. Améliorations • Améliore la simultanéité des prédictions. Cela permet de mieux utiliser les moteurs d'accélérateur des appareils, tels que GPUs.
1.2.0	Nouvelles fonctionnalités • Prend en charge ce composant afin de récupérer automatiquement les modèles compilés par SageMaker AI Neo que vous chargez sur Amazon S3. Lorsque vous activez cette fonctionnalité, vous pouvez déployer de nouveaux modèles sur les appareils principaux sans avoir à créer de AWS IoT Greengrass déploiement. • Ajoute la prise en charge d'une base de données de sauvegarde que ce composant utilise pour préserver les données d'exécution, en cas de panne du composant ou de panne d'alimentation du périphérique. • Permet de configurer les indicateurs d'exécution du DLR lorsque vous configurez ce composant.
1.1.1	Version mise à jour pour la version 2.7.0 de Greengrass Nucleus.

Version	Modifications
1.1.0	Nouvelles fonctionnalités • Ajoute la prise en charge des appareils principaux de Greengrass exécutant Amazon Linux 2. • Ajoute le nouveau paramètre <code>CaptureDataDiskPath</code> de configuration. Vous pouvez utiliser ce paramètre pour spécifier le chemin du dossier de données capturé sur votre appareil. Corrections de bugs et améliorations • Version mise à jour pour la version 2.5.0 de Greengrass Nucleus.
1.0.3	Version mise à jour pour la version 2.4.0 de Greengrass Nucleus.
1.0.2	Corrections de bugs et améliorations Met à jour le script d'installation dans le cycle de vie du composant. Python 3.6 ou version ultérieure doit désormais être installé sur vos appareils principaux, y compris <code>pip</code> pour votre version de Python, avant de déployer ce composant.
1.0.1	Version mise à jour pour la version 2.3.0 de Greengrass Nucleus.
1.0.0	Première version.

Classification des images DLR

Le composant de classification d'images DLR (`aws.greengrass.DLRImageClassification`) contient un exemple de code d'inférence permettant d'effectuer une inférence de classification d'images à l'aide des modèles [Deep Learning Runtime](#) et Resnet-50. Ce composant utilise la variante [magasin de modèles de classification d'images DLR](#) et les [Temps d'exécution du DLR](#) composants comme dépendances pour télécharger le DLR et les exemples de modèles.

Pour utiliser ce composant d'inférence avec un modèle DLR entraîné sur mesure, [créez une version personnalisée](#) du composant Model Store dépendant. Pour utiliser votre propre code d'inférence personnalisé, vous pouvez utiliser la recette de ce composant comme modèle pour [créer un composant d'inférence personnalisé](#).

Rubriques

- [Versions](#)
- [Type](#)
- [Système d'exploitation](#)
- [Prérequis](#)
- [Dépendances](#)
- [Configuration](#)
- [Fichier journal local](#)
- [Journal des modifications](#)

Versions

Les versions de ce composant sont les suivantes :

- 2,1x
- 2,0.x

Type

Ce composant est un composant générique (`aws.greengrass.generic`). Le [noyau Greengrass](#) exécute les scripts de cycle de vie du composant.

Pour de plus amples informations, veuillez consulter [Types de composants](#).

Système d'exploitation

Ce composant peut être installé sur les appareils principaux qui exécutent les systèmes d'exploitation suivants :

- Linux
- Windows

Prérequis

Ce composant répond aux exigences suivantes :

- Sur les appareils principaux de Greengrass exécutant Amazon Linux 2 ou Ubuntu 18.04, la version 2.27 ou ultérieure de la [bibliothèque GNU C](#) (glibc) est installée sur l'appareil.

- Sur les appareils ARMv7L, tels que Raspberry Pi, les dépendances pour OpenCV-Python sont installées sur l'appareil. Exécutez la commande suivante pour installer les dépendances.

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev  
libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

- Les appareils Raspberry Pi qui exécutent le système d'exploitation Raspberry Pi Bullseye doivent répondre aux exigences suivantes :
- NumPy 1.22.4 ou version ultérieure installée sur l'appareil. Raspberry Pi OS Bullseye inclut une version antérieure de NumPy. Vous pouvez donc exécuter la commande suivante pour effectuer la mise à niveau NumPy sur l'appareil.

```
pip3 install --upgrade numpy
```

- L'ancienne pile de caméras activée sur l'appareil. Raspberry Pi OS Bullseye inclut une nouvelle pile de caméras activée par défaut et non compatible. Vous devez donc activer la pile de caméras existante.

Pour activer l'ancienne pile de caméras

1. Exécutez la commande suivante pour ouvrir l'outil de configuration du Raspberry Pi.

```
sudo raspi-config
```

2. Sélectionnez Options d'interface.
3. Sélectionnez Legacy camera pour activer l'ancienne pile de caméras.
4. Redémarrez l'appareil Raspberry Pi.

Dépendances

Lorsque vous déployez un composant, il déploie AWS IoT Greengrass également des versions compatibles de ses dépendances. Cela signifie que vous devez satisfaire aux exigences relatives au composant et à toutes ses dépendances pour réussir le déploiement du composant. Cette section répertorie les dépendances des [versions publiées](#) de ce composant et les contraintes de version sémantiques qui définissent les versions des composants pour chaque dépendance. Vous pouvez également consulter les dépendances de chaque version du composant dans la [AWS IoT Greengrass console](#). Sur la page de détails du composant, recherchez la liste des dépendances.

2.1.13 and 2.1.14

Le tableau suivant répertorie les dépendances pour les versions 2.1.13 et 2.1.14 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0,0 <2,13,0	Flexible
magasin de modèles de classification d'images DLR	~2,10	Stricte
DLR	~1,6,0	Stricte

2.1.12

Le tableau suivant répertorie les dépendances pour la version 2.1.12 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0,0 <2,12,0	Flexible
magasin de modèles de classification d'images DLR	~2,10	Stricte
DLR	~1,6,0	Stricte

2.1.11

Le tableau suivant répertorie les dépendances pour la version 2.1.11 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0,0 <2,11,0	Flexible
magasin de modèles de classification d'images DLR	~2,10	Stricte

Dépendance	Versions compatibles	Type de dépendance
DLR	~1,6,0	Stricte

2.1.10

Le tableau suivant répertorie les dépendances pour la version 2.1.10 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0,0 <2,1,0	Flexible
magasin de modèles de classification d'images DLR	~2,10	Stricte
DLR	~1,6,0	Stricte

2.1.9

Le tableau suivant répertorie les dépendances pour la version 2.1.9 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0,0 <2,9.0	Flexible
magasin de modèles de classification d'images DLR	~2,10	Stricte
DLR	~1,6,0	Stricte

2.1.8

Le tableau suivant répertorie les dépendances pour la version 2.1.8 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0,0 <2,8.0	Flexible

Dépendance	Versions compatibles	Type de dépendance
magasin de modèles de classification d'images DLR	~2,10	Stricte
DLR	~1,6,0	Stricte

2.1.7

Le tableau suivant répertorie les dépendances pour la version 2.1.7 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,7.0	Flexible
magasin de modèles de classification d'images DLR	~2,10	Stricte
DLR	~1,6,0	Stricte

2.1.6

Le tableau suivant répertorie les dépendances pour la version 2.1.6 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,6.0	Flexible
magasin de modèles de classification d'images DLR	~2,10	Stricte
DLR	~1,6,0	Stricte

2.1.4 - 2.1.5

Le tableau suivant répertorie les dépendances pour les versions 2.1.4 à 2.1.5 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,5.0	Flexible
magasin de modèles de classification d'images DLR	~2,10	Stricte
DLR	~1,6,0	Stricte

2.1.3

Le tableau suivant répertorie les dépendances pour la version 2.1.3 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,4.0	Flexible
magasin de modèles de classification d'images DLR	~2,10	Stricte
DLR	~1,6,0	Stricte

2.1.2

Le tableau suivant répertorie les dépendances pour la version 2.1.2 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,3.0	Flexible
magasin de modèles de classification d'images DLR	~2,10	Stricte
DLR	~1,6,0	Stricte

2.1.1

Le tableau suivant répertorie les dépendances pour la version 2.1.1 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,2.0	Flexible
magasin de modèles de classification d'images DLR	~2,10	Stricte
DLR	~1,6,0	Stricte

2.0.x

Le tableau suivant répertorie les dépendances pour la version 2.0.x de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	~2,0.0	Flexible
magasin de modèles de classification d'images DLR	~2,0.0	Stricte
DLR	~1,3,0	Flexible

Configuration

Ce composant fournit les paramètres de configuration suivants que vous pouvez personnaliser lorsque vous déployez le composant.

2.1.x

`accessControl`

(Facultatif) Objet contenant la [politique d'autorisation](#) qui permet au composant de publier des messages dans la rubrique de notifications par défaut.

Par défaut :

```
{
  "aws.greengrass.ipc.mqttproxy": {
```

```
"aws.greengrass.DLRIImageClassification:mqttproxy:1": {
  "policyDescription": "Allows access to publish via topic ml/dlr/image-
classification.",
  "operations": [
    "aws.greengrass#PublishToIoTCore"
  ],
  "resources": [
    "ml/dlr/image-classification"
  ]
}
}
```

PublishResultsOnTopic

(Facultatif) Rubrique sur laquelle vous souhaitez publier les résultats de l'inférence. Si vous modifiez cette valeur, vous devez également modifier la valeur de `resources` dans le `accessControl` paramètre pour qu'elle corresponde au nom de votre rubrique personnalisée.

Par défaut : `ml/dlr/image-classification`

Accelerator

L'accélérateur que vous souhaitez utiliser. Les valeurs prises en charge sont `cpu` et `gpu`.

Les modèles d'exemple du composant du modèle dépendant ne prennent en charge que l'accélération du processeur. Pour utiliser l'accélération GPU avec un autre modèle personnalisé, [créez un composant de modèle personnalisé](#) pour remplacer le composant de modèle public.

Par défaut : `cpu`

ImageDirectory

(Facultatif) Le chemin du dossier sur le périphérique où les composants d'inférence lisent les images. Vous pouvez modifier cette valeur en fonction de n'importe quel emplacement de votre appareil auquel vous avez read/write accès.

Par défaut : `/greengrass/v2/packages/artifacts-unarchived/component-name/image_classification/sample_images/`

 Note

Si vous définissez la valeur de `UseCamera` à `true`, ce paramètre de configuration est ignoré.

ImageName

(Facultatif) Nom de l'image que le composant d'inférence utilise comme entrée pour effectuer une prédiction. Le composant recherche l'image dans le dossier spécifié dans `ImageDirectory`. Par défaut, le composant utilise l'image d'exemple dans le répertoire d'images par défaut. AWS IoT Greengrass prend en charge les formats d'image suivants : `jpeg`, `jpg`, `png`, `etnp`.

Par défaut : `cat.jpeg`

 Note

Si vous définissez la valeur de `UseCamera` à `true`, ce paramètre de configuration est ignoré.

InferenceInterval

(Facultatif) Le délai en secondes entre chaque prédiction effectuée par le code d'inférence. L'exemple de code d'inférence s'exécute indéfiniment et répète ses prédictions à l'intervalle de temps spécifié. Par exemple, vous pouvez réduire cet intervalle si vous souhaitez utiliser des images prises par une caméra pour des prévisions en temps réel.

Par défaut : `3600`

ModelResourceKey

(Facultatif) Les modèles utilisés dans le composant de modèle public dépendant. Modifiez ce paramètre uniquement si vous remplacez le composant du modèle public par un composant personnalisé.

Par défaut :

```
{
```

```
"armv7l": "DLR-resnet50-armv7l-cpu-ImageClassification",  
"aarch64": "DLR-resnet50-aarch64-cpu-ImageClassification",  
"x86_64": "DLR-resnet50-x86_64-cpu-ImageClassification",  
"windows": "DLR-resnet50-win-cpu-ImageClassification"  
}
```

UseCamera

(Facultatif) Valeur de chaîne qui définit s'il faut utiliser les images d'une caméra connectée au périphérique principal de Greengrass. Les valeurs prises en charge sont `true` et `false`.

Lorsque vous définissez cette valeur sur `true`, l'exemple de code d'inférence accède à la caméra de votre appareil et exécute l'inférence localement sur l'image capturée. Les valeurs des `ImageDirectory` paramètres `ImageName` et sont ignorées. Assurez-vous que l'utilisateur exécutant ce composant a read/write accès à l'emplacement où l'appareil photo stocke les images capturées.

Par défaut : `false`

Note

Lorsque vous consultez la recette de ce composant, le paramètre `UseCamera` de configuration n'apparaît pas dans la configuration par défaut. Toutefois, vous pouvez modifier la valeur de ce paramètre dans une mise à [jour de fusion de configuration](#) lorsque vous déployez le composant.

Lorsque vous définissez sur `UseCamera` `true`, vous devez également créer un lien symbolique pour permettre au composant d'inférence d'accéder à votre caméra depuis l'environnement virtuel créé par le composant d'exécution. Pour plus d'informations sur l'utilisation d'une caméra avec les composants d'inférence d'échantillons, consultez [Mettre à jour les configurations des composants](#).

2.0.x

MLRootPath

(Facultatif) Le chemin du dossier sur les périphériques principaux Linux où les composants d'inférence lisent les images et écrivent les résultats d'inférence. Vous pouvez modifier cette valeur à n'importe quel emplacement de votre appareil auquel l'utilisateur exécutant ce composant a read/write accès.

Par défaut : `/greengrass/v2/work/variant.DLR/greengrass_ml`

Par défaut : `/greengrass/v2/work/variant.TensorFlowLite/greengrass_ml`

Accelerator

L'accélérateur que vous souhaitez utiliser. Les valeurs prises en charge sont cpu et gpu.

Les modèles d'exemple du composant du modèle dépendant ne prennent en charge que l'accélération du processeur. Pour utiliser l'accélération GPU avec un autre modèle personnalisé, [créez un composant de modèle personnalisé](#) pour remplacer le composant de modèle public.

Par défaut : cpu

ImageName

(Facultatif) Nom de l'image que le composant d'inférence utilise comme entrée pour effectuer une prédiction. Le composant recherche l'image dans le dossier spécifié dans `ImageDirectory`. L'emplacement par défaut est `MLRootPath/images`. AWS IoT Greengrass prend en charge les formats d'image suivants : jpeg, jpg, png, et npy.

Par défaut : `cat.jpeg`

InferenceInterval

(Facultatif) Le délai en secondes entre chaque prédiction effectuée par le code d'inférence. L'exemple de code d'inférence s'exécute indéfiniment et répète ses prédictions à l'intervalle de temps spécifié. Par exemple, vous pouvez réduire cet intervalle si vous souhaitez utiliser des images prises par une caméra pour des prévisions en temps réel.

Par défaut : 3600

ModelResourceKey

(Facultatif) Les modèles utilisés dans le composant de modèle public dépendant. Modifiez ce paramètre uniquement si vous remplacez le composant du modèle public par un composant personnalisé.

Par défaut :

```
armv7l: "DLR-resnet50-armv7l-cpu-ImageClassification"
x86_64: "DLR-resnet50-x86_64-cpu-ImageClassification"
```

Fichier journal local

Ce composant utilise le fichier journal suivant.

Linux

```
/greengrass/v2/logs/aws.greengrass.DLRIImageClassification.log
```

Windows

```
C:\greengrass\v2\logs\aws.greengrass.DLRIImageClassification.log
```

Pour consulter les journaux de ce composant

- Exécutez la commande suivante sur le périphérique principal pour afficher le fichier journal de ce composant en temps réel. Remplacez `/greengrass/v2` ou `C:\greengrass\v2` par le chemin d'accès au dossier AWS IoT Greengrass racine.

Linux

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

Windows (PowerShell)

```
Get-Content C:\greengrass\v2\logs\aws.greengrass.DLRIImageClassification.log -  
Tail 10 -Wait
```

Journal des modifications

Le tableau suivant décrit les modifications apportées à chaque version du composant.

Version	Modifications
2.1.14	Version mise à jour pour la version 2.12.5 de Greengrass Nucleus.
2.1.13	Version mise à jour pour la version 2.12.0 de Greengrass Nucleus.
2.1.12	Version mise à jour pour la version 2.11.0 de Greengrass Nucleus.

Version	Modifications
2.1.11	Version mise à jour pour la version 2.10.0 de Greengrass Nucleus.
2.1.10	Version mise à jour pour la version 2.9.0 de Greengrass Nucleus.
2.1.9	Version mise à jour pour la version 2.8.0 de Greengrass Nucleus.
2.1.8	Version mise à jour pour la version 2.7.0 de Greengrass Nucleus.
2.1.7	Version mise à jour pour la version 2.6.0 de Greengrass Nucleus.
2.1.6	Version mise à jour pour la version 2.5.0 de Greengrass Nucleus.
2.1.5	Le composant a été publié dans son intégralité Régions AWS.
2.1.4	Version mise à jour pour la version 2.4.0 de Greengrass Nucleus. Cette version n'est pas disponible en Europe (Londres) (eu-west-2).
2.1.3	Version mise à jour pour la version 2.3.0 de Greengrass Nucleus.
2.1.2	Version mise à jour pour la version 2.2.0 de Greengrass Nucleus.

Version	Modifications
2.1.1	Nouvelles fonctionnalités - Utilisez Deep Learning Runtime v1.6.0. - Ajout de la prise en charge de la classification des exemples d'images sur les plateformes Armv8 (AArch64). Cela étend la prise en charge de l'apprentissage automatique aux appareils principaux de Greengrass exécutant NVIDIA Jetson, tels que le Jetson Nano. - Activez l'intégration de la caméra pour l'inférence d'échantillons. Utilisez le nouveau paramètre <code>UseCamera</code> de configuration pour permettre à l'exemple de code d'inférence d'accéder à la caméra de votre appareil principal Greengrass et d'exécuter l'inférence localement sur l'image capturée. - Ajoutez la prise en charge de la publication des résultats d'inférence au AWS Cloud. Utilisez le nouveau paramètre <code>PublishResultsOnTopic</code> de configuration pour spécifier le sujet sur lequel vous souhaitez publier les résultats. - Ajoutez le nouveau paramètre <code>ImageDirectory</code> de configuration qui vous permet de spécifier un répertoire personnalisé pour l'image sur laquelle vous souhaitez effectuer une inférence. Corrections de bogues et améliorations - Écrivez les résultats d'inférence dans le fichier journal du composant plutôt que dans un fichier d'inférence distinct. - Utilisez le module de journalisation du logiciel AWS IoT Greengrass Core pour enregistrer la sortie des composants. - Utilisez le Kit SDK des appareils AWS IoT pour lire la configuration du composant et appliquer les modifications de configuration.
2.0.4	Première version.

Détection d'objets DLR

Le composant de détection d'objets DLR (`aws.greengrass.DLRObjectDetection`) contient un exemple de code d'inférence pour effectuer une inférence de détection d'objets à l'aide du [Deep Learning Runtime](#) et des exemples de modèles préentraînés. Ce composant utilise la variante

[Model Store dédié à la détection d'objets DLR](#) et les [Temps d'exécution du DLR](#) composants comme dépendances pour télécharger le DLR et les exemples de modèles.

Pour utiliser ce composant d'inférence avec un modèle DLR entraîné sur mesure, [créez une version personnalisée](#) du composant Model Store dépendant. Pour utiliser votre propre code d'inférence personnalisé, vous pouvez utiliser la recette de ce composant comme modèle pour [créer un composant d'inférence personnalisé](#).

Rubriques

- [Versions](#)
- [Type](#)
- [Système d'exploitation](#)
- [Prérequis](#)
- [Dépendances](#)
- [Configuration](#)
- [Fichier journal local](#)
- [Journal des modifications](#)

Versions

Les versions de ce composant sont les suivantes :

- 2,1x
- 2,0.x

Type

Ce composant est un composant générique (`aws.greengrass.generic`). Le [noyau Greengrass](#) exécute les scripts de cycle de vie du composant.

Pour de plus amples informations, veuillez consulter [Types de composants](#).

Système d'exploitation

Ce composant peut être installé sur les appareils principaux qui exécutent les systèmes d'exploitation suivants :

- Linux
- Windows

Prérequis

Ce composant répond aux exigences suivantes :

- Sur les appareils principaux de Greengrass exécutant Amazon Linux 2 ou Ubuntu 18.04, la version 2.27 ou ultérieure de la [bibliothèque GNU C](#) (glibc) est installée sur l'appareil.
- Sur les appareils ARMv7L, tels que Raspberry Pi, les dépendances pour OpenCV-Python sont installées sur l'appareil. Exécutez la commande suivante pour installer les dépendances.

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev  
libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

- Les appareils Raspberry Pi qui exécutent le système d'exploitation Raspberry Pi Bullseye doivent répondre aux exigences suivantes :
 - NumPy 1.22.4 ou version ultérieure installée sur l'appareil. Raspberry Pi OS Bullseye inclut une version antérieure de NumPy. Vous pouvez donc exécuter la commande suivante pour effectuer la mise à niveau NumPy sur l'appareil.

```
pip3 install --upgrade numpy
```

- L'ancienne pile de caméras activée sur l'appareil. Raspberry Pi OS Bullseye inclut une nouvelle pile de caméras qui est activée par défaut et qui n'est pas compatible. Vous devez donc activer la pile de caméras existante.

Pour activer l'ancienne pile de caméras

1. Exécutez la commande suivante pour ouvrir l'outil de configuration du Raspberry Pi.

```
sudo raspi-config
```

2. Sélectionnez Options d'interface.
3. Sélectionnez Legacy camera pour activer l'ancienne pile de caméras.
4. Redémarrez l'appareil Raspberry Pi.

Dépendances

Lorsque vous déployez un composant, il déploie AWS IoT Greengrass également des versions compatibles de ses dépendances. Cela signifie que vous devez satisfaire aux exigences relatives au composant et à toutes ses dépendances pour réussir le déploiement du composant. Cette section répertorie les dépendances des [versions publiées](#) de ce composant et les contraintes de version sémantiques qui définissent les versions des composants pour chaque dépendance. Vous pouvez également consulter les dépendances de chaque version du composant dans la [AWS IoT Greengrass console](#). Sur la page de détails du composant, recherchez la liste des dépendances.

2.1.13 and 2.1.14

Le tableau suivant répertorie les dépendances pour les versions 2.1.13 et 2.1.14 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,0,0 < 2,13,0$	Flexible
Model Store dédié à la détection d'objets DLR	$\sim 2,10$	Stricte
DLR	$\sim 1,6,0$	Stricte

2.1.12

Le tableau suivant répertorie les dépendances pour la version 2.1.12 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,0,0 < 2,12,0$	Flexible
Model Store dédié à la détection d'objets DLR	$\sim 2,10$	Stricte
DLR	$\sim 1,6,0$	Stricte

2.1.11

Le tableau suivant répertorie les dépendances pour la version 2.1.11 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,11.0	Flexible
Model Store dédié à la détection d'objets DLR	~2,10	Stricte
DLR	~1,6,0	Stricte

2.1.10

Le tableau suivant répertorie les dépendances pour la version 2.1.10 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0,0 <2,1,0	Flexible
Model Store dédié à la détection d'objets DLR	~2,10	Stricte
DLR	~1,6,0	Stricte

2.1.9

Le tableau suivant répertorie les dépendances pour la version 2.1.9 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,9.0	Flexible
Model Store dédié à la détection d'objets DLR	~2,10	Stricte
DLR	~1,6,0	Stricte

2.1.8

Le tableau suivant répertorie les dépendances pour la version 2.1.8 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,8.0	Flexible
Model Store dédié à la détection d'objets DLR	~2,10	Stricte
DLR	~1,6,0	Stricte

2.1.7

Le tableau suivant répertorie les dépendances pour la version 2.1.7 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,7.0	Flexible
Model Store dédié à la détection d'objets DLR	~2,10	Stricte
DLR	~1,6,0	Stricte

2.1.6

Le tableau suivant répertorie les dépendances pour la version 2.1.6 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,6.0	Flexible
Model Store dédié à la détection d'objets DLR	~2,10	Stricte
DLR	~1,6,0	Stricte

2.1.4 - 2.1.5

Le tableau suivant répertorie les dépendances pour les versions 2.1.4 à 2.1.5 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,5.0	Flexible
Model Store dédié à la détection d'objets DLR	~2,10	Stricte
DLR	~1,6,0	Stricte

2.1.3

Le tableau suivant répertorie les dépendances pour la version 2.1.3 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,4.0	Flexible
Model Store dédié à la détection d'objets DLR	~2,10	Stricte
DLR	~1,6,0	Stricte

2.1.2

Le tableau suivant répertorie les dépendances pour la version 2.1.2 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,3.0	Flexible
Model Store dédié à la détection d'objets DLR	~2,10	Stricte
DLR	~1,6,0	Stricte

2.1.1

Le tableau suivant répertorie les dépendances pour la version 2.1.1 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,2.0	Flexible
Model Store dédié à la détection d'objets DLR	~2,10	Stricte
DLR	~1,6,0	Stricte

2.0.x

Le tableau suivant répertorie les dépendances pour la version 2.0.x de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	~2,0.0	Flexible
Model Store dédié à la détection d'objets DLR	~2,0.0	Stricte
DLR	~1,3,0	Flexible

Configuration

Ce composant fournit les paramètres de configuration suivants que vous pouvez personnaliser lorsque vous déployez le composant.

2.1.x

`accessControl`

(Facultatif) Objet contenant la [politique d'autorisation](#) qui permet au composant de publier des messages dans la rubrique de notifications par défaut.

Par défaut :

```
{
  "aws.greengrass.ipc.mqttproxy": {
    "aws.greengrass.DLRObjectDetection:mqttproxy:1": {
      "policyDescription": "Allows access to publish via topic ml/dlr/object-
detection.",
      "operations": [
        "aws.greengrass#PublishToIoTCore"
      ],
      "resources": [
        "ml/dlr/object-detection"
      ]
    }
  }
}
```

PublishResultsOnTopic

(Facultatif) Rubrique sur laquelle vous souhaitez publier les résultats de l'inférence. Si vous modifiez cette valeur, vous devez également modifier la valeur de `resources` dans le `accessControl` paramètre pour qu'elle corresponde au nom de votre rubrique personnalisée.

Par défaut : `ml/dlr/object-detection`

Accelerator

L'accélérateur que vous souhaitez utiliser. Les valeurs prises en charge sont `cpu` et `gpu`.

Les modèles d'exemple du composant du modèle dépendant ne prennent en charge que l'accélération du processeur. Pour utiliser l'accélération GPU avec un autre modèle personnalisé, [créez un composant de modèle personnalisé](#) pour remplacer le composant de modèle public.

Par défaut : `cpu`

ImageDirectory

(Facultatif) Le chemin du dossier sur le périphérique où les composants d'inférence lisent les images. Vous pouvez modifier cette valeur en fonction de n'importe quel emplacement de votre appareil auquel vous avez read/write accès.

Par défaut : `/greengrass/v2/packages/artifacts-unarchived/component-name/object_detection/sample_images/`

 Note

Si vous définissez la valeur de `UseCamera` à `true`, ce paramètre de configuration est ignoré.

ImageName

(Facultatif) Nom de l'image que le composant d'inférence utilise comme entrée pour effectuer une prédiction. Le composant recherche l'image dans le dossier spécifié dans `ImageDirectory`. Par défaut, le composant utilise l'image d'exemple dans le répertoire d'images par défaut. AWS IoT Greengrass prend en charge les formats d'image suivants : `jpeg`, `jpg`, `png`, `etnp`.

Par défaut : `objects.jpg`

 Note

Si vous définissez la valeur de `UseCamera` à `true`, ce paramètre de configuration est ignoré.

InferenceInterval

(Facultatif) Le délai en secondes entre chaque prédiction effectuée par le code d'inférence. L'exemple de code d'inférence s'exécute indéfiniment et répète ses prédictions à l'intervalle de temps spécifié. Par exemple, vous pouvez réduire cet intervalle si vous souhaitez utiliser des images prises par une caméra pour des prévisions en temps réel.

Par défaut : `3600`

ModelResourceKey

(Facultatif) Les modèles utilisés dans le composant de modèle public dépendant. Modifiez ce paramètre uniquement si vous remplacez le composant du modèle public par un composant personnalisé.

Par défaut :

```
{
```

```
"armv7l": "DLR-yolo3-armv7l-cpu-ObjectDetection",  
"aarch64": "DLR-yolo3-aarch64-gpu-ObjectDetection",  
"x86_64": "DLR-yolo3-x86_64-cpu-ObjectDetection",  
"windows": "DLR-resnet50-win-cpu-ObjectDetection"  
}
```

UseCamera

(Facultatif) Valeur de chaîne qui définit s'il faut utiliser les images d'une caméra connectée au périphérique principal de Greengrass. Les valeurs prises en charge sont `true` et `false`.

Lorsque vous définissez cette valeur sur `true`, l'exemple de code d'inférence accède à la caméra de votre appareil et exécute l'inférence localement sur l'image capturée. Les valeurs des `ImageDirectory` paramètres `ImageName` et sont ignorées. Assurez-vous que l'utilisateur exécutant ce composant a read/write accès à l'emplacement où l'appareil photo stocke les images capturées.

Par défaut : `false`

Note

Lorsque vous consultez la recette de ce composant, le paramètre `UseCamera` de configuration n'apparaît pas dans la configuration par défaut. Toutefois, vous pouvez modifier la valeur de ce paramètre dans une mise à [jour de fusion de configuration](#) lorsque vous déployez le composant.

Lorsque vous définissez sur `UseCamera` `true`, vous devez également créer un lien symbolique pour permettre au composant d'inférence d'accéder à votre caméra depuis l'environnement virtuel créé par le composant d'exécution. Pour plus d'informations sur l'utilisation d'une caméra avec les composants d'inférence d'échantillons, consultez [Mettre à jour les configurations des composants](#).

2.0.x

MLRootPath

(Facultatif) Le chemin du dossier sur les périphériques principaux Linux où les composants d'inférence lisent les images et écrivent les résultats d'inférence. Vous pouvez modifier cette valeur à n'importe quel emplacement de votre appareil auquel l'utilisateur exécutant ce composant a read/write accès.

Par défaut : `/greengrass/v2/work/variant.DLR/greengrass_ml`

Par défaut : `/greengrass/v2/work/variant.TensorFlowLite/greengrass_ml`

Accelerator

Ne le modifiez pas. Actuellement, la seule valeur prise en charge pour l'accélérateur est `cpu` que les modèles des composants du modèle dépendant sont compilés uniquement pour l'accélérateur CPU.

ImageName

(Facultatif) Nom de l'image que le composant d'inférence utilise comme entrée pour effectuer une prédiction. Le composant recherche l'image dans le dossier spécifié dans `ImageDirectory`. L'emplacement par défaut est `MLRootPath/images`. AWS IoT Greengrass prend en charge les formats d'image suivants : `jpeg`, `jpg`, `png`, `etnpy`.

Par défaut : `objects.jpg`

InferenceInterval

(Facultatif) Le délai en secondes entre chaque prédiction effectuée par le code d'inférence. L'exemple de code d'inférence s'exécute indéfiniment et répète ses prédictions à l'intervalle de temps spécifié. Par exemple, vous pouvez réduire cet intervalle si vous souhaitez utiliser des images prises par une caméra pour des prévisions en temps réel.

Par défaut : `3600`

ModelResourceKey

(Facultatif) Les modèles utilisés dans le composant de modèle public dépendant. Modifiez ce paramètre uniquement si vous remplacez le composant du modèle public par un composant personnalisé.

Par défaut :

```
{
  armv7l: "DLR-yolo3-armv7l-cpu-ObjectDetection",
  x86_64: "DLR-yolo3-x86_64-cpu-ObjectDetection"
}
```

Fichier journal local

Ce composant utilise le fichier journal suivant.

Linux

```
/greengrass/v2/logs/aws.greengrass.DLRObjectDetection.log
```

Windows

```
C:\greengrass\v2\logs\aws.greengrass.DLRObjectDetection.log
```

Pour consulter les journaux de ce composant

- Exécutez la commande suivante sur le périphérique principal pour afficher le fichier journal de ce composant en temps réel. Remplacez */greengrass/v2* ou *C:\greengrass\v2* par le chemin d'accès au dossier AWS IoT Greengrass racine.

Linux

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

Windows (PowerShell)

```
Get-Content C:\greengrass\v2\logs\aws.greengrass.DLRObjectDetection.log -Tail 10  
-Wait
```

Journal des modifications

Le tableau suivant décrit les modifications apportées à chaque version du composant.

Version	Modifications
2.1.14	Version mise à jour pour la version 2.12.5 de Greengrass Nucleus.
2.1.13	Version mise à jour pour la version 2.12.0 de Greengrass Nucleus.
2.1.12	Version mise à jour pour la version 2.11.0 de Greengrass Nucleus.
2.1.11	Version mise à jour pour la version 2.10.0 de Greengrass Nucleus.
2.1.10	Version mise à jour pour la version 2.9.0 de Greengrass Nucleus.

Version	Modifications
2.1.9	Version mise à jour pour la version 2.8.0 de Greengrass Nucleus.
2.1.8	Version mise à jour pour la version 2.7.0 de Greengrass Nucleus.
2.1.7	Version mise à jour pour la version 2.6.0 de Greengrass Nucleus.
2.1.6	Version mise à jour pour la version 2.5.0 de Greengrass Nucleus.
2.1.5	Composant publié dans son intégralité Régions AWS.
2.1.4	Version mise à jour pour la version 2.4.0 de Greengrass Nucleus. Cette version n'est pas disponible en Europe (Londres) (eu-west-2).
2.1.3	Version mise à jour pour la version 2.3.0 de Greengrass Nucleus.
2.1.2	Corrections de bogues et améliorations • Résout un problème de mise à l'échelle de l'image qui entraînait des cadres de délimitation inexacts dans les exemples de résultats d'inférence de détection d'objets DLR.

Version	Modifications
2.1.1	Nouvelles fonctionnalités - Utilisez Deep Learning Runtime v1.6.0. - Ajout de la prise en charge de la détection d'échantillons d'objets sur les plateformes Armv8 (AArch64). Cela étend la prise en charge de l'apprentissage automatique pour les appareils principaux de Greengrass exécutant NVIDIA Jetson, tels que le Jetson Nano. - Activez l'intégration de la caméra pour l'inférence d'échantillons. Utilisez le nouveau paramètre <code>UseCamera</code> de configuration pour permettre à l'exemple de code d'inférence d'accéder à la caméra de votre appareil principal Greengrass et d'exécuter l'inférence localement sur l'image capturée. - Ajoutez la prise en charge de la publication des résultats d'inférence au AWS Cloud. Utilisez le nouveau paramètre <code>PublishResultsOnTopic</code> de configuration pour spécifier le sujet sur lequel vous souhaitez publier les résultats. - Ajoutez le nouveau paramètre <code>ImageDirectory</code> de configuration qui vous permet de spécifier un répertoire personnalisé pour l'image sur laquelle vous souhaitez effectuer une inférence. Corrections de bogues et améliorations - Écrivez les résultats d'inférence dans le fichier journal du composant plutôt que dans un fichier d'inférence distinct. - Utilisez le module de journalisation du logiciel AWS IoT Greengrass Core pour enregistrer la sortie des composants. - Utilisez le Kit SDK des appareils AWS IoT pour lire la configuration du composant et appliquer les modifications de configuration.
2.0.4	Première version.

magasin de modèles de classification d'images DLR

Le magasin de modèles de classification d'images DLR est un composant de modèle d'apprentissage automatique qui contient des modèles ResNet -50 pré-entraînés sous forme d'artefacts Greengrass.

Les modèles pré-entraînés utilisés dans ce composant sont extraits du GluonCV Model Zoo et compilés à l'aide d' SageMaker AI Neo Deep Learning Runtime.

Le composant d'inférence de [classification d'images DLR](#) utilise ce composant comme dépendance pour la source du modèle. Pour utiliser un modèle DLR entraîné sur mesure, [créez une version personnalisée](#) de ce composant de modèle et incluez votre modèle personnalisé en tant qu'artefact de composant. Vous pouvez utiliser la recette de ce composant comme modèle pour créer des composants de modèle personnalisés.

Note

Le nom du composant Model Store de classification d'images DLR varie en fonction de sa version. Le nom du composant pour la version 2.1.x et les versions ultérieures est `variant.DLR.ImageClassification.ModelStore`. Le nom du composant pour la version 2.0.x est `variant.ImageClassification.ModelStore`.

Rubriques

- [Versions](#)
- [Type](#)
- [Système d'exploitation](#)
- [Prérequis](#)
- [Dépendances](#)
- [Configuration](#)
- [Fichier journal local](#)
- [Journal des modifications](#)

Versions

Les versions de ce composant sont les suivantes :

- 2,1x () `variant.DLR.ImageClassification.ModelStore`
- 2,0.x () `variant.ImageClassification.ModelStore`

Type

Ce composant est un composant générique (`aws.greengrass.generic`). Le [noyau Greengrass](#) exécute les scripts de cycle de vie du composant.

Pour de plus amples informations, veuillez consulter [Types de composants](#).

Système d'exploitation

Ce composant peut être installé sur les appareils principaux qui exécutent les systèmes d'exploitation suivants :

- Linux
- Windows

Prérequis

Ce composant répond aux exigences suivantes :

- Sur les appareils principaux de Greengrass exécutant Amazon Linux 2 ou Ubuntu 18.04, la version 2.27 ou ultérieure de la [bibliothèque GNU C](#) (glibc) est installée sur l'appareil.
- Sur les appareils ARMv7L, tels que Raspberry Pi, les dépendances pour OpenCV-Python sont installées sur l'appareil. Exécutez la commande suivante pour installer les dépendances.

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev  
libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

- Les appareils Raspberry Pi qui exécutent le système d'exploitation Raspberry Pi Bullseye doivent répondre aux exigences suivantes :
 - NumPy 1.22.4 ou version ultérieure installée sur l'appareil. Raspberry Pi OS Bullseye inclut une version antérieure de NumPy. Vous pouvez donc exécuter la commande suivante pour effectuer la mise à niveau NumPy sur l'appareil.

```
pip3 install --upgrade numpy
```

- L'ancienne pile de caméras activée sur l'appareil. Raspberry Pi OS Bullseye inclut une nouvelle pile de caméras qui est activée par défaut et qui n'est pas compatible. Vous devez donc activer la pile de caméras existante.

Pour activer l'ancienne pile de caméras

1. Exécutez la commande suivante pour ouvrir l'outil de configuration du Raspberry Pi.

```
sudo raspi-config
```

2. Sélectionnez Options d'interface.
3. Sélectionnez Legacy camera pour activer l'ancienne pile de caméras.
4. Redémarrez l'appareil Raspberry Pi.

Dépendances

Lorsque vous déployez un composant, il déploie AWS IoT Greengrass également des versions compatibles de ses dépendances. Cela signifie que vous devez satisfaire aux exigences relatives au composant et à toutes ses dépendances pour réussir le déploiement du composant. Cette section répertorie les dépendances des [versions publiées](#) de ce composant et les contraintes de version sémantiques qui définissent les versions des composants pour chaque dépendance. Vous pouvez également consulter les dépendances de chaque version du composant dans la [AWS IoT Greengrass console](#). Sur la page de détails du composant, recherchez la liste des dépendances.

2.1.12 - 2.1.14

Le tableau suivant répertorie les dépendances pour les versions 2.1.12 et 2.1.13 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,0,0$ $< 2,13,0$	Flexible

2.1.11

Le tableau suivant répertorie les dépendances pour la version 2.1.11 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,0,0$ $< 2,12,0$	Flexible

2.1.10

Le tableau suivant répertorie les dépendances pour la version 2.1.10 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,11.0	Flexible

2.1.9

Le tableau suivant répertorie les dépendances pour la version 2.1.9 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0,0 <2,1,0	Flexible

2.1.8

Le tableau suivant répertorie les dépendances pour la version 2.1.8 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,9.0	Flexible

2.1.7

Le tableau suivant répertorie les dépendances pour la version 2.1.7 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,8.0	Flexible

2.1.6

Le tableau suivant répertorie les dépendances pour la version 2.1.6 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,7.0	Flexible

2.1.5

Le tableau suivant répertorie les dépendances pour la version 2.1.5 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,6.0	Flexible

2.1.4

Le tableau suivant répertorie les dépendances pour la version 2.1.4 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,5.0	Flexible

2.1.3

Le tableau suivant répertorie les dépendances pour la version 2.1.3 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,4.0	Flexible

2.1.2

Le tableau suivant répertorie les dépendances pour la version 2.1.2 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,3.0	Flexible

2.1.1

Le tableau suivant répertorie les dépendances pour la version 2.1.1 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,2.0	Flexible

2.0.x

Le tableau suivant répertorie les dépendances pour la version 2.0.x de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	~2,0.0	Flexible

Configuration

Ce composant ne possède aucun paramètre de configuration.

Fichier journal local

Ce composant ne génère pas de journaux.

Journal des modifications

Le tableau suivant décrit les modifications apportées à chaque version du composant.

Version	Modifications
2.1.13	Version mise à jour pour la version 2.12.5 de Greengrass Nucleus.
2.1.12	Version mise à jour pour la version 2.12.0 de Greengrass Nucleus.
2.1.11	Version mise à jour pour la version 2.11.0 de Greengrass Nucleus.
2.1.10	Version mise à jour pour la version 2.10.0 de Greengrass Nucleus.

Version	Modifications
2.1.9	Version mise à jour pour la version 2.9.0 de Greengrass Nucleus.
2.1.8	Version mise à jour pour la version 2.8.0 de Greengrass Nucleus.
2.1.7	Version mise à jour pour la version 2.7.0 de Greengrass Nucleus.
2.1.6	Version mise à jour pour la version 2.6.0 de Greengrass Nucleus.
2.1.5	Nouvelles fonctionnalités • Ajoute des exemples de modèles de classification d'images pour les principaux appareils Windows. • Version mise à jour pour la version 2.5.0 de Greengrass Nucleus.
2.1.4	Version mise à jour pour la version 2.4.0 de Greengrass Nucleus.
2.1.3	Version mise à jour pour la version 2.3.0 de Greengrass Nucleus.
2.1.2	Version mise à jour pour la version 2.2.0 de Greengrass Nucleus.
2.1.1	Nouvelles fonctionnalités • Ajoutez un exemple de modèle de classification d'images ResNet -50 pour les plateformes Armv8 (AArch64). Cela étend la prise en charge de l'apprentissage automatique pour les appareils principaux de Greengrass exécutant NVIDIA Jetson, tels que le Jetson Nano.
2.0.4	Première version.

Model Store dédié à la détection d'objets DLR

Le magasin de modèles de détection d'objets DLR est un composant de modèle d'apprentissage automatique qui contient des YOLOv3 modèles préentraînés sous forme d'artefacts Greengrass. Les exemples de modèles utilisés dans ce composant sont extraits du [GluonCV Model Zoo](#) et compilés à l'aide d' SageMaker AI Neo [Deep](#) Learning Runtime.

Le composant d'inférence [de détection d'objets DLR](#) utilise ce composant comme dépendance pour la source du modèle. Pour utiliser un modèle DLR entraîné sur mesure, [créez une version personnalisée](#) de ce composant de modèle et incluez votre modèle personnalisé en tant qu'artefact

de composant. Vous pouvez utiliser la recette de ce composant comme modèle pour créer des composants de modèle personnalisés.

 Note

Le nom du composant du Model Store de détection d'objets DLR varie en fonction de sa version. Le nom du composant pour la version 2.1.x et les versions ultérieures est `variant.DLR.ObjectDetection.ModelStore`. Le nom du composant pour la version 2.0.x est `variant.ObjectDetection.ModelStore`.

Rubriques

- [Versions](#)
- [Type](#)
- [Système d'exploitation](#)
- [Prérequis](#)
- [Dépendances](#)
- [Configuration](#)
- [Fichier journal local](#)
- [Journal des modifications](#)

Versions

Les versions de ce composant sont les suivantes :

- 2,1x
- 2,0.x

Type

Ce composant est un composant générique (`aws.greengrass.generic`). Le [noyau Greengrass](#) exécute les scripts de cycle de vie du composant.

Pour de plus amples informations, veuillez consulter [Types de composants](#).

Système d'exploitation

Ce composant peut être installé sur les appareils principaux qui exécutent les systèmes d'exploitation suivants :

- Linux
- Windows

Prérequis

Ce composant répond aux exigences suivantes :

- Sur les appareils principaux de Greengrass exécutant Amazon Linux 2 ou Ubuntu 18.04, la version 2.27 ou ultérieure de la [bibliothèque GNU C](#) (glibc) est installée sur l'appareil.
- Sur les appareils ARMv7L, tels que le Raspberry Pi, les dépendances pour OpenCV-Python sont installées sur l'appareil. Exécutez la commande suivante pour installer les dépendances.

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev  
libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

- Les appareils Raspberry Pi qui exécutent le système d'exploitation Raspberry Pi Bullseye doivent répondre aux exigences suivantes :
 - NumPy 1.22.4 ou version ultérieure installée sur l'appareil. Raspberry Pi OS Bullseye inclut une version antérieure de NumPy. Vous pouvez donc exécuter la commande suivante pour effectuer la mise à niveau NumPy sur l'appareil.

```
pip3 install --upgrade numpy
```

- L'ancienne pile de caméras activée sur l'appareil. Raspberry Pi OS Bullseye inclut une nouvelle pile de caméras qui est activée par défaut et qui n'est pas compatible. Vous devez donc activer la pile de caméras existante.

Pour activer l'ancienne pile de caméras

1. Exécutez la commande suivante pour ouvrir l'outil de configuration du Raspberry Pi.

```
sudo raspi-config
```

2. Sélectionnez Options d'interface.

3. Sélectionnez Legacy camera pour activer l'ancienne pile de caméras.
4. Redémarrez l'appareil Raspberry Pi.

Dépendances

Lorsque vous déployez un composant, il déploie AWS IoT Greengrass également des versions compatibles de ses dépendances. Cela signifie que vous devez satisfaire aux exigences relatives au composant et à toutes ses dépendances pour réussir le déploiement du composant. Cette section répertorie les dépendances des [versions publiées](#) de ce composant et les contraintes de version sémantiques qui définissent les versions des composants pour chaque dépendance. Vous pouvez également consulter les dépendances de chaque version du composant dans la [AWS IoT Greengrass console](#). Sur la page de détails du composant, recherchez la liste des dépendances.

2.1.13 and 2.1.14

Le tableau suivant répertorie les dépendances pour les versions 2.1.13 et 2.1.14 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,0.0 < 2,13.0$	Flexible

2.1.12

Le tableau suivant répertorie les dépendances pour la version 2.1.12 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,0.0 < 2,12.0$	Flexible

2.1.11

Le tableau suivant répertorie les dépendances pour la version 2.1.11 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,0.0 < 2,11.0$	Flexible

2.1.10

Le tableau suivant répertorie les dépendances pour la version 2.1.10 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,0,0 < 2,1,0$	Flexible

2.1.9

Le tableau suivant répertorie les dépendances pour la version 2.1.9 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,0,0 < 2,9,0$	Flexible

2.1.8

Le tableau suivant répertorie les dépendances pour la version 2.1.8 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,0,0 < 2,8,0$	Flexible

2.1.7

Le tableau suivant répertorie les dépendances pour la version 2.1.7 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,0,0 < 2,7,0$	Flexible

2.1.5 and 2.1.6

Le tableau suivant répertorie les dépendances pour les versions 2.1.5 et 2.1.6 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,6.0	Flexible

2.1.4

Le tableau suivant répertorie les dépendances pour la version 2.1.4 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,5.0	Flexible

2.1.3

Le tableau suivant répertorie les dépendances pour la version 2.1.3 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,4.0	Flexible

2.1.2

Le tableau suivant répertorie les dépendances pour la version 2.1.2 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,3.0	Flexible

2.1.1

Le tableau suivant répertorie les dépendances pour la version 2.1.1 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,2.0	Flexible

2.0.x

Le tableau suivant répertorie les dépendances pour la version 2.0.x de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	~2,0.0	Flexible

Configuration

Ce composant ne possède aucun paramètre de configuration.

Fichier journal local

Ce composant ne génère pas de journaux.

Journal des modifications

Le tableau suivant décrit les modifications apportées à chaque version du composant.

Version	Modifications
2.1.14	Version mise à jour pour la version 2.12.5 de Greengrass Nucleus.
2.1.13	Version mise à jour pour la version 2.12.0 de Greengrass Nucleus.
2.1.12	Version mise à jour pour la version 2.11.0 de Greengrass Nucleus.
2.1.11	Version mise à jour pour la version 2.10.0 de Greengrass Nucleus.
2.1.10	Version mise à jour pour la version 2.9.0 de Greengrass Nucleus.
2.1.9	Version mise à jour pour la version 2.8.0 de Greengrass Nucleus.
2.1.8	Version mise à jour pour la version 2.7.0 de Greengrass Nucleus.
2.1.7	Version mise à jour pour la version 2.6.0 de Greengrass Nucleus.
2.1.6	Ajoute un modèle de processeur pour résoudre un problème sur les appareils Armv8 (AArch64).

Version	Modifications
2.1.5	Nouvelles fonctionnalités • Ajoute des exemples de modèles de détection d'objets pour les principaux appareils Windows. Corrections de bugs et améliorations • Version mise à jour pour la version 2.5.0 de Greengrass Nucleus.
2.1.4	Version mise à jour pour la version 2.4.0 de Greengrass Nucleus.
2.1.3	Version mise à jour pour la version 2.3.0 de Greengrass Nucleus.
2.1.2	Version mise à jour pour la version 2.2.0 de Greengrass Nucleus.
2.1.1	Nouvelles fonctionnalités • Ajoutez un exemple de modèle de détection d'YOLOv3 objets pour les plateformes Armv8 (AArch64). Cela étend la prise en charge de l'apprentissage automatique pour les appareils principaux de Greengrass exécutant NVIDIA Jetson, tels que le Jetson Nano.
2.0.4	Première version.

Temps d'exécution du DLR

Le composant d'exécution DLR (`variant.DLR`) contient un script qui installe [Deep Learning Runtime](#) (DLR) et ses dépendances dans un environnement virtuel sur votre appareil. Les [Détection d'objets DLR](#) composants [Classification des images DLR](#) et utilisent ce composant comme dépendance pour l'installation du DLR. La version 1.6.x du composant installe le DLR v1.6.0 et la version 1.3.x du composant installe le DLR v1.3.0.

Pour utiliser un autre environnement d'exécution, vous pouvez utiliser la recette de ce composant comme modèle pour [créer un composant d'apprentissage automatique personnalisé](#).

Rubriques

- [Versions](#)
- [Type](#)
- [Système d'exploitation](#)

- [Prérequis](#)
- [Dépendances](#)
- [Configuration](#)
- [Utilisation](#)
- [Fichier journal local](#)
- [Journal des modifications](#)

Versions

Les versions de ce composant sont les suivantes :

- 1.6. x
- 1.3.x

Type

Ce composant est un composant générique (`aws.greengrass.generic`). Le [noyau Greengrass](#) exécute les scripts de cycle de vie du composant.

Pour de plus amples informations, veuillez consulter [Types de composants](#).

Système d'exploitation

Ce composant peut être installé sur les appareils principaux qui exécutent les systèmes d'exploitation suivants :

- Linux
- Windows

Prérequis

Ce composant répond aux exigences suivantes :

- Sur les appareils principaux de Greengrass exécutant Amazon Linux 2 ou Ubuntu 18.04, la version 2.27 ou ultérieure de la [bibliothèque GNU C](#) (glibc) est installée sur l'appareil.
- Sur les appareils ARMv7L, tels que le Raspberry Pi, les dépendances pour OpenCV-Python sont installées sur l'appareil. Exécutez la commande suivante pour installer les dépendances.

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev  
libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

- Les appareils Raspberry Pi qui exécutent le système d'exploitation Raspberry Pi Bullseye doivent répondre aux exigences suivantes :
- NumPy 1.22.4 ou version ultérieure installée sur l'appareil. Raspberry Pi OS Bullseye inclut une version antérieure de NumPy. Vous pouvez donc exécuter la commande suivante pour effectuer la mise à niveau NumPy sur l'appareil.

```
pip3 install --upgrade numpy
```

- L'ancienne pile de caméras activée sur l'appareil. Raspberry Pi OS Bullseye inclut une nouvelle pile de caméras qui est activée par défaut et qui n'est pas compatible. Vous devez donc activer la pile de caméras existante.

Pour activer l'ancienne pile de caméras

1. Exécutez la commande suivante pour ouvrir l'outil de configuration du Raspberry Pi.

```
sudo raspi-config
```

2. Sélectionnez Options d'interface.
3. Sélectionnez Legacy camera pour activer l'ancienne pile de caméras.
4. Redémarrez l'appareil Raspberry Pi.

Points de terminaison et ports

Par défaut, ce composant utilise un script d'installation pour installer les packages à l'aide des `pip` commandes `apt` `yumbrew`, et, en fonction de la plate-forme utilisée par le périphérique principal. Ce composant doit être capable d'exécuter des requêtes sortantes vers différents index de packages et référentiels pour exécuter le script d'installation. Pour autoriser le trafic sortant de ce composant à passer par un proxy ou un pare-feu, vous devez identifier les points de terminaison des index de packages et des référentiels auxquels votre périphérique principal se connecte pour l'installation.

Tenez compte des points suivants lorsque vous identifiez les points de terminaison requis pour le script d'installation de ce composant :

- Les points de terminaison dépendent de la plate-forme de l'appareil principal. Par exemple, un périphérique principal qui exécute Ubuntu utilise apt plutôt que yum ou brew. En outre, les appareils qui utilisent le même index de packages peuvent avoir des listes de sources différentes, de sorte qu'ils peuvent récupérer des packages à partir de différents référentiels.
- Les points de terminaison peuvent différer entre plusieurs appareils utilisant le même index de packages, car chaque appareil possède ses propres listes de sources qui définissent où récupérer les packages.
- Les points de terminaison peuvent changer au fil du temps. Chaque index URLs de package fournit les référentiels dans lesquels vous téléchargez des packages, et le propriétaire d'un package peut modifier ce que fournit l'index de package.

Pour plus d'informations sur les dépendances installées par ce composant et sur la façon de désactiver le script d'installation, consultez le paramètre [UseInstaller](#) de configuration.

Pour plus d'informations sur les points de terminaison et les ports requis pour le fonctionnement de base, consultez [Autoriser le trafic des appareils via un proxy ou un pare-feu](#).

Dépendances

Lorsque vous déployez un composant, il déploie AWS IoT Greengrass également des versions compatibles de ses dépendances. Cela signifie que vous devez satisfaire aux exigences relatives au composant et à toutes ses dépendances pour réussir le déploiement du composant. Cette section répertorie les dépendances des [versions publiées](#) de ce composant et les contraintes de version sémantiques qui définissent les versions des composants pour chaque dépendance. Vous pouvez également consulter les dépendances de chaque version du composant dans la [AWS IoT Greengrass console](#). Sur la page de détails du composant, recherchez la liste des dépendances.

1.6.11 - 1.6.16

Le tableau suivant répertorie les dépendances pour les versions 1.6.11 à 1.6.16 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <3,0.0	Flexible

1.6.10

Le tableau suivant répertorie les dépendances pour la version 1.6.10 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,9.0	Flexible

1.6.9

Le tableau suivant répertorie les dépendances pour la version 1.6.9 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,8.0	Flexible

1.6.8

Le tableau suivant répertorie les dépendances pour la version 1.6.8 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,7.0	Flexible

1.6.6 and 1.6.7

Le tableau suivant répertorie les dépendances pour les versions 1.6.6 et 1.6.7 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,6.0	Flexible

1.6.4 and 1.6.5

Le tableau suivant répertorie les dépendances pour les versions 1.6.4 et 1.6.5 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,5.0	Flexible

1.6.3

Le tableau suivant répertorie les dépendances pour la version 1.6.3 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,4.0	Flexible

1.6.2

Le tableau suivant répertorie les dépendances pour la version 1.6.2 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,3.0	Flexible

1.6.1

Le tableau suivant répertorie les dépendances pour la version 1.6.1 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,2.0	Flexible

1.3.x

Le tableau suivant répertorie les dépendances pour la version 1.3.x de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	~2,0.0	Flexible

Pour plus d'informations sur les dépendances des composants, consultez la [référence de la recette des composants](#).

Configuration

Ce composant fournit les paramètres de configuration suivants que vous pouvez personnaliser lorsque vous déployez le composant.

MLRootPath

(Facultatif) Le chemin du dossier sur les périphériques principaux Linux où les composants d'inférence lisent les images et écrivent les résultats d'inférence. Vous pouvez modifier cette valeur à n'importe quel emplacement de votre appareil auquel l'utilisateur exécutant ce composant a read/write accès.

Par défaut : `/greengrass/v2/work/variant.DLR/greengrass_ml`

WindowsMLRootPath

Cette fonctionnalité est disponible dans la version 1.6.6 et les versions ultérieures de ce composant.

(Facultatif) Le chemin du dossier sur le périphérique principal de Windows où les composants d'inférence lisent les images et écrivent les résultats d'inférence. Vous pouvez modifier cette valeur à n'importe quel emplacement de votre appareil auquel l'utilisateur exécutant ce composant a read/write accès.

Par défaut : `C:\greengrass\v2\work\variant.DLR\greengrass_ml`

UseInstaller

(Facultatif) Valeur de chaîne qui définit s'il faut utiliser le script d'installation dans ce composant pour installer le DLR et ses dépendances. Les valeurs prises en charge sont `true` et `false`.

Définissez cette valeur sur `false` si vous souhaitez utiliser un script personnalisé pour l'installation du DLR ou si vous souhaitez inclure des dépendances d'exécution dans une image Linux prédéfinie. Pour utiliser ce composant avec les composants d'inférence DLR AWS fournis, installez les bibliothèques suivantes, y compris les dépendances, et mettez-les à la disposition de l'utilisateur du système, par exemple celui `ggc_user` qui exécute les composants ML.

- [Python](#) 3.7 ou version ultérieure, y compris `pip` pour votre version de Python.
- [Deep Learning Runtime](#) v1.6.0
- [NumPy](#).
- [OpenCV-Python](#).
- [Kit SDK des appareils AWS IoT v2 pour Python](#).

- [AWS Python CRT \(Common Runtime\)](#).
- [Picamera](#) (pour les appareils Raspberry Pi uniquement).
- [awscammodule](#) (pour les AWS DeepLens appareils).
- LibGL (pour les appareils Linux)

Par défaut : `true`

Utilisation

Utilisez ce composant avec le paramètre `UseInstaller` de configuration défini sur `true` pour installer le DLR et ses dépendances sur votre appareil. Le composant configure un environnement virtuel sur votre appareil qui inclut l'OpenCV NumPy et les bibliothèques requises pour le DLR.

Note

Le script d'installation de ce composant installe également les dernières versions des bibliothèques système supplémentaires requises pour configurer l'environnement virtuel sur votre appareil et pour utiliser l'infrastructure d'apprentissage automatique installée. Cela peut mettre à niveau les bibliothèques système existantes sur votre appareil. Consultez le tableau suivant pour obtenir la liste des bibliothèques que ce composant installe pour chaque système d'exploitation pris en charge. Si vous souhaitez personnaliser ce processus d'installation, définissez le paramètre `UseInstaller` de configuration sur `false` et développez votre propre script d'installation.

Plateforme	Bibliothèques installées sur le système de l'appareil	Bibliothèques installées dans l'environnement virtuel
Armv7l	<code>build-essential</code> , <code>cmake</code> , <code>ca-certificates</code> , <code>git</code>	<code>setuptools</code> , <code>wheel</code>
Amazon Linux 2	<code>mesa-libGL</code>	Aucune
Ubuntu	<code>wget</code>	Aucune

Lorsque vous déployez votre composant d'inférence, ce composant d'exécution vérifie d'abord si le DLR et ses dépendances sont déjà installés sur votre appareil, puis il les installe pour vous.

Fichier journal local

Ce composant utilise le fichier journal suivant.

Linux

```
/greengrass/v2/logs/variant.DLR.log
```

Windows

```
C:\greengrass\v2\logs\variant.DLR.log
```

Pour consulter les journaux de ce composant

- Exécutez la commande suivante sur le périphérique principal pour afficher le fichier journal de ce composant en temps réel. Remplacez `/greengrass/v2` ou `C:\greengrass\v2` par le chemin d'accès au dossier AWS IoT Greengrass racine.

Linux

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

Windows (PowerShell)

```
Get-Content C:\greengrass\v2\logs\variant.DLR.log -Tail 10 -Wait
```

Journal des modifications

Le tableau suivant décrit les modifications apportées à chaque version du composant.

Version	Modifications
1.6,16	Version mise à jour pour Greengrass nucleus version 2.12.5.
1.6,12	Corrections de bogues et améliorations Corrige le script d'installation pour les utilisateurs du système d'exploitation Windows.

Version	Modifications
1.6.11	Version mise à jour pour la version 2.9.0 de Greengrass Nucleus.
1.6.10	Version mise à jour pour la version 2.8.0 de Greengrass Nucleus.
1.6.9	Version mise à jour pour la version 2.7.0 de Greengrass Nucleus.
1.6.8	Version mise à jour pour la version 2.6.0 de Greengrass Nucleus.
1.6.7	Corrections de bogues et améliorations • Met à jour le script <code>UseInstaller</code> d'installation pour installer LibGL, qui n'est pas disponible par défaut sur certaines plateformes Linux. • Met à jour le script d'installation pour toujours utiliser Python 3.9 dans l'environnement virtuel de ce composant. Cette modification permet de garantir la compatibilité avec les autres bibliothèques.
1.6.6	Nouvelles fonctionnalités • Ajoute la prise en charge des appareils principaux qui exécutent Windows. • Ajoute le nouveau paramètre <code>WindowsMLRootPath</code> de configuration que vous pouvez utiliser pour configurer le dossier des résultats d'inférence sur les appareils principaux de Windows.
1.6.5	Nouvelles fonctionnalités • Ajoute le nouveau paramètre de configuration <code>UseInstaller</code> que vous pouvez utiliser pour désactiver le script d'installation dans ce composant.
1.6.4	Version mise à jour pour la version 2.4.0 de Greengrass Nucleus.
1.6.3	Version mise à jour pour la version 2.3.0 de Greengrass Nucleus.
1.6.2	Version mise à jour pour la version 2.2.0 de Greengrass Nucleus.

Version	Modifications
1.6.1	Nouvelles fonctionnalités • Installez Deep Learning Runtime v1.6.0 et ses dépendances. • Ajout du support pour l'installation du DLR sur les plateformes Armv8 (AArch64). Cela étend la prise en charge de l'apprentissage automatique pour les appareils principaux de Greengrass exécutant NVIDIA Jetson, tels que le Jetson Nano. Corrections de bogues et améliorations • Installez-le Kit SDK des appareils AWS IoT dans l'environnement virtuel pour lire la configuration des composants et appliquer les modifications de configuration. • Corrections de bogues et améliorations mineures supplémentaires.
1.3.2	Première version. Installe le DLR v1.3.0.

TensorFlow Classification d'images Lite

Le composant de classification d'images TensorFlow Lite

(`aws.greengrass.TensorFlowLiteImageClassification`) contient un exemple de code d'inférence permettant d'effectuer une inférence de classification d'images à l'aide du moteur d'exécution [TensorFlow Lite](#) et un exemple de modèle quantifié MobileNet 1.0 préentraîné. Ce composant utilise la variante [TensorFlow Boutique de modèles de classification d'images Lite](#) et les [TensorFlow Temps d'exécution allégé](#) composants comme dépendances pour télécharger le runtime TensorFlow Lite et le modèle d'exemple.

Pour utiliser ce composant d'inférence avec un modèle TensorFlow Lite entraîné sur mesure, [créez une version personnalisée](#) du composant Model Store dépendant. Pour utiliser votre propre code d'inférence personnalisé, vous pouvez utiliser la recette de ce composant comme modèle pour [créer un composant d'inférence personnalisé](#).

Rubriques

- [Versions](#)
- [Type](#)
- [Système d'exploitation](#)

- [Prérequis](#)
- [Dépendances](#)
- [Configuration](#)
- [Fichier journal local](#)
- [Journal des modifications](#)

Versions

Les versions de ce composant sont les suivantes :

- 2,1x

Type

Ce composant est un composant générique (`aws.greengrass.generic`). Le [noyau Greengrass](#) exécute les scripts de cycle de vie du composant.

Pour de plus amples informations, veuillez consulter [Types de composants](#).

Système d'exploitation

Ce composant peut être installé sur les appareils principaux qui exécutent les systèmes d'exploitation suivants :

- Linux
- Windows

Prérequis

Ce composant répond aux exigences suivantes :

- Sur les appareils principaux de Greengrass exécutant Amazon Linux 2 ou Ubuntu 18.04, la version 2.27 ou ultérieure de la [bibliothèque GNU C](#) (glibc) est installée sur l'appareil.
- Sur les appareils ARMv7L, tels que le Raspberry Pi, les dépendances pour OpenCV-Python sont installées sur l'appareil. Exécutez la commande suivante pour installer les dépendances.

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev  
libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

- Les appareils Raspberry Pi qui exécutent le système d'exploitation Raspberry Pi Bullseye doivent répondre aux exigences suivantes :
 - NumPy 1.22.4 ou version ultérieure installée sur l'appareil. Raspberry Pi OS Bullseye inclut une version antérieure de NumPy. Vous pouvez donc exécuter la commande suivante pour effectuer la mise à niveau NumPy sur l'appareil.

```
pip3 install --upgrade numpy
```

- L'ancienne pile de caméras activée sur l'appareil. Raspberry Pi OS Bullseye inclut une nouvelle pile de caméras qui est activée par défaut et qui n'est pas compatible. Vous devez donc activer la pile de caméras existante.

Pour activer l'ancienne pile de caméras

1. Exécutez la commande suivante pour ouvrir l'outil de configuration du Raspberry Pi.

```
sudo raspi-config
```

2. Sélectionnez Options d'interface.
3. Sélectionnez Legacy camera pour activer l'ancienne pile de caméras.
4. Redémarrez l'appareil Raspberry Pi.

Dépendances

Lorsque vous déployez un composant, il déploie AWS IoT Greengrass également des versions compatibles de ses dépendances. Cela signifie que vous devez satisfaire aux exigences relatives au composant et à toutes ses dépendances pour réussir le déploiement du composant. Cette section répertorie les dépendances des [versions publiées](#) de ce composant et les contraintes de version sémantiques qui définissent les versions des composants pour chaque dépendance. Vous pouvez également consulter les dépendances de chaque version du composant dans la [AWS IoT Greengrass console](#). Sur la page de détails du composant, recherchez la liste des dépendances.

2.1.11 and 2.1.12

Le tableau suivant répertorie les dépendances pour les versions 2.1.11 et 2.1.12 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0,0 <2,13,0	Flexible
TensorFlow Boutique de modèles de classification d'images Lite	>=2,10 <2,2,0	Stricte
TensorFlow Lite	>=2,5,0 <2,6,0	Stricte

2.1.10

Le tableau suivant répertorie les dépendances pour la version 2.1.10 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0,0 <2,12,0	Flexible
TensorFlow Boutique de modèles de classification d'images Lite	>=2,10 <2,2,0	Stricte
TensorFlow Lite	>=2,5,0 <2,6,0	Stricte

2.1.9

Le tableau suivant répertorie les dépendances pour la version 2.1.9 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0,0 <2,11,0	Flexible
TensorFlow Boutique de modèles de classification d'images Lite	>=2,10 <2,2,0	Stricte
TensorFlow Lite	>=2,5,0 <2,6,0	Stricte

2.1.8

Le tableau suivant répertorie les dépendances pour la version 2.1.8 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,0,0 < 2,1,0$	Flexible
TensorFlow Boutique de modèles de classification d'images Lite	$\geq 2,10 < 2,2,0$	Stricte
TensorFlow Lite	$\geq 2,5,0 < 2,6,0$	Stricte

2.1.7

Le tableau suivant répertorie les dépendances pour la version 2.1.7 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,0,0 < 2,9,0$	Flexible
TensorFlow Boutique de modèles de classification d'images Lite	$\geq 2,10 < 2,2,0$	Stricte
TensorFlow Lite	$\geq 2,5,0 < 2,6,0$	Stricte

2.1.6

Le tableau suivant répertorie les dépendances pour la version 2.1.6 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,0,0 < 2,8,0$	Flexible

Dépendance	Versions compatibles	Type de dépendance
TensorFlow Boutique de modèles de classification d'images Lite	>=2,10 <2,2,0	Stricte
TensorFlow Lite	>=2,5,0 <2,6,0	Stricte

2.1.5

Le tableau suivant répertorie les dépendances pour la version 2.1.5 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,7.0	Flexible
TensorFlow Boutique de modèles de classification d'images Lite	>=2,10 <2,2,0	Stricte
TensorFlow Lite	>=2,5,0 <2,6,0	Stricte

2.1.4

Le tableau suivant répertorie les dépendances pour la version 2.1.4 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,6.0	Flexible
TensorFlow Boutique de modèles de classification d'images Lite	>=2,10 <2,2,0	Stricte
TensorFlow Lite	>=2,5,0 <2,6,0	Stricte

2.1.3

Le tableau suivant répertorie les dépendances pour la version 2.1.3 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,0,0 < 2,5,0$	Flexible
TensorFlow Boutique de modèles de classification d'images Lite	$\geq 2,10 < 2,2,0$	Stricte
TensorFlow Lite	$\geq 2,5,0 < 2,6,0$	Stricte

2.1.2

Le tableau suivant répertorie les dépendances pour la version 2.1.2 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,0,0 < 2,4,0$	Flexible
TensorFlow Boutique de modèles de classification d'images Lite	$\geq 2,10 < 2,2,0$	Stricte
TensorFlow Lite	$\geq 2,5,0 < 2,6,0$	Stricte

2.1.1

Le tableau suivant répertorie les dépendances pour la version 2.1.1 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,0,0 < 2,3,0$	Flexible

Dépendance	Versions compatibles	Type de dépendance
TensorFlow Boutique de modèles de classification d'images Lite	>=2,10 <2,2,0	Stricte
TensorFlow Lite	>=2,5,0 <2,6,0	Stricte

2.1.0

Le tableau suivant répertorie les dépendances pour la version 2.1.0 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,2.0	Flexible
TensorFlow Boutique de modèles de classification d'images Lite	>=2,10 <2,2,0	Stricte
TensorFlow Lite	>=2,5,0 <2,6,0	Stricte

Configuration

Ce composant fournit les paramètres de configuration suivants que vous pouvez personnaliser lorsque vous déployez le composant.

accessControl

(Facultatif) Objet contenant la [politique d'autorisation](#) qui permet au composant de publier des messages dans la rubrique de notifications par défaut.

Par défaut :

```
{
  "aws.greengrass.ipc.mqttproxy": {
    "aws.greengrass.TensorFlowLiteImageClassification:mqttproxy:1": {
      "policyDescription": "Allows access to publish via topic ml/tflite/image-classification.",

```

```
    "operations": [  
      "aws.greengrass#PublishToIoTCore"  
    ],  
    "resources": [  
      "ml/tflite/image-classification"  
    ]  
  }  
}  
}
```

PublishResultsOnTopic

(Facultatif) Rubrique sur laquelle vous souhaitez publier les résultats de l'inférence. Si vous modifiez cette valeur, vous devez également modifier la valeur de `resources` dans le `accessControl` paramètre pour qu'elle corresponde au nom de votre rubrique personnalisée.

Par défaut : `ml/tflite/image-classification`

Accelerator

L'accélérateur que vous souhaitez utiliser. Les valeurs prises en charge sont `cpu` et `gpu`.

Les modèles d'exemple du composant du modèle dépendant ne prennent en charge que l'accélération du processeur. Pour utiliser l'accélération GPU avec un autre modèle personnalisé, [créez un composant de modèle personnalisé](#) pour remplacer le composant de modèle public.

Par défaut : `cpu`

ImageDirectory

(Facultatif) Le chemin du dossier sur le périphérique où les composants d'inférence lisent les images. Vous pouvez modifier cette valeur en fonction de n'importe quel emplacement de votre appareil auquel vous avez read/write accès.

Par défaut : `/greengrass/v2/packages/artifacts-unarchived/component-name/image_classification/sample_images/`

Note

Si vous définissez la valeur de `UseCamera` to `true`, ce paramètre de configuration est ignoré.

ImageName

(Facultatif) Nom de l'image que le composant d'inférence utilise comme entrée pour effectuer une prédiction. Le composant recherche l'image dans le dossier spécifié dans `ImageDirectory`. Par défaut, le composant utilise l'image d'exemple dans le répertoire d'images par défaut. AWS IoT Greengrass prend en charge les formats d'image suivants : `jpeg`, `jpg`, `png`, `etnpy`.

Par défaut : `cat.jpeg`

Note

Si vous définissez la valeur de `UseCamera` à `true`, ce paramètre de configuration est ignoré.

InferenceInterval

(Facultatif) Le délai en secondes entre chaque prédiction effectuée par le code d'inférence. L'exemple de code d'inférence s'exécute indéfiniment et répète ses prédictions à l'intervalle de temps spécifié. Par exemple, vous pouvez réduire cet intervalle si vous souhaitez utiliser des images prises par une caméra pour des prévisions en temps réel.

Par défaut : `3600`

ModelResourceKey

(Facultatif) Les modèles utilisés dans le composant de modèle public dépendant. Modifiez ce paramètre uniquement si vous remplacez le composant du modèle public par un composant personnalisé.

Par défaut :

```
{
  "model": "TensorFlowLite-Mobilenet"
}
```

UseCamera

(Facultatif) Valeur de chaîne qui définit s'il faut utiliser les images d'une caméra connectée au périphérique principal de Greengrass. Les valeurs prises en charge sont `true` et `false`.

Lorsque vous définissez cette valeur sur `true`, l'exemple de code d'inférence accède à la caméra de votre appareil et exécute l'inférence localement sur l'image capturée. Les valeurs des `ImageDirectory` paramètres `ImageName` et sont ignorées. Assurez-vous que l'utilisateur exécutant ce composant a read/write accès à l'emplacement où l'appareil photo stocke les images capturées.

Par défaut : `false`

Note

Lorsque vous consultez la recette de ce composant, le paramètre `UseCamera` de configuration n'apparaît pas dans la configuration par défaut. Toutefois, vous pouvez modifier la valeur de ce paramètre dans une mise à [jour de fusion de configuration](#) lorsque vous déployez le composant.

Lorsque vous définissez sur `UseCamera` `true`, vous devez également créer un lien symbolique pour permettre au composant d'inférence d'accéder à votre caméra depuis l'environnement virtuel créé par le composant d'exécution. Pour plus d'informations sur l'utilisation d'une caméra avec les composants d'inférence d'échantillons, consultez [Mettre à jour les configurations des composants](#).

Fichier journal local

Ce composant utilise le fichier journal suivant.

Linux

```
/greengrass/v2/logs/aws.greengrass.TensorFlowLiteImageClassification.log
```

Windows

```
C:\greengrass\v2\logs\aws.greengrass.TensorFlowLiteImageClassification.log
```

Pour consulter les journaux de ce composant

- Exécutez la commande suivante sur le périphérique principal pour afficher le fichier journal de ce composant en temps réel. Remplacez `/greengrass/v2` ou `C:\greengrass\v2` par le chemin d'accès au dossier AWS IoT Greengrass racine.

Linux

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

Windows (PowerShell)

```
Get-Content C:\greengrass\v2\logs  
\aws.greengrass.TensorFlowLiteImageClassification.log -Tail 10 -Wait
```

Journal des modifications

Le tableau suivant décrit les modifications apportées à chaque version du composant.

Version	Modifications
2.1.12	Version mise à jour pour la version 2.12.5 de Greengrass Nucleus.
2.1.11	Version mise à jour pour la version 2.12.0 de Greengrass Nucleus.
2.1.10	Version mise à jour pour la version 2.11.0 de Greengrass Nucleus.
2.1.9	Version mise à jour pour la version 2.10.0 de Greengrass Nucleus.
2.1.8	Version mise à jour pour la version 2.9.0 de Greengrass Nucleus.
2.1.7	Version mise à jour pour la version 2.8.0 de Greengrass Nucleus.
2.1.6	Version mise à jour pour la version 2.7.0 de Greengrass Nucleus.
2.1.5	Version mise à jour pour la version 2.6.0 de Greengrass Nucleus.
2.1.4	Version mise à jour pour la version 2.5.0 de Greengrass Nucleus.
2.1.3	Version mise à jour pour la version 2.4.0 de Greengrass Nucleus.
2.1.2	Version mise à jour pour la version 2.3.0 de Greengrass Nucleus.
2.1.1	Version mise à jour pour la version 2.2.0 de Greengrass Nucleus.

Version	Modifications
2.1.0	Première version.

TensorFlow Détection d'objets allégée

Le composant de détection d'objets TensorFlow Lite (`aws.greengrass.TensorFlowLiteObjectDetection`) contient un exemple de code d'inférence permettant d'effectuer une inférence de détection d'objets à l'aide de [TensorFlow Lite](#) et un exemple de modèle Single Shot Detection (SSD) MobileNet 1.0 préentraîné. Ce composant utilise la variante [TensorFlow Boutique de modèles de détection d'objets Lite](#) et les [TensorFlow Temps d'exécution allégé](#) composants comme dépendances pour télécharger TensorFlow Lite et l'exemple de modèle.

Pour utiliser ce composant d'inférence avec un modèle TensorFlow Lite entraîné sur mesure, vous pouvez [créer une version personnalisée](#) du composant Model Store dépendant. Pour utiliser votre propre code d'inférence personnalisé, utilisez la recette de ce composant comme modèle pour [créer un composant d'inférence personnalisé](#).

Rubriques

- [Versions](#)
- [Type](#)
- [Système d'exploitation](#)
- [Prérequis](#)
- [Dépendances](#)
- [Configuration](#)
- [Fichier journal local](#)
- [Journal des modifications](#)

Versions

Les versions de ce composant sont les suivantes :

- 2,1x

Type

Ce composant est un composant générique (`aws.greengrass.generic`). Le [noyau Greengrass](#) exécute les scripts de cycle de vie du composant.

Pour de plus amples informations, veuillez consulter [Types de composants](#).

Système d'exploitation

Ce composant peut être installé sur les appareils principaux qui exécutent les systèmes d'exploitation suivants :

- Linux
- Windows

Prérequis

Ce composant répond aux exigences suivantes :

- Sur les appareils principaux de Greengrass exécutant Amazon Linux 2 ou Ubuntu 18.04, la version 2.27 ou ultérieure de la [bibliothèque GNU C](#) (glibc) est installée sur l'appareil.
- Sur les appareils ARMv7L, tels que Raspberry Pi, les dépendances pour OpenCV-Python sont installées sur l'appareil. Exécutez la commande suivante pour installer les dépendances.

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev  
libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

- Les appareils Raspberry Pi qui exécutent le système d'exploitation Raspberry Pi Bullseye doivent répondre aux exigences suivantes :
 - NumPy 1.22.4 ou version ultérieure installée sur l'appareil. Raspberry Pi OS Bullseye inclut une version antérieure de NumPy. Vous pouvez donc exécuter la commande suivante pour effectuer la mise à niveau NumPy sur l'appareil.

```
pip3 install --upgrade numpy
```

- L'ancienne pile de caméras activée sur l'appareil. Raspberry Pi OS Bullseye inclut une nouvelle pile de caméras activée par défaut et non compatible. Vous devez donc activer la pile de caméras existante.

Pour activer l'ancienne pile de caméras

1. Exécutez la commande suivante pour ouvrir l'outil de configuration du Raspberry Pi.

```
sudo raspi-config
```

2. Sélectionnez Options d'interface.
3. Sélectionnez Legacy camera pour activer l'ancienne pile de caméras.
4. Redémarrez l'appareil Raspberry Pi.

Dépendances

Lorsque vous déployez un composant, il déploie AWS IoT Greengrass également des versions compatibles de ses dépendances. Cela signifie que vous devez satisfaire aux exigences relatives au composant et à toutes ses dépendances pour réussir le déploiement du composant. Cette section répertorie les dépendances des [versions publiées](#) de ce composant et les contraintes de version sémantiques qui définissent les versions des composants pour chaque dépendance. Vous pouvez également consulter les dépendances de chaque version du composant dans la [AWS IoT Greengrass console](#). Sur la page de détails du composant, recherchez la liste des dépendances.

2.1.11 and 2.1.12

Le tableau suivant répertorie les dépendances pour les versions 2.1.11 et 2.1.12 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,0,0$ $< 2,13,0$	Flexible
TensorFlow Boutique de modèles de classification d'images Lite	$\geq 2,10$ $< 2,2,0$	Stricte
TensorFlow Lite	$\geq 2,5,0$ $< 2,6,0$	Stricte

2.1.10

Le tableau suivant répertorie les dépendances pour la version 2.1.10 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,12.0	Flexible
TensorFlow Boutique de modèles de classification d'images Lite	>=2,10 <2,2,0	Stricte
TensorFlow Lite	>=2,5,0 <2,6,0	Stricte

2.1.9

Le tableau suivant répertorie les dépendances pour la version 2.1.9 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,11.0	Flexible
TensorFlow Boutique de modèles de classification d'images Lite	>=2,10 <2,2,0	Stricte
TensorFlow Lite	>=2,5,0 <2,6,0	Stricte

2.1.8

Le tableau suivant répertorie les dépendances pour la version 2.1.8 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0,0 <2,1,0	Flexible
TensorFlow Boutique de modèles de classification d'images Lite	>=2,10 <2,2,0	Stricte
TensorFlow Lite	>=2,5,0 <2,6,0	Stricte

2.1.7

Le tableau suivant répertorie les dépendances pour la version 2.1.7 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,0,0$ $< 2,9,0$	Flexible
TensorFlow Boutique de modèles de classification d'images Lite	$\geq 2,10$ $< 2,2,0$	Stricte
TensorFlow Lite	$\geq 2,5,0$ $< 2,6,0$	Stricte

2.1.6

Le tableau suivant répertorie les dépendances pour la version 2.1.6 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,0,0$ $< 2,8,0$	Flexible
TensorFlow Boutique de modèles de classification d'images Lite	$\geq 2,10$ $< 2,2,0$	Stricte
TensorFlow Lite	$\geq 2,5,0$ $< 2,6,0$	Stricte

2.1.5

Le tableau suivant répertorie les dépendances pour la version 2.1.5 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,0,0$ $< 2,7,0$	Flexible

Dépendance	Versions compatibles	Type de dépendance
TensorFlow Boutique de modèles de classification d'images Lite	>=2,10 <2,2,0	Stricte
TensorFlow Lite	>=2,5,0 <2,6,0	Stricte

2.1.4

Le tableau suivant répertorie les dépendances pour la version 2.1.4 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,6.0	Flexible
TensorFlow Boutique de modèles de classification d'images Lite	>=2,10 <2,2,0	Stricte
TensorFlow Lite	>=2,5,0 <2,6,0	Stricte

2.1.3

Le tableau suivant répertorie les dépendances pour la version 2.1.3 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,5.0	Flexible
TensorFlow Boutique de modèles de classification d'images Lite	>=2,10 <2,2,0	Stricte
TensorFlow Lite	>=2,5,0 <2,6,0	Stricte

2.1.2

Le tableau suivant répertorie les dépendances pour la version 2.1.2 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,0,0 < 2,4,0$	Flexible
TensorFlow Boutique de modèles de classification d'images Lite	$\geq 2,10 < 2,2,0$	Stricte
TensorFlow Lite	$\geq 2,5,0 < 2,6,0$	Stricte

2.1.1

Le tableau suivant répertorie les dépendances pour la version 2.1.1 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,0,0 < 2,3,0$	Flexible
TensorFlow Boutique de modèles de classification d'images Lite	$\geq 2,10 < 2,2,0$	Stricte
TensorFlow Lite	$\geq 2,5,0 < 2,6,0$	Stricte

2.1.0

Le tableau suivant répertorie les dépendances pour la version 2.1.0 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,0,0 < 2,2,0$	Flexible

Dépendance	Versions compatibles	Type de dépendance
TensorFlow Boutique de modèles de classification d'images Lite	>=2,10 <2,2,0	Stricte
TensorFlow Lite	>=2,5,0 <2,6,0	Stricte

Configuration

Ce composant fournit les paramètres de configuration suivants que vous pouvez personnaliser lorsque vous déployez le composant.

accessControl

(Facultatif) Objet contenant la [politique d'autorisation](#) qui permet au composant de publier des messages dans la rubrique de notifications par défaut.

Par défaut :

```
{
  "aws.greengrass.ipc.mqttproxy": {
    "aws.greengrass.TensorFlowLiteObjectDetection:mqttproxy:1": {
      "policyDescription": "Allows access to publish via topic ml/tflite/object-
detection.",
      "operations": [
        "aws.greengrass#PublishToIoTCore"
      ],
      "resources": [
        "ml/tflite/object-detection"
      ]
    }
  }
}
```

PublishResultsOnTopic

(Facultatif) Rubrique sur laquelle vous souhaitez publier les résultats de l'inférence. Si vous modifiez cette valeur, vous devez également modifier la valeur de `resources` dans le `accessControl` paramètre pour qu'elle corresponde au nom de votre rubrique personnalisée.

Par défaut : `ml/tflite/object-detection`

Accelerator

L'accélérateur que vous souhaitez utiliser. Les valeurs prises en charge sont `cpu` et `gpu`.

Les modèles d'exemple du composant du modèle dépendant ne prennent en charge que l'accélération du processeur. Pour utiliser l'accélération GPU avec un autre modèle personnalisé, [créez un composant de modèle personnalisé](#) pour remplacer le composant de modèle public.

Par défaut : `cpu`

ImageDirectory

(Facultatif) Le chemin du dossier sur le périphérique où les composants d'inférence lisent les images. Vous pouvez modifier cette valeur en fonction de n'importe quel emplacement de votre appareil auquel vous avez read/write accès.

Par défaut : `/greengrass/v2/packages/artifacts-unarchived/component-name/object_detection/sample_images/`

Note

Si vous définissez la valeur de `UseCamera` à `true`, ce paramètre de configuration est ignoré.

ImageName

(Facultatif) Nom de l'image que le composant d'inférence utilise comme entrée pour effectuer une prédiction. Le composant recherche l'image dans le dossier spécifié dans `ImageDirectory`. Par défaut, le composant utilise l'image d'exemple dans le répertoire d'images par défaut. AWS IoT Greengrass prend en charge les formats d'image suivants : `jpeg`, `jpg`, `png`, et `npz`.

Par défaut : `objects.jpg`

Note

Si vous définissez la valeur de `UseCamera` à `true`, ce paramètre de configuration est ignoré.

InferenceInterval

(Facultatif) Le délai en secondes entre chaque prédiction effectuée par le code d'inférence. L'exemple de code d'inférence s'exécute indéfiniment et répète ses prédictions à l'intervalle de temps spécifié. Par exemple, vous pouvez réduire cet intervalle si vous souhaitez utiliser des images prises par une caméra pour des prévisions en temps réel.

Par défaut : 3600

ModelResourceKey

(Facultatif) Les modèles utilisés dans le composant de modèle public dépendant. Modifiez ce paramètre uniquement si vous remplacez le composant du modèle public par un composant personnalisé.

Par défaut :

```
{  
  "model": "TensorFlowLite-SSD"  
}
```

UseCamera

(Facultatif) Valeur de chaîne qui définit s'il faut utiliser les images d'une caméra connectée au périphérique principal de Greengrass. Les valeurs prises en charge sont `true` et `false`.

Lorsque vous définissez cette valeur sur `true`, l'exemple de code d'inférence accède à la caméra de votre appareil et exécute l'inférence localement sur l'image capturée. Les valeurs des `ImageDirectory` paramètres `ImageName` et sont ignorées. Assurez-vous que l'utilisateur exécutant ce composant a read/write accès à l'emplacement où l'appareil photo stocke les images capturées.

Par défaut : `false`

Note

Lorsque vous consultez la recette de ce composant, le paramètre `UseCamera` de configuration n'apparaît pas dans la configuration par défaut. Toutefois, vous pouvez modifier la valeur de ce paramètre dans une mise à [jour de fusion de configuration](#) lorsque vous déployez le composant.

Lorsque vous définissez `UseCamera` sur `true`, vous devez également créer un lien symbolique pour permettre au composant d'inférence d'accéder à votre caméra depuis l'environnement virtuel créé par le composant d'exécution. Pour plus d'informations sur l'utilisation d'une caméra avec les composants d'inférence d'échantillons, consultez [Mettre à jour les configurations des composants](#).

Note

Lorsque vous consultez la recette de ce composant, le paramètre `UseCamera` de configuration n'apparaît pas dans la configuration par défaut. Toutefois, vous pouvez modifier la valeur de ce paramètre dans une mise à [jour de fusion de configuration](#) lorsque vous déployez le composant.

Lorsque vous définissez `UseCamera` sur `true`, vous devez également créer un lien symbolique pour permettre au composant d'inférence d'accéder à votre caméra depuis l'environnement virtuel créé par le composant d'exécution. Pour plus d'informations sur l'utilisation d'une caméra avec les composants d'inférence d'échantillons, consultez [Mettre à jour les configurations des composants](#).

Fichier journal local

Ce composant utilise le fichier journal suivant.

Linux

```
/greengrass/v2/logs/aws.greengrass.TensorFlowLiteObjectDetection.log
```

Windows

```
C:\greengrass\v2\logs\aws.greengrass.TensorFlowLiteObjectDetection.log
```

Pour consulter les journaux de ce composant

- Exécutez la commande suivante sur le périphérique principal pour afficher le fichier journal de ce composant en temps réel. Remplacez `/greengrass/v2` ou `C:\greengrass\v2` par le chemin d'accès au dossier AWS IoT Greengrass racine.

Linux

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

Windows (PowerShell)

```
Get-Content C:\greengrass\v2\logs  
\aws.greengrass.TensorFlowLiteObjectDetection.log -Tail 10 -Wait
```

Journal des modifications

Le tableau suivant décrit les modifications apportées à chaque version du composant.

Version	Modifications
2.1.12	Version mise à jour pour la version 2.12.5 de Greengrass Nucleus.
2.1.11	Version mise à jour pour la version 2.12.0 de Greengrass Nucleus.
2.1.10	Version mise à jour pour la version 2.11.0 de Greengrass Nucleus.
2.1.9	Version mise à jour pour la version 2.10.0 de Greengrass Nucleus.
2.1.8	Version mise à jour pour la version 2.9.0 de Greengrass Nucleus.
2.1.7	Version mise à jour pour la version 2.8.0 de Greengrass Nucleus.
2.1.6	Version mise à jour pour la version 2.7.0 de Greengrass Nucleus.
2.1.5	Version mise à jour pour la version 2.6.0 de Greengrass Nucleus.
2.1.4	Version mise à jour pour la version 2.5.0 de Greengrass Nucleus.
2.1.3	Version mise à jour pour la version 2.4.0 de Greengrass Nucleus.
2.1.2	Version mise à jour pour la version 2.3.0 de Greengrass Nucleus.

Version	Modifications
2.1.1	Corrections de bogues et améliorations • Résout un problème de mise à l'échelle de l'image qui entraînait des cadres de délimitation inexacts dans les résultats d'inférence de détection d'objets de Sample TensorFlow Lite.
2.1.0	Première version.

TensorFlow Boutique de modèles de classification d'images Lite

Le magasin de modèles de classification d'images TensorFlow Lite (`variant.TensorFlowLite.ImageClassification.ModelStore`) est un composant de modèle d'apprentissage automatique qui contient un modèle MobileNet v1 préentraîné en tant qu'artefact Greengrass. L'exemple de modèle utilisé dans ce composant est extrait du [TensorFlowHub](#) et implémenté à l'aide de [TensorFlow Lite](#).

Le composant [TensorFlow Classification d'images Lite](#) d'inférence utilise ce composant comme dépendance pour la source du modèle. Pour utiliser un modèle TensorFlow Lite entraîné sur mesure, [créez une version personnalisée](#) de ce composant de modèle et incluez votre modèle personnalisé en tant qu'artefact de composant. Vous pouvez utiliser la recette de ce composant comme modèle pour créer des composants de modèle personnalisés.

Rubriques

- [Versions](#)
- [Type](#)
- [Système d'exploitation](#)
- [Prérequis](#)
- [Dépendances](#)
- [Configuration](#)
- [Fichier journal local](#)
- [Journal des modifications](#)

Versions

Les versions de ce composant sont les suivantes :

- 2,1x

Type

Ce composant est un composant générique (`aws.greengrass.generic`). Le [noyau Greengrass](#) exécute les scripts de cycle de vie du composant.

Pour de plus amples informations, veuillez consulter [Types de composants](#).

Système d'exploitation

Ce composant peut être installé sur les appareils principaux qui exécutent les systèmes d'exploitation suivants :

- Linux
- Windows

Prérequis

Ce composant répond aux exigences suivantes :

- Sur les appareils principaux de Greengrass exécutant Amazon Linux 2 ou Ubuntu 18.04, la version 2.27 ou ultérieure de la [bibliothèque GNU C](#) (glibc) est installée sur l'appareil.
- Sur les appareils ARMv7L, tels que le Raspberry Pi, les dépendances pour OpenCV-Python sont installées sur l'appareil. Exécutez la commande suivante pour installer les dépendances.

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev  
libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

- Les appareils Raspberry Pi qui exécutent le système d'exploitation Raspberry Pi Bullseye doivent répondre aux exigences suivantes :
 - NumPy 1.22.4 ou version ultérieure installée sur l'appareil. Raspberry Pi OS Bullseye inclut une version antérieure de NumPy. Vous pouvez donc exécuter la commande suivante pour effectuer la mise à niveau NumPy sur l'appareil.

```
pip3 install --upgrade numpy
```

- L'ancienne pile de caméras activée sur l'appareil. Raspberry Pi OS Bullseye inclut une nouvelle pile de caméras qui est activée par défaut et qui n'est pas compatible. Vous devez donc activer la pile de caméras existante.

Pour activer l'ancienne pile de caméras

1. Exécutez la commande suivante pour ouvrir l'outil de configuration du Raspberry Pi.

```
sudo raspi-config
```

2. Sélectionnez Options d'interface.
3. Sélectionnez Legacy camera pour activer l'ancienne pile de caméras.
4. Redémarrez l'appareil Raspberry Pi.

Dépendances

Lorsque vous déployez un composant, il déploie AWS IoT Greengrass également des versions compatibles de ses dépendances. Cela signifie que vous devez satisfaire aux exigences relatives au composant et à toutes ses dépendances pour réussir le déploiement du composant. Cette section répertorie les dépendances des [versions publiées](#) de ce composant et les contraintes de version sémantiques qui définissent les versions des composants pour chaque dépendance. Vous pouvez également consulter les dépendances de chaque version du composant dans la [AWS IoT Greengrass console](#). Sur la page de détails du composant, recherchez la liste des dépendances.

2.1.11 and 2.1.12

Le tableau suivant répertorie les dépendances pour les versions 2.1.11 et 2.1.12 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0,0 <2,13,0	Flexible

2.1.10

Le tableau suivant répertorie les dépendances pour la version 2.1.10 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,12.0	Flexible

2.1.9

Le tableau suivant répertorie les dépendances pour la version 2.1.9 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,11.0	Flexible

2.1.8

Le tableau suivant répertorie les dépendances pour la version 2.1.8 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0,0 <2,1,0	Flexible

2.1.7

Le tableau suivant répertorie les dépendances pour la version 2.1.7 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,9.0	Flexible

2.1.6

Le tableau suivant répertorie les dépendances pour la version 2.1.6 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,8.0	Flexible

2.1.5

Le tableau suivant répertorie les dépendances pour la version 2.1.5 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,7.0	Flexible

2.1.4

Le tableau suivant répertorie les dépendances pour la version 2.1.4 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,6.0	Flexible

2.1.3

Le tableau suivant répertorie les dépendances pour la version 2.1.3 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,5.0	Flexible

2.1.2

Le tableau suivant répertorie les dépendances pour la version 2.1.2 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,4.0	Flexible

2.1.1

Le tableau suivant répertorie les dépendances pour la version 2.1.1 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,3.0	Flexible

2.1.0

Le tableau suivant répertorie les dépendances pour la version 2.1.0 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,2.0	Flexible

Configuration

Ce composant ne possède aucun paramètre de configuration.

Fichier journal local

Ce composant ne génère pas de journaux.

Journal des modifications

Le tableau suivant décrit les modifications apportées à chaque version du composant.

Version	Modifications
2.1.12	Version mise à jour pour la version 2.12.5 de Greengrass Nucleus.
2.1.11	Version mise à jour pour la version 2.12.0 de Greengrass Nucleus.
2.1.10	Version mise à jour pour la version 2.11.0 de Greengrass Nucleus.
2.1.9	Version mise à jour pour la version 2.10.0 de Greengrass Nucleus.
2.1.8	Version mise à jour pour la version 2.9.0 de Greengrass Nucleus.
2.1.7	Version mise à jour pour la version 2.8.0 de Greengrass Nucleus.

Version	Modifications
2.1.6	Version mise à jour pour la version 2.7.0 de Greengrass Nucleus.
2.1.5	Version mise à jour pour la version 2.6.0 de Greengrass Nucleus.
2.1.4	Version mise à jour pour la version 2.5.0 de Greengrass Nucleus.
2.1.3	Version mise à jour pour la version 2.4.0 de Greengrass Nucleus.
2.1.2	Version mise à jour pour la version 2.3.0 de Greengrass Nucleus.
2.1.1	Version mise à jour pour la version 2.2.0 de Greengrass Nucleus.
2.1.0	Première version.

TensorFlow Boutique de modèles de détection d'objets Lite

Le magasin de modèles de détection d'objets TensorFlow Lite (variant.`TensorFlowLite.ObjectDetection.ModelStore`) est un composant du modèle d'apprentissage automatique qui contient un MobileNet modèle SSD (Single Shot Detection) préentraîné en tant qu'artefact Greengrass. L'exemple de modèle utilisé dans ce composant est extrait du [TensorFlow Hub](#) et implémenté à l'aide de [TensorFlow Lite](#).

Le composant d'inférence [de détection d'objets TensorFlow Lite](#) utilise ce composant comme dépendance pour la source du modèle. Pour utiliser un modèle TensorFlow Lite entraîné sur mesure, [créez une version personnalisée](#) de ce composant de modèle et incluez votre modèle personnalisé en tant qu'artefact de composant. Vous pouvez utiliser la recette de ce composant comme modèle pour créer des composants de modèle personnalisés.

Rubriques

- [Versions](#)
- [Type](#)
- [Système d'exploitation](#)
- [Prérequis](#)
- [Dépendances](#)
- [Configuration](#)

- [Fichier journal local](#)
- [Journal des modifications](#)

Versions

Les versions de ce composant sont les suivantes :

- 2,1x

Type

Ce composant est un composant générique (`aws.greengrass.generic`). Le [noyau Greengrass](#) exécute les scripts de cycle de vie du composant.

Pour de plus amples informations, veuillez consulter [Types de composants](#).

Système d'exploitation

Ce composant peut être installé sur les appareils principaux qui exécutent les systèmes d'exploitation suivants :

- Linux
- Windows

Prérequis

Ce composant répond aux exigences suivantes :

- Sur les appareils principaux de Greengrass exécutant Amazon Linux 2 ou Ubuntu 18.04, la version 2.27 ou ultérieure de la [bibliothèque GNU C](#) (glibc) est installée sur l'appareil.
- Sur les appareils ARMv7L, tels que le Raspberry Pi, les dépendances pour OpenCV-Python sont installées sur l'appareil. Exécutez la commande suivante pour installer les dépendances.

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev  
libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

- Les appareils Raspberry Pi qui exécutent le système d'exploitation Raspberry Pi Bullseye doivent répondre aux exigences suivantes :

- NumPy 1.22.4 ou version ultérieure installée sur l'appareil. Raspberry Pi OS Bullseye inclut une version antérieure de NumPy. Vous pouvez donc exécuter la commande suivante pour effectuer la mise à niveau NumPy sur l'appareil.

```
pip3 install --upgrade numpy
```

- L'ancienne pile de caméras activée sur l'appareil. Raspberry Pi OS Bullseye inclut une nouvelle pile de caméras qui est activée par défaut et qui n'est pas compatible. Vous devez donc activer la pile de caméras existante.

Pour activer l'ancienne pile de caméras

1. Exécutez la commande suivante pour ouvrir l'outil de configuration du Raspberry Pi.

```
sudo raspi-config
```

2. Sélectionnez Options d'interface.
3. Sélectionnez Legacy camera pour activer l'ancienne pile de caméras.
4. Redémarrez l'appareil Raspberry Pi.

Dépendances

Lorsque vous déployez un composant, il déploie AWS IoT Greengrass également des versions compatibles de ses dépendances. Cela signifie que vous devez satisfaire aux exigences relatives au composant et à toutes ses dépendances pour réussir le déploiement du composant. Cette section répertorie les dépendances des [versions publiées](#) de ce composant et les contraintes de version sémantiques qui définissent les versions des composants pour chaque dépendance. Vous pouvez également consulter les dépendances de chaque version du composant dans la [AWS IoT Greengrass console](#). Sur la page de détails du composant, recherchez la liste des dépendances.

2.1.11 and 2.1.12

Le tableau suivant répertorie les dépendances pour les versions 2.1.11 et 2.1.12 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0,0 <2,13,0	Flexible

2.1.10

Le tableau suivant répertorie les dépendances pour la version 2.1.10 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,12.0	Flexible

2.1.9

Le tableau suivant répertorie les dépendances pour la version 2.1.9 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,11.0	Flexible

2.1.8

Le tableau suivant répertorie les dépendances pour la version 2.1.8 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0,0 <2,1,0	Flexible

2.1.7

Le tableau suivant répertorie les dépendances pour la version 2.1.7 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,9.0	Flexible

2.1.6

Le tableau suivant répertorie les dépendances pour la version 2.1.6 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,8.0	Flexible

2.1.5

Le tableau suivant répertorie les dépendances pour la version 2.1.5 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,7.0	Flexible

2.1.4

Le tableau suivant répertorie les dépendances pour la version 2.1.4 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,6.0	Flexible

2.1.3

Le tableau suivant répertorie les dépendances pour la version 2.1.3 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,5.0	Flexible

2.1.2

Le tableau suivant répertorie les dépendances pour la version 2.1.2 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,4.0	Flexible

2.1.1

Le tableau suivant répertorie les dépendances pour la version 2.1.1 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,3.0	Flexible

2.1.0

Le tableau suivant répertorie les dépendances pour la version 2.1.0 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,2.0	Flexible

Configuration

Ce composant ne possède aucun paramètre de configuration.

Fichier journal local

Ce composant ne génère pas de journaux.

Journal des modifications

Le tableau suivant décrit les modifications apportées à chaque version du composant.

Version	Modifications
2.1.12	Version mise à jour pour la version 2.12.5 de Greengrass Nucleus.
2.1.11	Version mise à jour pour la version 2.12.0 de Greengrass Nucleus.
2.1.10	Version mise à jour pour la version 2.11.0 de Greengrass Nucleus.
2.1.9	Version mise à jour pour la version 2.10.0 de Greengrass Nucleus.
2.1.8	Version mise à jour pour la version 2.9.0 de Greengrass Nucleus.

Version	Modifications
2.1.7	Version mise à jour pour la version 2.8.0 de Greengrass Nucleus.
2.1.6	Version mise à jour pour la version 2.7.0 de Greengrass Nucleus.
2.1.5	Version mise à jour pour la version 2.6.0 de Greengrass Nucleus.
2.1.4	Version mise à jour pour la version 2.5.0 de Greengrass Nucleus.
2.1.3	Version mise à jour pour la version 2.4.0 de Greengrass Nucleus.
2.1.2	Version mise à jour pour la version 2.3.0 de Greengrass Nucleus.
2.1.1	Version mise à jour pour la version 2.2.0 de Greengrass Nucleus.
2.1.0	Première version.

TensorFlow Temps d'exécution allégé

Le composant d'exécution TensorFlow Lite (`variant.TensorFlowLite`) contient un script qui installe la version [TensorFlow Lite](#) 2.5.0 et ses dépendances dans un environnement virtuel sur votre appareil. Le composant de [classification d'images TensorFlow TensorFlow Lite et de détection d'objets Lite](#) utilise ce composant d'exécution comme dépendance pour l'installation de TensorFlow Lite.

Note

TensorFlow Le composant d'exécution Lite v2.5.6 et versions ultérieures réinstalle les installations existantes du moteur d'exécution TensorFlow Lite et ses dépendances. Cette réinstallation permet de s'assurer que le périphérique principal exécute des versions compatibles de TensorFlow Lite et de ses dépendances.

Pour utiliser un autre environnement d'exécution, vous pouvez utiliser la recette de ce composant comme modèle pour [créer un composant d'apprentissage automatique personnalisé](#).

Rubriques

- [Versions](#)

- [Type](#)
- [Système d'exploitation](#)
- [Prérequis](#)
- [Dépendances](#)
- [Configuration](#)
- [Utilisation](#)
- [Fichier journal local](#)
- [Journal des modifications](#)

Versions

Les versions de ce composant sont les suivantes :

- 2,5.x

Type

Ce composant est un composant générique (`aws.greengrass.generic`). Le [noyau Greengrass](#) exécute les scripts de cycle de vie du composant.

Pour de plus amples informations, veuillez consulter [Types de composants](#).

Système d'exploitation

Ce composant peut être installé sur les appareils principaux qui exécutent les systèmes d'exploitation suivants :

- Linux
- Windows

Prérequis

Ce composant répond aux exigences suivantes :

- Sur les appareils principaux de Greengrass exécutant Amazon Linux 2 ou Ubuntu 18.04, la version 2.27 ou ultérieure de la [bibliothèque GNU C](#) (glibc) est installée sur l'appareil.

- Sur les appareils ARMv7L, tels que le Raspberry Pi, les dépendances pour OpenCV-Python sont installées sur l'appareil. Exécutez la commande suivante pour installer les dépendances.

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev  
libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

- Les appareils Raspberry Pi qui exécutent le système d'exploitation Raspberry Pi Bullseye doivent répondre aux exigences suivantes :
 - NumPy 1.22.4 ou version ultérieure installée sur l'appareil. Raspberry Pi OS Bullseye inclut une version antérieure de NumPy. Vous pouvez donc exécuter la commande suivante pour effectuer la mise à niveau NumPy sur l'appareil.

```
pip3 install --upgrade numpy
```

- L'ancienne pile de caméras activée sur l'appareil. Raspberry Pi OS Bullseye inclut une nouvelle pile de caméras qui est activée par défaut et qui n'est pas compatible. Vous devez donc activer la pile de caméras existante.

Pour activer l'ancienne pile de caméras

1. Exécutez la commande suivante pour ouvrir l'outil de configuration du Raspberry Pi.

```
sudo raspi-config
```

2. Sélectionnez Options d'interface.
3. Sélectionnez Legacy camera pour activer l'ancienne pile de caméras.
4. Redémarrez l'appareil Raspberry Pi.

Points de terminaison et ports

Par défaut, ce composant utilise un script d'installation pour installer les packages à l'aide des `pip` commandes `apt` `yumbrew`, et, en fonction de la plate-forme utilisée par le périphérique principal. Ce composant doit être capable d'exécuter des requêtes sortantes vers différents index de packages et référentiels pour exécuter le script d'installation. Pour autoriser le trafic sortant de ce composant à passer par un proxy ou un pare-feu, vous devez identifier les points de terminaison des index de packages et des référentiels auxquels votre périphérique principal se connecte pour l'installation.

Tenez compte des points suivants lorsque vous identifiez les points de terminaison requis pour le script d'installation de ce composant :

- Les points de terminaison dépendent de la plate-forme de l'appareil principal. Par exemple, un périphérique principal qui exécute Ubuntu utilise apt plutôt que yum ou brew. En outre, les appareils qui utilisent le même index de packages peuvent avoir des listes de sources différentes, de sorte qu'ils peuvent récupérer des packages à partir de différents référentiels.
- Les points de terminaison peuvent différer entre plusieurs appareils utilisant le même index de packages, car chaque appareil possède ses propres listes de sources qui définissent où récupérer les packages.
- Les points de terminaison peuvent changer au fil du temps. Chaque index URLs de package fournit les référentiels dans lesquels vous téléchargez des packages, et le propriétaire d'un package peut modifier ce que fournit URLs l'index de package.

Pour plus d'informations sur les dépendances installées par ce composant et sur la façon de désactiver le script d'installation, consultez le paramètre [UseInstaller](#) de configuration.

Pour plus d'informations sur les points de terminaison et les ports requis pour le fonctionnement de base, consultez [Autoriser le trafic des appareils via un proxy ou un pare-feu](#).

Dépendances

Lorsque vous déployez un composant, il déploie AWS IoT Greengrass également des versions compatibles de ses dépendances. Cela signifie que vous devez satisfaire aux exigences relatives au composant et à toutes ses dépendances pour réussir le déploiement du composant. Cette section répertorie les dépendances des [versions publiées](#) de ce composant et les contraintes de version sémantiques qui définissent les versions des composants pour chaque dépendance. Vous pouvez également consulter les dépendances de chaque version du composant dans la [AWS IoT Greengrass console](#). Sur la page de détails du composant, recherchez la liste des dépendances.

2.5.14 and 2.5.15

Le tableau suivant répertorie les dépendances pour les versions 2.5.14 et 2.5.15 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,13.0	Flexible

2.5.13

Le tableau suivant répertorie les dépendances pour la version 2.5.13 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,12.0	Flexible

2.5.12

Le tableau suivant répertorie les dépendances pour la version 2.5.12 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,11.0	Flexible

2.5.11

Le tableau suivant répertorie les dépendances pour la version 2.5.11 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0,0 <2,1,0	Flexible

2.5.10

Le tableau suivant répertorie les dépendances pour la version 2.5.10 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,9.0	Flexible

2.5.9

Le tableau suivant répertorie les dépendances pour la version 2.5.9 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,8.0	Flexible

2.5.8

Le tableau suivant répertorie les dépendances pour la version 2.5.8 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,7.0	Flexible

2.5.5 - 2.5.7

Le tableau suivant répertorie les dépendances pour les versions 2.5.5 à 2.5.7 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,6.0	Flexible

2.5.3 and 2.5.4

Le tableau suivant répertorie les dépendances pour les versions 2.5.3 et 2.5.4 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,5.0	Flexible

2.5.2

Le tableau suivant répertorie les dépendances pour la version 2.5.2 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,4.0	Flexible

2.5.1

Le tableau suivant répertorie les dépendances pour la version 2.5.1 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,3.0	Flexible

2.5.0

Le tableau suivant répertorie les dépendances pour la version 2.5.0 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,2.0	Flexible

Pour plus d'informations sur les dépendances des composants, consultez la [référence de la recette des composants](#).

Configuration

Ce composant fournit les paramètres de configuration suivants que vous pouvez personnaliser lorsque vous déployez le composant.

MLRootPath

(Facultatif) Le chemin du dossier sur les périphériques principaux Linux où les composants d'inférence lisent les images et écrivent les résultats d'inférence. Vous pouvez modifier cette valeur à n'importe quel emplacement de votre appareil auquel l'utilisateur exécutant ce composant a read/write accès.

Par défaut : `/greengrass/v2/work/variant.TensorFlowLite/greengrass_ml`

WindowsMLRootPath

Cette fonctionnalité est disponible dans la version 1.6.6 et les versions ultérieures de ce composant.

(Facultatif) Le chemin du dossier sur le périphérique principal de Windows où les composants d'inférence lisent les images et écrivent les résultats d'inférence. Vous pouvez modifier cette

valeur à n'importe quel emplacement de votre appareil auquel l'utilisateur exécutant ce composant a read/write accès.

Par défaut : `C:\greengrass\v2\work\variant.DLR\greengrass_ml`

UseInstaller

(Facultatif) Valeur de chaîne qui définit s'il faut utiliser le script d'installation dans ce composant pour installer TensorFlow Lite et ses dépendances. Les valeurs prises en charge sont `true` et `false`.

Définissez cette valeur sur `false` si vous souhaitez utiliser un script personnalisé pour l'installation de TensorFlow Lite ou si vous souhaitez inclure des dépendances d'exécution dans une image Linux prédéfinie. Pour utiliser ce composant avec les composants d'inférence TensorFlow Lite AWS fournis, installez les bibliothèques suivantes, y compris les dépendances, et mettez-les à la disposition de l'utilisateur du système, par exemple celui `ggc_user` qui exécute les composants ML.

- [Python](#) 3.8 ou version ultérieure, y compris `pip` pour votre version de Python
- [TensorFlow Lite v2.5.0](#)
- [NumPy](#)
- [OpenCV-Python](#)
- [Kit SDK des appareils AWS IoT v2 pour Python](#)
- [AWS Common Runtime \(CRT\) Python](#)
- [Picamera](#) (pour les appareils Raspberry Pi)
- [awscammodule](#) (pour AWS DeepLens appareils)
- `LibGL` (pour les appareils Linux)

Par défaut : `true`

Utilisation

Utilisez ce composant avec le paramètre `UseInstaller` de configuration défini sur `true` pour installer TensorFlow Lite et ses dépendances sur votre appareil. Le composant configure un environnement virtuel sur votre appareil qui inclut l'OpenCV NumPy et les bibliothèques requises pour Lite. TensorFlow

Note

Le script d'installation de ce composant installe également les dernières versions des bibliothèques système supplémentaires requises pour configurer l'environnement virtuel sur votre appareil et pour utiliser l'infrastructure d'apprentissage automatique installée. Cela peut mettre à niveau les bibliothèques système existantes sur votre appareil. Consultez le tableau suivant pour obtenir la liste des bibliothèques que ce composant installe pour chaque système d'exploitation pris en charge. Si vous souhaitez personnaliser ce processus d'installation, définissez le paramètre `UseInstaller` de configuration sur `false` et développez votre propre script d'installation.

Plateforme	Bibliothèques installées sur le système de l'appareil	Bibliothèques installées dans l'environnement virtuel
Armv7l	build-essential , cmake, ca-certificates , git	setuptools , wheel
Amazon Linux 2	mesa-libGL	Aucune
Ubuntu	wget	Aucune

Lorsque vous déployez votre composant d'inférence, ce composant d'exécution vérifie d'abord si TensorFlow Lite et ses dépendances sont déjà installés sur votre appareil. Si ce n'est pas le cas, le composant d'exécution les installe pour vous.

Fichier journal local

Ce composant utilise le fichier journal suivant.

Linux

```
/greengrass/v2/logs/variant.TensorFlowLite.log
```

Windows

```
C:\greengrass\v2\logs\variant.TensorFlowLite.log
```

Pour consulter les journaux de ce composant

- Exécutez la commande suivante sur le périphérique principal pour afficher le fichier journal de ce composant en temps réel. Remplacez `/greengrass/v2` ou `C:\greengrass\v2` par le chemin d'accès au dossier AWS IoT Greengrass racine.

Linux

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

Windows (PowerShell)

```
Get-Content C:\greengrass\v2\logs\variant.TensorFlowLite.log -Tail 10 -Wait
```

Journal des modifications

Le tableau suivant décrit les modifications apportées à chaque version du composant.

Version	Modifications
2.5,15	Version mise à jour pour la version 2.12.5 de Greengrass Nucleus.
2.5,14	Version mise à jour pour la version 2.12.0 de Greengrass Nucleus.
2.5,13	Version mise à jour pour la version 2.11.0 de Greengrass Nucleus.
2.5,12	Version mise à jour pour la version 2.10.0 de Greengrass Nucleus.
2.5.11	Version mise à jour pour la version 2.9.0 de Greengrass Nucleus.
2.5.10	Version mise à jour pour la version 2.8.0 de Greengrass Nucleus.
2.5.9	Version mise à jour pour la version 2.7.0 de Greengrass Nucleus.
2.5.8	Version mise à jour pour la version 2.6.0 de Greengrass Nucleus.
2.5.7	Corrections de bugs et améliorations Met à jour le script <code>UseInstaller</code> d'installation pour installer LibGL, qui n'est pas disponible par défaut sur certaines plateformes Linux.

Version	Modifications
	Met à jour le script d'UseInstaller installation pour toujours utiliser Python 3.9 dans l'environnement virtuel de ce composant. Cette modification permet de garantir la compatibilité avec les autres bibliothèques.
2.5.6	Corrections de bugs et améliorations - Met à jour ce composant pour installer le dernier correctif de TensorFlow Lite 2.5.0 (<code>tf-lite-runtime-2.5.0.post1</code>), afin que vous puissiez utiliser ce composant avec Python 3.9. Si ce composant ne parvient pas à installer ce correctif, il s'installe à la <code>tf-lite-runtime-2.5.0</code> place. - Met à jour ce composant pour réinstaller les installations existantes de TensorFlow Lite et ses dépendances. Cette modification permet de garantir que le périphérique principal exécute des versions compatibles de TensorFlow Lite et de ses dépendances.
2.5.5	Nouvelles fonctionnalités - Ajoute la prise en charge des appareils principaux qui exécutent Windows. - Ajoute le nouveau paramètre <code>WindowsMLRootPath</code> de configuration que vous pouvez utiliser pour configurer le dossier des résultats d'inférence sur les appareils principaux de Windows.
2.5.4	Nouvelles fonctionnalités Ajoute le nouveau paramètre <code>UseInstaller</code> de configuration qui permet de désactiver le script d'installation dans ce composant.
2.5.3	Version mise à jour pour la version 2.4.0 de Greengrass Nucleus.
2.5.2	Version mise à jour pour la version 2.3.0 de Greengrass Nucleus.
2.5.1	Version mise à jour pour la version 2.2.0 de Greengrass Nucleus.
2.5.0	Première version.

Adaptateur de protocole Modbus-RTU

Le composant adaptateur de protocole Modbus-RTU (`aws.greengrass.Modbus`) interroge les informations des appareils Modbus RTU locaux.

Pour demander des informations à un appareil Modbus RTU local avec ce composant, publiez un message dans la rubrique où ce composant est abonné. Dans le message, spécifiez la demande Modbus RTU à envoyer à un appareil. Ce composant publie ensuite une réponse contenant le résultat de la demande Modbus RTU.

Note

Ce composant fournit des fonctionnalités similaires à celles du connecteur d'adaptateur de protocole Modbus RTU de la version 1. AWS IoT Greengrass Pour plus d'informations, consultez la section [Connecteur d'adaptateur de protocole Modbus RTU](#) dans le guide du développeur AWS IoT Greengrass V1.

Rubriques

- [Versions](#)
- [Type](#)
- [Système d'exploitation](#)
- [Prérequis](#)
- [Dépendances](#)
- [Configuration](#)
- [Données d'entrée](#)
- [Données de sortie](#)
- [Demandes et réponses de Modbus RTU](#)
- [Fichier journal local](#)
- [Licences](#)
- [Journal des modifications](#)

Versions

Les versions de ce composant sont les suivantes :

- 2,1x
- 2,0.x

Type

Ce composant est un composant Lambda () `aws.greengrass.lambda`. [Le noyau Greengrass exécute la fonction Lambda de ce composant à l'aide du composant Lambda Launcher.](#)

Pour de plus amples informations, veuillez consulter [Types de composants](#).

Système d'exploitation

Ce composant ne peut être installé que sur les appareils principaux de Linux.

Prérequis

Ce composant répond aux exigences suivantes :

- Votre appareil principal doit répondre aux exigences pour exécuter les fonctions Lambda. Si vous souhaitez que le périphérique principal exécute des fonctions Lambda conteneurisées, le périphérique doit répondre aux exigences requises. Pour de plus amples informations, veuillez consulter [Exigences relatives à la fonction Lambda](#).
- [Python](#) version 3.7 installé sur le périphérique principal et ajouté à la variable d'environnement PATH.
- Connexion physique entre le périphérique AWS IoT Greengrass principal et les appareils Modbus. Le périphérique principal doit être connecté physiquement au réseau Modbus RTU via un port série, tel qu'un port USB.
- Pour recevoir les données de sortie de ce composant, vous devez fusionner la mise à jour de configuration suivante pour l'[ancien composant routeur d'abonnement](#) (`aws.greengrass.LegacySubscriptionRouter`) lorsque vous déployez ce composant. Cette configuration indique le sujet dans lequel ce composant publie les réponses.

Legacy subscription router v2.1.x

```
{
  "subscriptions": {
    "aws-greengrass-modbus": {
      "id": "aws-greengrass-modbus",
```

```
    "source": "component:aws.greengrass.Modbus",
    "subject": "modbus/adapter/response",
    "target": "cloud"
  }
}
```

Legacy subscription router v2.0.x

```
{
  "subscriptions": {
    "aws-greengrass-modbus": {
      "id": "aws-greengrass-modbus",
      "source": "arn:aws:lambda:region:aws:function:aws-greengrass-
modbus:version",
      "subject": "modbus/adapter/response",
      "target": "cloud"
    }
  }
}
```

- *region* Remplacez-le par Région AWS celui que vous utilisez.
- *version* Remplacez-le par la version de la fonction Lambda exécutée par ce composant. Pour trouver la version de la fonction Lambda, vous devez consulter la recette de la version de ce composant que vous souhaitez déployer. Ouvrez la page de détails de ce composant dans la [AWS IoT Greengrass console](#) et recherchez la paire clé-valeur de la fonction Lambda. Cette paire clé-valeur contient le nom et la version de la fonction Lambda.

Important

Vous devez mettre à jour la version de la fonction Lambda sur l'ancien routeur d'abonnement chaque fois que vous déployez ce composant. Cela garantit que vous utilisez la bonne version de la fonction Lambda pour la version du composant que vous déployez.

Pour de plus amples informations, veuillez consulter [Créer des déploiements](#).

- L'adaptateur de protocole Modbus-RTU est compatible pour fonctionner dans un VPC.

Dépendances

Lorsque vous déployez un composant, il déploie AWS IoT Greengrass également des versions compatibles de ses dépendances. Cela signifie que vous devez satisfaire aux exigences relatives au composant et à toutes ses dépendances pour réussir le déploiement du composant. Cette section répertorie les dépendances des [versions publiées](#) de ce composant et les contraintes de version sémantiques qui définissent les versions des composants pour chaque dépendance. Vous pouvez également consulter les dépendances de chaque version du composant dans la [AWS IoT Greengrass console](#). Sur la page de détails du composant, recherchez la liste des dépendances.

2.1.11

Le tableau suivant répertorie les dépendances pour la version 2.1.11 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,0,0 < 2,16,0$	Stricte
Lanceur Lambda	$\wedge 2,0,0$	Stricte
Environnements d'exécution (runtimes) Lambda	$\wedge 2,0,0$	Flexible
Service d'échange de jetons	$\wedge 2,0,0$	Stricte

2.1.10

Le tableau suivant répertorie les dépendances pour la version 2.1.10 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,0,0 < 2,15,0$	Stricte
Lanceur Lambda	$\wedge 2,0,0$	Stricte
Environnements d'exécution (runtimes) Lambda	$\wedge 2,0,0$	Flexible
Service d'échange de jetons	$\wedge 2,0,0$	Stricte

2.1.9

Le tableau suivant répertorie les dépendances pour la version 2.1.9 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,0.0 < 2,14.0$	Stricte
Lanceur Lambda	$\wedge 2,0.0$	Stricte
Environnements d'exécution (runtimes) Lambda	$\wedge 2,0.0$	Flexible
Service d'échange de jetons	$\wedge 2,0.0$	Stricte

2.1.8

Le tableau suivant répertorie les dépendances pour la version 2.1.8 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,0,0 < 2,13,0$	Stricte
Lanceur Lambda	$\wedge 2,0.0$	Stricte
Environnements d'exécution (runtimes) Lambda	$\wedge 2,0.0$	Flexible
Service d'échange de jetons	$\wedge 2,0.0$	Stricte

2.1.7

Le tableau suivant répertorie les dépendances pour la version 2.1.7 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,0.0 < 2,12.0$	Stricte

Dépendance	Versions compatibles	Type de dépendance
Lanceur Lambda	^2,0.0	Stricte
Environnements d'exécution (runtimes) Lambda	^2,0.0	Flexible
Service d'échange de jetons	^2,0.0	Stricte

2.1.6

Le tableau suivant répertorie les dépendances pour la version 2.1.6 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,11.0	Stricte
Lanceur Lambda	^2,0.0	Stricte
Environnements d'exécution (runtimes) Lambda	^2,0.0	Flexible
Service d'échange de jetons	^2,0.0	Stricte

2.1.4 and 2.1.5

Le tableau suivant répertorie les dépendances pour les versions 2.1.4 et 2.1.5 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0,0 <2,1,0	Stricte
Lanceur Lambda	^2,0.0	Stricte
Environnements d'exécution (runtimes) Lambda	^2,0.0	Flexible
Service d'échange de jetons	^2,0.0	Stricte

2.1.3

Le tableau suivant répertorie les dépendances pour la version 2.1.3 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,9.0	Stricte
Lanceur Lambda	^2,0.0	Stricte
Environnements d'exécution (runtimes) Lambda	^2,0.0	Flexible
Service d'échange de jetons	^2,0.0	Stricte

2.1.2

Le tableau suivant répertorie les dépendances pour la version 2.1.2 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,8.0	Stricte
Lanceur Lambda	^2,0.0	Stricte
Environnements d'exécution (runtimes) Lambda	^2,0.0	Flexible
Service d'échange de jetons	^2,0.0	Stricte

2.1.1

Le tableau suivant répertorie les dépendances pour la version 2.1.1 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,7.0	Stricte

Dépendance	Versions compatibles	Type de dépendance
Lanceur Lambda	^2,0.0	Stricte
Environnements d'exécution (runtimes) Lambda	^2,0.0	Flexible
Service d'échange de jetons	^2,0.0	Stricte

2.0.8 and 2.1.0

Le tableau suivant répertorie les dépendances pour les versions 2.0.8 et 2.1.0 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,6.0	Stricte
Lanceur Lambda	^2,0.0	Stricte
Environnements d'exécution (runtimes) Lambda	^2,0.0	Flexible
Service d'échange de jetons	^2,0.0	Stricte

2.0.7

Le tableau suivant répertorie les dépendances pour la version 2.0.7 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,5.0	Stricte
Lanceur Lambda	^2,0.0	Stricte
Environnements d'exécution (runtimes) Lambda	^2,0.0	Flexible
Service d'échange de jetons	^2,0.0	Stricte

2.0.6

Le tableau suivant répertorie les dépendances pour la version 2.0.6 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,0.0 < 2,4.0$	Stricte
Lanceur Lambda	$\wedge 2,0.0$	Stricte
Environnements d'exécution (runtimes) Lambda	$\wedge 2,0.0$	Flexible
Service d'échange de jetons	$\wedge 2,0.0$	Stricte

2.0.5

Le tableau suivant répertorie les dépendances pour la version 2.0.5 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,0.0 < 2,3.0$	Stricte
Lanceur Lambda	$\wedge 2,0.0$	Stricte
Environnements d'exécution (runtimes) Lambda	$\wedge 2,0.0$	Flexible
Service d'échange de jetons	$\wedge 2,0.0$	Stricte

2.0.4

Le tableau suivant répertorie les dépendances pour la version 2.0.4 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,0.0 < 2,2.0$	Stricte

Dépendance	Versions compatibles	Type de dépendance
Lanceur Lambda	^2,0.0	Stricte
Environnements d'exécution (runtimes) Lambda	^2,0.0	Flexible
Service d'échange de jetons	^2,0.0	Stricte

2.0.3

Le tableau suivant répertorie les dépendances pour la version 2.0.3 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0,3 <2,10	Stricte
Lanceur Lambda	>=10,0	Stricte
Environnements d'exécution (runtimes) Lambda	>=10,0	Flexible
Service d'échange de jetons	>=10,0	Stricte

Pour plus d'informations sur les dépendances des composants, consultez la [référence de la recette des composants](#).

Configuration

Ce composant fournit les paramètres de configuration suivants que vous pouvez personnaliser lorsque vous déployez le composant.

Note

La configuration par défaut de ce composant inclut les paramètres de la fonction Lambda. Nous vous recommandons de modifier uniquement les paramètres suivants pour configurer ce composant sur vos appareils.

v2.1.x

lambdaParams

Objet contenant les paramètres de la fonction Lambda de ce composant. Cet objet contient les informations suivantes :

EnvironmentVariables

Objet contenant les paramètres de la fonction Lambda. Cet objet contient les informations suivantes :

ModbusLocalPort

Le chemin absolu vers le port série physique Modbus sur le périphérique principal, tel que `/dev/ttyS2`.

Pour exécuter ce composant dans un conteneur, vous devez définir ce chemin en tant que périphérique système (`dockerContainerParams.devices`) auquel le composant peut accéder. Ce composant s'exécute dans un conteneur par défaut.

Note

Ce composant doit avoir read/write accès à l'appareil.

ModbusBaudRate

(Facultatif) Une valeur de chaîne qui indique le débit en bauds pour les communications série avec les périphériques Modbus TCP locaux.

Par défaut : 9600

ModbusByteSize

(Facultatif) Une valeur de chaîne qui indique la taille d'un octet lors d'une communication série avec des périphériques Modbus TCP locaux. Choisissez 5, 67, ou 8 bits.

Par défaut : 8

ModbusParity

(Facultatif) Mode de parité à utiliser pour vérifier l'intégrité des données dans les communications série avec les appareils Modbus TCP locaux.

- E— Vérifiez l'intégrité des données avec une parité uniforme.
- O— Vérifiez l'intégrité des données avec une parité impaire.
- N— Ne vérifiez pas l'intégrité des données.

Par défaut : N

ModbusStopBits

(Facultatif) Valeur de chaîne qui indique le nombre de bits indiquant la fin d'un octet lors d'une communication série avec des périphériques Modbus TCP locaux.

Par défaut : 1

containerMode

(Facultatif) Mode de conteneurisation de ce composant. Sélectionnez parmi les options suivantes :

- GreengrassContainer— Le composant s'exécute dans un environnement d'exécution isolé à l'intérieur du AWS IoT Greengrass conteneur.

Si vous spécifiez cette option, vous devez spécifier un périphérique système (`incontainerParams.devices`) pour permettre au conteneur d'accéder au périphérique Modbus.

- NoContainer— Le composant ne s'exécute pas dans un environnement d'exécution isolé.

Par défaut : GreengrassContainer

containerParams

(Facultatif) Objet contenant les paramètres du conteneur pour ce composant. Le composant utilise ces paramètres si vous le spécifiez GreengrassContainer pour `containerMode`.

Cet objet contient les informations suivantes :

memorySize

(Facultatif) La quantité de mémoire (en kilo-octets) à allouer au composant.

La valeur par défaut est de 512 Mo (525 312 Ko).

devices

(Facultatif) Objet qui spécifie les périphériques système auxquels le composant peut accéder dans un conteneur.

 Important

Pour exécuter ce composant dans un conteneur, vous devez spécifier le périphérique système que vous configurez dans la variable d'`ModbusLocalPort` environnement.

Cet objet contient les informations suivantes :

0— Il s'agit d'un index de tableau sous forme de chaîne.

Un objet qui contient les informations suivantes :

`path`

Le chemin d'accès au périphérique système sur le périphérique principal. Elle doit avoir la même valeur que celle pour laquelle vous avez configuré `ModbusLocalPort`.

`permission`

(Facultatif) L'autorisation d'accéder au périphérique système depuis le conteneur. Cette valeur doit être `rw`, ce qui indique que le composant a read/write accès au périphérique système.

Par défaut : `rw`

`addGroupOwner`

(Facultatif) S'il faut ou non ajouter le groupe système qui exécute le composant en tant que propriétaire du périphérique système.

Par défaut : `true`

`pubsubTopics`

(Facultatif) Objet contenant les rubriques auxquelles le composant s'abonne pour recevoir des messages. Vous pouvez spécifier chaque sujet et indiquer si le composant est abonné à des sujets MQTT provenant de sujets locaux AWS IoT Core ou locaux publish/subscribe .

Cet objet contient les informations suivantes :

0— Il s'agit d'un index de tableau sous forme de chaîne.

Un objet qui contient les informations suivantes :

type

(Facultatif) Type de publish/subscribe messagerie utilisé par ce composant pour s'abonner aux messages. Sélectionnez parmi les options suivantes :

- PUB_SUB – Abonnez-vous aux messages locaux de publication/abonnement. Si vous choisissez cette option, le sujet ne peut pas contenir de caractères génériques MQTT. Pour plus d'informations sur la façon d'envoyer des messages à partir d'un composant personnalisé lorsque vous spécifiez cette option, consultez [Publier/souscrire des messages locaux](#).
- IOT_CORE— Abonnez-vous aux messages AWS IoT Core MQTT. Si vous choisissez cette option, le sujet peut contenir des caractères génériques MQTT. Pour plus d'informations sur la façon d'envoyer des messages à partir de composants personnalisés lorsque vous spécifiez cette option, consultez [Publier/souscrire AWS IoT Core des messages MQTT](#).

Par défaut : PUB_SUB

topic

(Facultatif) Rubrique à laquelle le composant s'abonne pour recevoir des messages. Si vous spécifiez IotCore pour type, vous pouvez utiliser des caractères génériques MQTT (+et#) dans cette rubrique.

Exemple Exemple : mise à jour de fusion de configuration (mode conteneur)

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "ModbusLocalPort": "/dev/ttyS2"
    }
  },
  "containerMode": "GreengrassContainer",
  "containerParams": {
    "devices": {
      "0": {
        "path": "/dev/ttyS2",
        "permission": "rw",
        "addGroupOwner": true
      }
    }
  }
}
```

```
}
```

Exemple Exemple : mise à jour de fusion de configuration (pas de mode conteneur)

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "ModbusLocalPort": "/dev/ttyS2"
    }
  },
  "containerMode": "NoContainer"
}
```

v2.0.x

lambdaParams

Objet contenant les paramètres de la fonction Lambda de ce composant. Cet objet contient les informations suivantes :

EnvironmentVariables

Objet contenant les paramètres de la fonction Lambda. Cet objet contient les informations suivantes :

ModbusLocalPort

Le chemin absolu vers le port série physique Modbus sur le périphérique principal, tel que `/dev/ttyS2`.

Pour exécuter ce composant dans un conteneur, vous devez définir ce chemin en tant que périphérique système (`containerParams.devices`) auquel le composant peut accéder. Ce composant s'exécute dans un conteneur par défaut.

Note

Ce composant doit avoir read/write accès à l'appareil.

containerMode

(Facultatif) Mode de conteneurisation de ce composant. Sélectionnez parmi les options suivantes :

- **GreengrassContainer**— Le composant s'exécute dans un environnement d'exécution isolé à l'intérieur du AWS IoT Greengrass conteneur.

Si vous spécifiez cette option, vous devez spécifier un périphérique système (incontainerParams.devices) pour permettre au conteneur d'accéder au périphérique Modbus.

- **NoContainer**— Le composant ne s'exécute pas dans un environnement d'exécution isolé.

Par défaut : **GreengrassContainer**

containerParams

(Facultatif) Objet contenant les paramètres du conteneur pour ce composant. Le composant utilise ces paramètres si vous le spécifiez **GreengrassContainer** pour **containerMode**.

Cet objet contient les informations suivantes :

memorySize

(Facultatif) La quantité de mémoire (en kilo-octets) à allouer au composant.

La valeur par défaut est de 512 Mo (525 312 Ko).

devices

(Facultatif) Objet qui spécifie les périphériques système auxquels le composant peut accéder dans un conteneur.

Important

Pour exécuter ce composant dans un conteneur, vous devez spécifier le périphérique système que vous configurez dans la variable d'**ModbusLocalPort** environnement.

Cet objet contient les informations suivantes :

0— Il s'agit d'un index de tableau sous forme de chaîne.

Un objet qui contient les informations suivantes :

path

Le chemin d'accès au périphérique système sur le périphérique principal. Elle doit avoir la même valeur que celle pour laquelle vous avez configuré `ModbusLocalPort`.

permission

(Facultatif) L'autorisation d'accéder au périphérique système depuis le conteneur. Cette valeur doit être `rw`, ce qui indique que le composant a read/write accès au périphérique système.

Par défaut : `rw`

addGroupOwner

(Facultatif) S'il faut ou non ajouter le groupe système qui exécute le composant en tant que propriétaire du périphérique système.

Par défaut : `true`

pubsubTopics

(Facultatif) Objet contenant les rubriques auxquelles le composant s'abonne pour recevoir des messages. Vous pouvez spécifier chaque sujet et indiquer si le composant est abonné à des sujets MQTT provenant de sujets locaux AWS IoT Core ou locaux publish/subscribe .

Cet objet contient les informations suivantes :

0— Il s'agit d'un index de tableau sous forme de chaîne.

Un objet qui contient les informations suivantes :

type

(Facultatif) Type de publish/subscribe messagerie utilisé par ce composant pour s'abonner aux messages. Sélectionnez parmi les options suivantes :

- `PUB_SUB` – Abonnez-vous aux messages locaux de publication/abonnement. Si vous choisissez cette option, le sujet ne peut pas contenir de caractères génériques MQTT. Pour plus d'informations sur la façon d'envoyer des messages à partir d'un composant personnalisé lorsque vous spécifiez cette option, consultez [Publier/souscrire des messages locaux](#).
- `IOT_CORE`— Abonnez-vous aux messages AWS IoT Core MQTT. Si vous choisissez cette option, le sujet peut contenir des caractères génériques MQTT. Pour plus

d'informations sur la façon d'envoyer des messages à partir de composants personnalisés lorsque vous spécifiez cette option, consultez [Publier/souscrire AWS IoT Core des messages MQTT](#).

Par défaut : PUB_SUB

topic

(Facultatif) Rubrique à laquelle le composant s'abonne pour recevoir des messages. Si vous spécifiez IotCore pour type, vous pouvez utiliser des caractères génériques MQTT (+et#) dans cette rubrique.

Exemple Exemple : mise à jour de fusion de configuration (mode conteneur)

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "ModbusLocalPort": "/dev/ttyS2"
    }
  },
  "containerMode": "GreengrassContainer",
  "containerParams": {
    "devices": {
      "0": {
        "path": "/dev/ttyS2",
        "permission": "rw",
        "addGroupOwner": true
      }
    }
  }
}
```

Exemple Exemple : mise à jour de fusion de configuration (pas de mode conteneur)

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "ModbusLocalPort": "/dev/ttyS2"
    }
  },
  "containerMode": "NoContainer"
}
```

Données d'entrée

Ce composant accepte les paramètres de demande Modbus RTU sur le sujet suivant et envoie la demande Modbus RTU à l'appareil. Par défaut, ce composant s'abonne aux publish/subscribe messages locaux. Pour plus d'informations sur la façon de publier des messages vers ce composant à partir de vos composants personnalisés, consultez [Publier/souscrire des messages locaux](#).

Rubrique par défaut (publication/abonnement local) : `modbus/adapter/request`

Le message accepte les propriétés suivantes. Les messages d'entrée doivent être au format JSON.

`request`

Les paramètres de la demande Modbus RTU à envoyer.

La forme du message de demande dépend du type de demande Modbus RTU qu'il représente. Les propriétés suivantes sont requises pour toutes les demandes.

Type : `object` qui contient les informations suivantes :

`operation`

Nom de l'opération à exécuter. Par exemple, spécifiez `ReadCoilsRequest` de lire les bobines sur un périphérique Modbus RTU. Pour plus d'informations sur les opérations prises en charge, consultez [Demandes et réponses de Modbus RTU](#).

Type : `string`

`device`

L'appareil cible de la requête.

Cette valeur doit être un entier compris entre 0 et 247.

Type : `integer`

Les autres paramètres à inclure dans la demande dépendent de l'opération. Ce composant gère le contrôle de [redondance cyclique \(CRC\) afin de vérifier](#) les demandes de données pour vous.

Note

Si votre demande inclut une `address` propriété, vous devez spécifier sa valeur sous forme d'entier. Par exemple, `"address": 1`.

id

ID arbitraire de la demande. Utilisez cette propriété pour associer une demande d'entrée à une réponse de sortie. Lorsque vous spécifiez cette propriété, le composant définit la `id` propriété de l'objet de réponse sur cette valeur.

Type : `string`

Exemple Exemple d'entrée : demande de lecture de bobines

```
{
  "request": {
    "operation": "ReadCoilsRequest",
    "device": 1,
    "address": 1,
    "count": 1
  },
  "id": "MyRequest"
}
```

Données de sortie

Ce composant publie par défaut les réponses sous forme de données de sortie sur le sujet MQTT suivant. Vous devez spécifier cette rubrique `subject` dans la configuration de l'[ancien composant routeur d'abonnement](#). Pour plus d'informations sur la façon de s'abonner à des messages sur ce sujet dans vos composants personnalisés, consultez [Publier/souscrire AWS IoT Core des messages MQTT](#).

Rubrique par défaut (AWS IoT Core MQTT) : `modbus/adapter/response`

La forme du message de réponse dépend de l'opération de demande et de l'état de la réponse. Pour obtenir des exemples, consultez [Exemples de demandes et de réponses](#).

Chaque réponse inclut les propriétés suivantes :

response

Réponse de l'appareil Modbus RTU.

Type : `object` qui contient les informations suivantes :

status

Le statut d'une demande. Le statut peut avoir l'une des valeurs suivantes :

- **Success**— La demande était valide, le composant l'a envoyée au réseau Modbus RTU et le réseau Modbus RTU a renvoyé une réponse.
- **Exception**— La demande était valide, le composant l'a envoyée au réseau Modbus RTU et le réseau Modbus RTU a renvoyé une exception. Pour de plus amples informations, veuillez consulter [Statut de la réponse : Exception](#).
- **No Response**— La demande n'était pas valide et le composant a détecté l'erreur avant d'envoyer la demande au réseau Modbus RTU. Pour de plus amples informations, veuillez consulter [Statut de la réponse : Pas de réponse](#).

operation

Opération demandée par le composant.

device

L'appareil sur lequel le composant a envoyé la demande.

payload

Réponse de l'appareil Modbus RTU. Si tel status est le cas **No Response**, cet objet contient uniquement une `error` propriété avec la description de l'erreur (par exemple, `[Input/Output] No Response received from the remote unit`).

id

L'ID de la demande, que vous pouvez utiliser pour identifier quelle réponse correspond à quelle demande.

Note

Une réponse pour une opération d'écriture est simplement un écho de la demande. Bien que les réponses écrites n'incluent pas d'informations pertinentes, il est recommandé de vérifier le statut de la réponse pour voir si la demande aboutit ou échoue.

Exemple Exemple de sortie : réussite

```
{  
  "response" : {
```

```
{
  "status" : "success",
  "device": 1,
  "operation": "ReadCoilsRequest",
  "payload": {
    "function_code": 1,
    "bits": [1]
  }
},
"id" : "MyRequest"
}
```

Exemple 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" : "MyRequest"
}
```

Pour obtenir plus d'exemples, consultez [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

Nom de l'opération dans la demande	Code de fonction dans la réponse
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.

Bobines de lecture

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"
}
```

Lire les 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"
}
```

Lire les registres de détention

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"
}
```

Lire les registres d'entrée

Exemple de requête :

```
{
  "request": {
    "operation": "ReadInputRegistersRequest",
    "device": 1,
    "address": 1,
    "count": 1
  },
  "id": "TestRequest"
}
```

Écrire une seule bobine

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"
}
```

Écrire un registre unique

Exemple de requête :

```
{
  "request": {
    "operation": "WriteSingleRegisterRequest",
    "device": 1,
    "address": 1,
    "value": 1
  },
  "id": "TestRequest"
}
```

Écrire plusieurs bobines

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"
}
```

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"
}
```

Masque, écriture, registre

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"
}
```

```
}
```

Lecture et écriture de 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

La réponse inclut les registres que le composant lit.

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.
- L'error contient la raison 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 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"
}
```

Fichier journal local

Ce composant utilise le fichier journal suivant.

```
/greengrass/v2/logs/aws.greengrass.Modbus.log
```

Pour consulter les journaux de ce composant

- Exécutez la commande suivante sur le périphérique principal pour afficher le fichier journal de ce composant en temps réel. Remplacez */greengrass/v2* par le chemin d'accès au dossier AWS IoT Greengrass racine.

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

Licences

Ce composant inclut les logiciels/licences tiers suivants :

- [Licence pymodbus/BSD](#)
- [Licence pyserial](#) /BSD

Ce composant 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 composant.

Version	Modifications
2.1.11	Version mise à jour pour la version 2.15.0 de Greengrass Nucleus.
2.1.10	Version mise à jour pour la version 2.14.0 de Greengrass Nucleus.
2.1.9	Version mise à jour pour la version 2.13.0 de Greengrass Nucleus.
2.1.8	Version mise à jour pour la version 2.12.0 de Greengrass Nucleus.
2.1.7	Version mise à jour pour la version 2.11.0 de Greengrass Nucleus.
2.1.6	Version mise à jour pour la version 2.10.0 de Greengrass Nucleus.
2.1.5	Corrections de bogues et améliorations • Résout un problème lié à l'ReadDiscreteInput opération.
2.1.4	Version mise à jour pour la version 2.9.0 de Greengrass Nucleus.
2.1.3	Version mise à jour pour la version 2.8.0 de Greengrass Nucleus.
2.1.2	Version mise à jour pour la version 2.7.0 de Greengrass Nucleus.
2.1.1	Version mise à jour pour la version 2.6.0 de Greengrass Nucleus.

Version	Modifications
2.1.0	Nouvelles fonctionnalités • Ajoute les <code>ModbusStopBits</code> options <code>ModbusBaudRate</code> <code>ModbusByteSize</code> ,<code>ModbusParity</code> , et que vous pouvez spécifier pour configurer la communication série avec les appareils Modbus RTU.
2.0.8	Version mise à jour pour la version 2.5.0 de Greengrass Nucleus.
2.0.7	Version mise à jour pour la version 2.4.0 de Greengrass Nucleus.
2.0.6	Version mise à jour pour la version 2.3.0 de Greengrass Nucleus.
2.0.5	Version mise à jour pour la version 2.2.0 de Greengrass Nucleus.
2.0.4	Version mise à jour pour la version 2.1.0 de Greengrass Nucleus.
2.0.3	Première version.

Pont MQTT

Le composant du pont MQTT (`aws.greengrass.clientdevices.mqtt.Bridge`) relaie les messages MQTT entre les appareils clients, les publications et abonnements locaux de Greengrass, et. AWS IoT Core Vous pouvez utiliser ce composant pour agir sur les messages MQTT des appareils clients dans des composants personnalisés et pour synchroniser les appareils clients avec le AWS Cloud.

Note

Les appareils clients sont des appareils IoT locaux qui se connectent à un appareil principal de Greengrass pour envoyer des messages MQTT et des données à traiter. Pour de plus amples informations, veuillez consulter [Interagissez avec les appareils IoT locaux](#).

Vous pouvez utiliser ce composant pour relayer des messages entre les courtiers de messages suivants :

- MQTT local — Le courtier MQTT local gère les messages entre les appareils clients et un périphérique principal.
- Local publish/subscribe — Le courtier de messages Greengrass local gère les messages entre les composants d'un appareil principal. Pour plus d'informations sur la façon d'interagir avec ces messages dans les composants Greengrass, consultez. [Publier/souscrire des messages locaux](#)
- AWS IoT Core — Le broker AWS IoT Core MQTT gère les messages entre les appareils IoT et les AWS Cloud destinations. Pour plus d'informations sur la façon d'interagir avec ces messages dans les composants Greengrass, consultez. [Publier/souscrire AWS IoT Core des messages MQTT](#)

Note

Le pont MQTT utilise QoS 1 pour publier et s' AWS IoT Core abonner, même lorsqu'un appareil client utilise QoS 0 pour publier et s'abonner au courtier MQTT local. Par conséquent, vous pouvez observer une latence supplémentaire lorsque vous relayez des messages MQTT depuis des appareils clients sur le broker MQTT local vers. AWS IoT Core Pour plus d'informations sur la configuration MQTT sur les appareils principaux, consultez [Configurer les délais d'expiration et les paramètres de cache du MQTT](#).

Rubriques

- [Versions](#)
- [Type](#)
- [Système d'exploitation](#)
- [Prérequis](#)
- [Dépendances](#)
- [Configuration](#)
- [Fichier journal local](#)
- [Journal des modifications](#)

Versions

Les versions de ce composant sont les suivantes :

- 2.3.x

- 2.2.x
- 2,1x
- 2,0.x

Type

Ce composant est un composant de plugin (`aws.greengrass.plugin`). Le [noyau Greengrass](#) exécute ce composant dans la même machine virtuelle Java (JVM) que le noyau. Le noyau redémarre lorsque vous modifiez la version de ce composant sur le périphérique principal.

Ce composant utilise le même fichier journal que le noyau Greengrass. Pour de plus amples informations, veuillez consulter [AWS IoT Greengrass Journaux de surveillance](#).

Pour de plus amples informations, veuillez consulter [Types de composants](#).

Système d'exploitation

Ce composant peut être installé sur les appareils principaux qui exécutent les systèmes d'exploitation suivants :

- Linux
- Windows

Prérequis

Ce composant répond aux exigences suivantes :

- Si vous configurez le composant broker MQTT du périphérique principal pour utiliser un port autre que le port par défaut 8883, vous devez utiliser le pont MQTT v2.1.0 ou version ultérieure. Configurez-le pour qu'il se connecte au port sur lequel le broker opère.
- Le composant de pont MQTT peut être exécuté dans un VPC.

Dépendances

Lorsque vous déployez un composant, il déploie AWS IoT Greengrass également des versions compatibles de ses dépendances. Cela signifie que vous devez satisfaire aux exigences relatives au composant et à toutes ses dépendances pour réussir le déploiement du composant. Cette

section répertorie les dépendances des [versions publiées](#) de ce composant et les contraintes de version sémantiques qui définissent les versions des composants pour chaque dépendance. Vous pouvez également consulter les dépendances de chaque version du composant dans la [AWS IoT Greengrass console](#). Sur la page de détails du composant, recherchez la liste des dépendances.

2.3.2

Le tableau suivant répertorie les dépendances pour la version 2.3.2 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Authentification de l'appareil client	$\geq 2,2,0$ $< 2,6,0$	Stricte

2.3.0 and 2.3.1

Le tableau suivant répertorie les dépendances pour les versions 2.3.0 et 2.3.1 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Authentification de l'appareil client	$\geq 2,2,0$ $< 2,5,0$	Stricte

2.2.5 and 2.2.6

Le tableau suivant répertorie les dépendances pour les versions 2.2.5 et 2.2.6 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Authentification de l'appareil client	$\geq 2,2,0$ $< 2,5,0$	Stricte

2.2.3 and 2.2.4

Le tableau suivant répertorie les dépendances pour les versions 2.2.3 et 2.2.4 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Authentification de l'appareil client	$\geq 2,2,0$ $< 2,4,0$	Stricte

2.2.0 – 2.2.2

Le tableau suivant répertorie les dépendances pour les versions 2.2.0 à 2.2.2 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Authentification de l'appareil client	$\geq 2,2,0$ $< 2,3,0$	Stricte

2.1.1

Le tableau suivant répertorie les dépendances pour la version 2.1.1 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Authentification de l'appareil client	$\geq 2,0,0$ $< 2,2,0$	Stricte

2.0.0 to 2.1.0

Le tableau suivant répertorie les dépendances pour les versions 2.0.0 à 2.1.0 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Authentification de l'appareil client	$\geq 2,0,0$ $< 2,1,0$	Stricte

Pour plus d'informations sur les dépendances des composants, consultez la [référence de la recette des composants](#).

Configuration

Ce composant fournit les paramètres de configuration suivants que vous pouvez personnaliser lorsque vous déployez le composant.

2.3.0 – 2.3.2

`mqttTopicMapping`

Les mappages de sujets que vous souhaitez relier. Ce composant s'abonne aux messages du sujet source et publie les messages qu'il reçoit dans le sujet de destination. Chaque mappage de rubrique définit le sujet, le type de source et le type de destination.

Cet objet contient les informations suivantes :

`topicMappingNameKey`

Nom de ce mappage de rubriques. *`topicMappingNameKey`* Remplacez-le par un nom qui vous aide à identifier ce mappage de rubriques.

Cet objet contient les informations suivantes :

`topic`

Le sujet ou le filtre de sujet permettant de faire le lien entre les courtiers source et cible.

Vous pouvez utiliser les caractères génériques de sujet + et # MQTT pour relayer des messages sur tous les sujets correspondant à un filtre de sujet. Pour plus d'informations, consultez les [rubriques relatives au MQTT](#) dans le Guide du AWS IoT Core développeur.

Note

[Pour utiliser des caractères génériques de sujet MQTT avec le courtier Pubsub source, vous devez utiliser la version 2.6.0 ou ultérieure du composant Greengrass nucleus.](#)

`targetTopicPrefix`

Le préfixe à ajouter au sujet cible lorsque ce composant relaie le message.

source

Le courtier de messages source. Sélectionnez parmi les options suivantes :

- LocalMqtt— Le broker MQTT local où les appareils clients communiquent.
- Pubsub— Le courtier de publish/subscribe messages Greengrass local.
- IotCore— Le courtier de messages AWS IoT Core MQTT.

Note

Le pont MQTT utilise QoS 1 pour publier et s' AWS IoT Core abonner, même lorsqu'un appareil client utilise QoS 0 pour publier et s'abonner au courtier MQTT local. Par conséquent, vous pouvez observer une latence supplémentaire lorsque vous relayez des messages MQTT depuis des appareils clients sur le broker MQTT local vers. AWS IoT Core Pour plus d'informations sur la configuration MQTT sur les appareils principaux, consultez [Configurer les délais d'expiration et les paramètres de cache du MQTT](#).

sourceet target doit être différent.

target

Le courtier de messages cible. Sélectionnez parmi les options suivantes :

- LocalMqtt— Le broker MQTT local où les appareils clients communiquent.
- Pubsub— Le courtier de publish/subscribe messages Greengrass local.
- IotCore— Le courtier de messages AWS IoT Core MQTT.

Note

Le pont MQTT utilise QoS 1 pour publier et s' AWS IoT Core abonner, même lorsqu'un appareil client utilise QoS 0 pour publier et s'abonner au courtier MQTT local. Par conséquent, vous pouvez observer une latence supplémentaire lorsque vous relayez des messages MQTT depuis des appareils clients sur le broker MQTT local vers. AWS IoT Core Pour plus d'informations sur la configuration MQTT sur les appareils principaux, consultez [Configurer les délais d'expiration et les paramètres de cache du MQTT](#).

source et target doit être différent.

mqtt5 RouteOptions

(Facultatif) Fournit des options pour configurer les mappages de rubriques afin de relier les messages entre le sujet source et le sujet de destination.

Cet objet contient les informations suivantes :

mqtt5RouteOptionsNameKey

Nom des options d'itinéraire pour le mappage d'un sujet. Remplacez *mqtt5RouteOptionsNameKey* par la correspondance *topicMappingNameKey* définie dans le mqttTopicMapping champ.

Cet objet contient les informations suivantes :

Pas de local

(Facultatif) Lorsqu'il est activé, le pont ne transmet pas de messages sur un sujet qu'il a lui-même publié. Utilisez-le pour éviter les boucles, comme suit :

```
{
  "mqtt5RouteOptions": {
    "toIoTCore": {
      "noLocal": true
    }
  },
  "mqttTopicMapping": {
    "toIoTCore": {
      "topic": "device",
      "source": "LocalMqtt",
      "target": "IotCore"
    },
    "toLocal": {
      "topic": "device",
      "source": "IotCore",
      "target": "LocalMqtt"
    }
  }
}
```

noLocal n'est pris en charge que pour les itinéraires où le source est LocalMqtt.

Valeur par défaut : false

`retainAsPublished`

(Facultatif) Lorsque cette option est activée, les messages transférés par le pont ont le même `retain` indicateur que les messages publiés au courtier pour cette route.

`retainAsPublished` n'est pris en charge que pour les itinéraires où le `source` est `LocalMqtt`.

Valeur par défaut : false

`mqtt`

(Facultatif) Paramètres du protocole MQTT pour communiquer avec le courtier local.

`version`

(Facultatif) Version du protocole MQTT utilisée par le pont pour communiquer avec le courtier local. Doit être identique à la version MQTT sélectionnée dans la configuration du noyau.

Choisissez parmi les options suivantes :

- `mqtt3`
- `mqtt5`

Vous devez déployer un broker MQTT lorsque le `target` champ `source` ou de `mqttTopicMapping` objet est défini sur `LocalMqtt`. Si vous choisissez `mqtt5` cette option, vous devez utiliser le [Courtier MQTT 5 \(EMQX\)](#).

Par défaut : `mqtt3`

`ackTimeoutSeconds`

(Facultatif) Intervalle de temps nécessaire pour attendre les paquets PUBACK, SUBACK ou UNSUBACK avant l'échec de l'opération.

Par défaut : 60

`connAckTimeoutMme`

(Facultatif) Intervalle de temps pendant lequel un paquet CONNACK doit être attendu avant d'arrêter la connexion.

Par défaut : 20000 (20 secondes)

pingTimeoutMs

(Facultatif) Durée en millisecondes pendant laquelle le pont attend de recevoir un message PINGACK du courtier local. Si le délai d'attente dépasse le délai imparti, le pont se ferme puis rouvre la connexion MQTT. Cette valeur doit être inférieure à keepAliveTimeoutSeconds.

Par défaut : 30000 (30 secondes)

keepAliveTimeoutSeconds

(Facultatif) Durée en secondes entre chaque message PING envoyé par le pont pour maintenir la connexion MQTT active. Cette valeur doit être supérieure à pingTimeoutMs.

Par défaut : 60

maxReconnectDelayMme

(Facultatif) Durée maximale en secondes pendant laquelle MQTT se reconnecte.

Par défaut : 30000 (30 secondes)

minReconnectDelayMme

(Facultatif) Durée minimale en secondes pendant laquelle MQTT se reconnecte.

Recevez au maximum

(Facultatif) Le nombre maximum de paquets QoS1 non reconnus que le pont peut envoyer.

Par défaut : 100

maximumPacketSize

Le nombre maximum d'octets que le client acceptera pour un paquet MQTT.

Par défaut : null (aucune limite)

sessionExpiryInterval

(Facultatif) Durée en secondes pendant laquelle vous pouvez demander la durée d'une session entre le pont et le courtier local.

Par défaut : 4294967295 (la session n'expire jamais)

brokerUri

(Facultatif) L'URI du broker MQTT local. Vous devez spécifier ce paramètre si vous configurez le broker MQTT pour utiliser un port différent du port par défaut 8883. Utilisez le format suivant, et remplacez-le *port* par le port sur lequel le broker MQTT fonctionne : `ssl://localhost:port`

Par défaut : `ssl://localhost:8883`

startupTimeoutSeconds

(Facultatif) Durée maximale en secondes pendant laquelle le composant démarre. L'état du composant change BROKEN s'il dépasse ce délai.

Par défaut : 120

Exemple Exemple : mise à jour de la fusion de configurations

L'exemple de mise à jour de configuration suivant spécifie les éléments suivants :

- Transférez les messages des appareils clients vers AWS IoT Core des sujets correspondant au filtre de `clients/+/hello/world` sujets.
- Transférez les messages des appareils clients vers `publish/subscribe` le local sur des sujets correspondant au filtre de `clients/+/detections` sujet, et ajoutez le `events/input/` préfixe au sujet cible. Le sujet cible obtenu correspond au filtre de `events/input/clients/+/detections` sujet.
- Transférez les messages des appareils clients vers AWS IoT Core des sujets correspondant au filtre de `clients/+/status` sujet et ajoutez le `$aws/rules/StatusUpdateRule/` préfixe au sujet cible. Cet exemple transmet ces messages directement à une [AWS IoT règle](#) nommée pour réduire les coûts `StatusUpdateRule` à l'aide de [Basic Ingest](#).

```
{
  "mqttTopicMapping": {
    "ClientDeviceHelloWorld": {
      "topic": "clients/+/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    },
    "ClientDeviceEvents": {
```

```
    "topic": "clients/+/detections",
    "targetTopicPrefix": "events/input/",
    "source": "LocalMqtt",
    "target": "Pubsub"
  },
  "ClientDeviceCloudStatusUpdate": {
    "topic": "clients/+/status",
    "targetTopicPrefix": "$aws/rules/StatusUpdateRule/",
    "source": "LocalMqtt",
    "target": "IotCore"
  }
}
```

Exemple Exemple : Configuration de MQTT 5

L'exemple de configuration suivant met à jour les éléments suivants :

- Permet au pont d'utiliser le protocole MQTT 5 avec le courtier local.
- Configure la conservation du MQTT en tant que paramètre publié pour le mappage des ClientDeviceHelloWorld rubriques.

```
{
  "mqttTopicMapping": {
    "ClientDeviceHelloWorld": {
      "topic": "clients/+/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    }
  },
  "mqtt5RouteOptions": {
    "ClientDeviceHelloWorld": {
      "retainAsPublished": true
    }
  },
  "mqtt": {
    "version": "mqtt5"
  }
}
```

2.2.6

mqttTopicMapping

Les mappages de sujets que vous souhaitez relier. Ce composant s'abonne aux messages du sujet source et publie les messages qu'il reçoit dans le sujet de destination. Chaque mappage de rubrique définit le sujet, le type de source et le type de destination.

Cet objet contient les informations suivantes :

topicMappingNameKey

Nom de ce mappage de rubriques. *topicMappingNameKey* Remplacez-le par un nom qui vous aide à identifier ce mappage de rubriques.

Cet objet contient les informations suivantes :

topic

Le sujet ou le filtre de sujet permettant de faire le lien entre les courtiers source et cible.

Vous pouvez utiliser les caractères génériques de sujet + et # MQTT pour relayer des messages sur tous les sujets correspondant à un filtre de sujet. Pour plus d'informations, consultez les [rubriques relatives au MQTT](#) dans le Guide du AWS IoT Core développeur.

Note

[Pour utiliser des caractères génériques de sujet MQTT avec le courtier Pubsub source, vous devez utiliser la version 2.6.0 ou ultérieure du composant Greengrass nucleus.](#)

targetTopicPrefix

Le préfixe à ajouter au sujet cible lorsque ce composant relaie le message.

source

Le courtier de messages source. Sélectionnez parmi les options suivantes :

- LocalMqtt— Le broker MQTT local où les appareils clients communiquent.
- Pubsub— Le courtier de publish/subscribe messages Greengrass local.
- IotCore— Le courtier de messages AWS IoT Core MQTT.

 Note

Le pont MQTT utilise QoS 1 pour publier et s' AWS IoT Core abonner, même lorsqu'un appareil client utilise QoS 0 pour publier et s'abonner au courtier MQTT local. Par conséquent, vous pouvez observer une latence supplémentaire lorsque vous relayez des messages MQTT depuis des appareils clients sur le broker MQTT local vers. AWS IoT Core Pour plus d'informations sur la configuration MQTT sur les appareils principaux, consultez [Configurer les délais d'expiration et les paramètres de cache du MQTT](#).

source et target doit être différent.

target

Le courtier de messages cible. Sélectionnez parmi les options suivantes :

- LocalMqtt— Le broker MQTT local où les appareils clients communiquent.
- Pubsub— Le courtier de publish/subscribe messages Greengrass local.
- IotCore— Le courtier de messages AWS IoT Core MQTT.

 Note

Le pont MQTT utilise QoS 1 pour publier et s' AWS IoT Core abonner, même lorsqu'un appareil client utilise QoS 0 pour publier et s'abonner au courtier MQTT local. Par conséquent, vous pouvez observer une latence supplémentaire lorsque vous relayez des messages MQTT depuis des appareils clients sur le broker MQTT local vers. AWS IoT Core Pour plus d'informations sur la configuration MQTT sur les appareils principaux, consultez [Configurer les délais d'expiration et les paramètres de cache du MQTT](#).

source et target doit être différent.

brokerUri

(Facultatif) L'URI du broker MQTT local. Vous devez spécifier ce paramètre si vous configurez le broker MQTT pour utiliser un port différent du port par défaut 8883. Utilisez le format

suitant, et remplacez-le *port* par le port sur lequel le broker MQTT fonctionne : `ssl://localhost:port`

Par défaut : `ssl://localhost:8883`

`startupTimeoutSeconds`

(Facultatif) Durée maximale en secondes pendant laquelle le composant démarre. L'état du composant change BROKEN s'il dépasse ce délai.

Par défaut : 120

Exemple Exemple : mise à jour de la fusion de configurations

L'exemple de mise à jour de configuration suivant spécifie les éléments suivants :

- Transférez les messages des appareils clients vers AWS IoT Core des sujets correspondant au filtre de `clients/+hello/world` sujets.
- Transférez les messages des appareils clients vers `publish/subscribe` le local sur des sujets correspondant au filtre de `clients/+detections` sujet, et ajoutez le `events/input/` préfixe au sujet cible. Le sujet cible obtenu correspond au filtre de `events/input/clients/+detections` sujet.
- Transférez les messages des appareils clients vers AWS IoT Core des sujets correspondant au filtre de `clients/+status` sujet et ajoutez le `$aws/rules/StatusUpdateRule/` préfixe au sujet cible. Cet exemple transmet ces messages directement à une [AWS IoT règle](#) nommée pour réduire les coûts `StatusUpdateRule` à l'aide de [Basic Ingest](#).

```
{
  "mqttTopicMapping": {
    "ClientDeviceHelloWorld": {
      "topic": "clients/+hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    },
    "ClientDeviceEvents": {
      "topic": "clients/+detections",
      "targetTopicPrefix": "events/input/",
      "source": "LocalMqtt",
      "target": "Pubsub"
    }
  }
}
```

```
    },  
    "ClientDeviceCloudStatusUpdate": {  
      "topic": "clients/+/status",  
      "targetTopicPrefix": "$aws/rules/StatusUpdateRule/",  
      "source": "LocalMqtt",  
      "target": "IotCore"  
    }  
  }  
}
```

2.2.0 - 2.2.5

mqttTopicMapping

Les mappages de sujets que vous souhaitez relier. Ce composant s'abonne aux messages du sujet source et publie les messages qu'il reçoit dans le sujet de destination. Chaque mappage de rubrique définit le sujet, le type de source et le type de destination.

Cet objet contient les informations suivantes :

topicMappingNameKey

Nom de ce mappage de rubriques. *topicMappingNameKey* Remplacez-le par un nom qui vous aide à identifier ce mappage de rubriques.

Cet objet contient les informations suivantes :

topic

Le sujet ou le filtre de sujet permettant de faire le lien entre les courtiers source et cible.

Vous pouvez utiliser les caractères génériques de sujet + et # MQTT pour relayer des messages sur tous les sujets correspondant à un filtre de sujet. Pour plus d'informations, consultez les [rubriques relatives au MQTT](#) dans le Guide du AWS IoT Core développeur.

Note

[Pour utiliser des caractères génériques de sujet MQTT avec le courtier Pubsub source, vous devez utiliser la version 2.6.0 ou ultérieure du composant Greengrass nucleus.](#)

targetTopicPrefix

Le préfixe à ajouter au sujet cible lorsque ce composant relaie le message.

source

Le courtier de messages source. Sélectionnez parmi les options suivantes :

- LocalMqtt— Le broker MQTT local où les appareils clients communiquent.
- Pubsub— Le courtier de publish/subscribe messages Greengrass local.
- IotCore— Le courtier de messages AWS IoT Core MQTT.

Note

Le pont MQTT utilise QoS 1 pour publier et s' AWS IoT Core abonner, même lorsqu'un appareil client utilise QoS 0 pour publier et s'abonner au courtier MQTT local. Par conséquent, vous pouvez observer une latence supplémentaire lorsque vous relavez des messages MQTT depuis des appareils clients sur le broker MQTT local vers. AWS IoT Core Pour plus d'informations sur la configuration MQTT sur les appareils principaux, consultez [Configurer les délais d'expiration et les paramètres de cache du MQTT](#).

sourceet target doit être différent.

target

Le courtier de messages cible. Sélectionnez parmi les options suivantes :

- LocalMqtt— Le broker MQTT local où les appareils clients communiquent.
- Pubsub— Le courtier de publish/subscribe messages Greengrass local.
- IotCore— Le courtier de messages AWS IoT Core MQTT.

Note

Le pont MQTT utilise QoS 1 pour publier et s' AWS IoT Core abonner, même lorsqu'un appareil client utilise QoS 0 pour publier et s'abonner au courtier MQTT local. Par conséquent, vous pouvez observer une latence supplémentaire lorsque vous relavez des messages MQTT depuis des appareils clients sur le broker MQTT local vers. AWS IoT Core Pour plus d'informations sur la configuration MQTT sur les appareils principaux,

consultez [Configurer les délais d'expiration et les paramètres de cache du MQTT](#).

source et target doit être différent.

brokerUri

(Facultatif) L'URI du broker MQTT local. Vous devez spécifier ce paramètre si vous configurez le broker MQTT pour utiliser un port différent du port par défaut 8883. Utilisez le format suivant, et remplacez-le *port* par le port sur lequel le broker MQTT fonctionne : `ssl://localhost:port`

Par défaut : `ssl://localhost:8883`

Exemple Exemple : mise à jour de la fusion de configurations

L'exemple de mise à jour de configuration suivant spécifie les éléments suivants :

- Transférez les messages des appareils clients vers AWS IoT Core des sujets correspondant au filtre de `clients/+hello/world` sujets.
- Transférez les messages des appareils clients vers `publish/subscribe` le local sur des sujets correspondant au filtre de `clients/+detections` sujet, et ajoutez le `events/input/` préfixe au sujet cible. Le sujet cible obtenu correspond au filtre de `events/input/clients/+detections` sujet.
- Transférez les messages des appareils clients vers AWS IoT Core des sujets correspondant au filtre de `clients/+status` sujet et ajoutez le `$aws/rules/StatusUpdateRule/` préfixe au sujet cible. Cet exemple transmet ces messages directement à une [AWS IoT règle](#) nommée pour réduire les coûts `StatusUpdateRule` à l'aide de [Basic Ingest](#).

```
{
  "mqttTopicMapping": {
    "ClientDeviceHelloWorld": {
      "topic": "clients/+hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    },
    "ClientDeviceEvents": {
      "topic": "clients/+detections",
      "targetTopicPrefix": "events/input/",
    }
  }
}
```

```
    "source": "LocalMqtt",
    "target": "Pubsub"
  },
  "ClientDeviceCloudStatusUpdate": {
    "topic": "clients+/status",
    "targetTopicPrefix": "$aws/rules/StatusUpdateRule/",
    "source": "LocalMqtt",
    "target": "IotCore"
  }
}
```

2.1.x

mqttTopicMapping

Les mappages de sujets que vous souhaitez relier. Ce composant s'abonne aux messages du sujet source et publie les messages qu'il reçoit dans le sujet de destination. Chaque mappage de rubrique définit le sujet, le type de source et le type de destination.

Cet objet contient les informations suivantes :

topicMappingNameKey

Nom de ce mappage de rubriques. *topicMappingNameKey* Remplacez-le par un nom qui vous aide à identifier ce mappage de rubriques.

Cet objet contient les informations suivantes :

topic

Le sujet ou le filtre de sujet permettant de faire le lien entre les courtiers source et cible.

Si vous spécifiez le courtier LocalMqtt ou le courtier IotCore source, vous pouvez utiliser les caractères génériques de sujet + et # MQTT pour relayer des messages sur tous les sujets correspondant à un filtre de sujet. Pour plus d'informations, consultez les [rubriques relatives au MQTT](#) dans le Guide du AWS IoT Core développeur.

source

Le courtier de messages source. Sélectionnez parmi les options suivantes :

- LocalMqtt— Le broker MQTT local où les appareils clients communiquent.
- Pubsub— Le courtier de publish/subscribe messages Greengrass local.
- IotCore— Le courtier de messages AWS IoT Core MQTT.

 Note

Le pont MQTT utilise QoS 1 pour publier et s' AWS IoT Core abonner, même lorsqu'un appareil client utilise QoS 0 pour publier et s'abonner au courtier MQTT local. Par conséquent, vous pouvez observer une latence supplémentaire lorsque vous relayez des messages MQTT depuis des appareils clients sur le broker MQTT local vers. AWS IoT Core Pour plus d'informations sur la configuration MQTT sur les appareils principaux, consultez [Configurer les délais d'expiration et les paramètres de cache du MQTT](#).

source et target doit être différent.

target

Le courtier de messages cible. Sélectionnez parmi les options suivantes :

- LocalMqtt— Le broker MQTT local où les appareils clients communiquent.
- Pubsub— Le courtier de publish/subscribe messages Greengrass local.
- IotCore— Le courtier de messages AWS IoT Core MQTT.

 Note

Le pont MQTT utilise QoS 1 pour publier et s' AWS IoT Core abonner, même lorsqu'un appareil client utilise QoS 0 pour publier et s'abonner au courtier MQTT local. Par conséquent, vous pouvez observer une latence supplémentaire lorsque vous relayez des messages MQTT depuis des appareils clients sur le broker MQTT local vers. AWS IoT Core Pour plus d'informations sur la configuration MQTT sur les appareils principaux, consultez [Configurer les délais d'expiration et les paramètres de cache du MQTT](#).

source et target doit être différent.

brokerUri

(Facultatif) L'URI du broker MQTT local. Vous devez spécifier ce paramètre si vous configurez le broker MQTT pour utiliser un port différent du port par défaut 8883. Utilisez le format

suivant, et remplacez-le *port* par le port sur lequel le broker MQTT fonctionne : `ssl://localhost:port`

Par défaut : `ssl://localhost:8883`

Exemple Exemple : mise à jour de la fusion de configurations

L'exemple de mise à jour de configuration suivant indique de relayer les messages des appareils clients vers AWS IoT Core les `clients/MyClientDevice2/hello/world` rubriques `clients/MyClientDevice1/hello/world` et.

```
{
  "mqttTopicMapping": {
    "ClientDevice1HelloWorld": {
      "topic": "clients/MyClientDevice1/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    },
    "ClientDevice2HelloWorld": {
      "topic": "clients/MyClientDevice2/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    }
  }
}
```

2.0.x

mqttTopicMapping

Les mappages de sujets que vous souhaitez relier. Ce composant s'abonne aux messages du sujet source et publie les messages qu'il reçoit dans le sujet de destination. Chaque mappage de rubrique définit le sujet, le type de source et le type de destination.

Cet objet contient les informations suivantes :

topicMappingNameKey

Nom de ce mappage de rubriques. *topicMappingNameKey* Remplacez-le par un nom qui vous aide à identifier ce mappage de rubriques.

Cet objet contient les informations suivantes :

topic

Le sujet ou le filtre de sujet permettant de faire le lien entre les courtiers source et cible.

Si vous spécifiez le courtier LocalMqtt ou le courtier IotCore source, vous pouvez utiliser les caractères génériques de sujet + et # MQTT pour relayer des messages sur tous les sujets correspondant à un filtre de sujet. Pour plus d'informations, consultez les [rubriques relatives au MQTT](#) dans le Guide du AWS IoT Core développeur.

source

Le courtier de messages source. Sélectionnez parmi les options suivantes :

- LocalMqtt— Le broker MQTT local où les appareils clients communiquent.
- Pubsub— Le courtier de publish/subscribe messages Greengrass local.
- IotCore— Le courtier de messages AWS IoT Core MQTT.

Note

Le pont MQTT utilise QoS 1 pour publier et s' AWS IoT Core abonner, même lorsqu'un appareil client utilise QoS 0 pour publier et s'abonner au courtier MQTT local. Par conséquent, vous pouvez observer une latence supplémentaire lorsque vous relayer des messages MQTT depuis des appareils clients sur le broker MQTT local vers. AWS IoT Core Pour plus d'informations sur la configuration MQTT sur les appareils principaux, consultez [Configurer les délais d'expiration et les paramètres de cache du MQTT](#).

sourceet target doit être différent.

target

Le courtier de messages cible. Sélectionnez parmi les options suivantes :

- LocalMqtt— Le broker MQTT local où les appareils clients communiquent.
- Pubsub— Le courtier de publish/subscribe messages Greengrass local.
- IotCore— Le courtier de messages AWS IoT Core MQTT.

 Note

Le pont MQTT utilise QoS 1 pour publier et s' AWS IoT Core abonner, même lorsqu'un appareil client utilise QoS 0 pour publier et s'abonner au courtier MQTT local. Par conséquent, vous pouvez observer une latence supplémentaire lorsque vous relayez des messages MQTT depuis des appareils clients sur le broker MQTT local vers. AWS IoT Core Pour plus d'informations sur la configuration MQTT sur les appareils principaux, consultez [Configurer les délais d'expiration et les paramètres de cache du MQTT](#).

source et target doit être différent.

Exemple Exemple : mise à jour de la fusion de configurations

L'exemple de mise à jour de configuration suivant indique de relayer les messages des appareils clients vers AWS IoT Core les clients/MyClientDevice2/hello/world rubriques clients/MyClientDevice1/hello/world et.

```
{
  "mqttTopicMapping": {
    "ClientDevice1HelloWorld": {
      "topic": "clients/MyClientDevice1/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    },
    "ClientDevice2HelloWorld": {
      "topic": "clients/MyClientDevice2/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    }
  }
}
```

Fichier journal local

Ce composant utilise le même fichier journal que le composant [Greengrass nucleus](#).

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

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

Pour consulter les journaux de ce composant

- Exécutez la commande suivante sur le périphérique principal pour afficher le fichier journal de ce composant en temps réel. Remplacez `/greengrass/v2` ou `C:\greengrass\v2` par le chemin d'accès au dossier AWS IoT Greengrass racine.

Linux

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

Windows (PowerShell)

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

Journal des modifications

Le tableau suivant décrit les modifications apportées à chaque version du composant.

Version	Modifications
2.3.2	Version mise à jour pour la version 2.5.0 d' authentification des appareils clients .
2.3.1	Corrections de bugs et améliorations Résout un problème où le client MQTT local entre dans une boucle de déconnexion.

Version	Modifications
2.3.0	Nouvelles fonctionnalités Ajoute la MQTT5 prise en charge du pont entre les sources MQTT locales AWS IoT Core et entre celles-ci.
2.2.6	Nouvelles fonctionnalités Ajoute une nouvelle option <code>startupTimeoutSeconds</code> de configuration.
2.2.5	Version mise à jour pour la version 2.4.0 d' authentification du périphérique client .
2.2.4	Version mise à jour pour la version 2.3.0 d' authentification des appareils clients Greengrass.
2.2.3	Cette version contient des corrections de bogues et des améliorations.
2.2.2	Corrections de bugs et améliorations Ajustements de journalisation.
2.2.1	Corrections de bugs et améliorations Résout les problèmes qui peuvent empêcher le pont MQTT de s'abonner aux rubriques MQTT.
2.2.0	Nouvelles fonctionnalités - Permet de prendre en charge les caractères génériques des rubriques MQTT (<code>#et+</code>) lorsque vous spécifiez <code>local publish/subscribe</code> comme courtier de messages source. Cette fonctionnalité nécessite la version 2.6.0 ou ultérieure du composant Greengrass nucleus. - Ajoute l'<code>targetTopicPrefix</code> option, que vous pouvez spécifier pour configurer le pont MQTT afin d'ajouter un préfixe au sujet cible lorsqu'il relaie un message.

Version	Modifications
2.1.1	Corrections de bugs et améliorations • Résout les problèmes liés à la façon dont ce composant gère les mises à jour de réinitialisation de configuration. • Réduit la fréquence des déconnexions du client MQTT lors de la rotation des certificats.
2.1.0	Nouvelles fonctionnalités • Ajoute le <code>brokerUri</code> paramètre, qui vous permet d'utiliser un port de broker MQTT autre que celui par défaut.
2.0.1	Cette version inclut des corrections de bogues et des améliorations.
2.0.0	Première version.

Courtier MQTT 3.1.1 (Moquette)

Le composant broker Moquette MQTT (`aws.greengrass.clientdevices.mqtt.Moquette`) gère les messages MQTT entre les appareils clients et un périphérique principal de Greengrass. Ce composant fournit une version modifiée du broker [Moquette MQTT](#). Déployez ce broker MQTT pour exécuter un broker MQTT léger. Pour plus d'informations sur le choix d'un courtier MQTT, consultez [Choisissez un courtier MQTT](#).

Ce broker implémente le protocole MQTT 3.1.1. Il inclut la prise en charge des messages conservés QoS 0, QoS 1, QoS 2, des messages de dernier testament et des sessions persistantes.

Note

Les appareils clients sont des appareils IoT locaux qui se connectent à un appareil principal de Greengrass pour envoyer des messages MQTT et des données à traiter. Pour de plus amples informations, veuillez consulter [Interagissez avec les appareils IoT locaux](#).

Rubriques

- [Versions](#)
- [Type](#)

- [Système d'exploitation](#)
- [Prérequis](#)
- [Dépendances](#)
- [Configuration](#)
- [Fichier journal local](#)
- [Journal des modifications](#)

Versions

Les versions de ce composant sont les suivantes :

- 2.3.x
- 2.2.x
- 2,1x
- 2,0.x

Type

Ce composant est un composant de plugin (`aws.greengrass.plugin`). Le [noyau Greengrass](#) exécute ce composant dans la même machine virtuelle Java (JVM) que le noyau. Le noyau redémarre lorsque vous modifiez la version de ce composant sur le périphérique principal.

Ce composant utilise le même fichier journal que le noyau Greengrass. Pour de plus amples informations, veuillez consulter [AWS IoT Greengrass Journaux de surveillance](#).

Pour de plus amples informations, veuillez consulter [Types de composants](#).

Système d'exploitation

Ce composant peut être installé sur les appareils principaux qui exécutent les systèmes d'exploitation suivants :

- Linux
- Windows

Prérequis

Ce composant répond aux exigences suivantes :

- Le périphérique principal doit être en mesure d'accepter des connexions sur le port où fonctionne le broker MQTT. Ce composant exécute le broker MQTT sur le port 8883 par défaut. Vous pouvez spécifier un port différent lorsque vous configurez ce composant.

Si vous spécifiez un port différent et que vous utilisez le [composant pont MQTT](#) pour relayer les messages MQTT à d'autres courtiers, vous devez utiliser le pont MQTT v2.1.0 ou version ultérieure. Configurez-le pour utiliser le port sur lequel le broker MQTT fonctionne.

Si vous spécifiez un port différent et que vous utilisez le [composant de détection IP](#) pour gérer les points de terminaison du broker MQTT, vous devez utiliser le détecteur IP v2.1.0 ou version ultérieure. Configurez-le pour signaler le port sur lequel le broker MQTT fonctionne.

- Le composant broker Moquette MQTT est compatible pour fonctionner dans un VPC.

Dépendances

Lorsque vous déployez un composant, il déploie AWS IoT Greengrass également des versions compatibles de ses dépendances. Cela signifie que vous devez satisfaire aux exigences relatives au composant et à toutes ses dépendances pour réussir le déploiement du composant. Cette section répertorie les dépendances des [versions publiées](#) de ce composant et les contraintes de version sémantiques qui définissent les versions des composants pour chaque dépendance. Vous pouvez également consulter les dépendances de chaque version du composant dans la [AWS IoT Greengrass console](#). Sur la page de détails du composant, recherchez la liste des dépendances.

2.3.7

Le tableau suivant répertorie les dépendances pour la version 2.3.7 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Authentification de l'appareil client	>=2,2,0 <2,6,0	Stricte

2.3.2 – 2.3.6

Le tableau suivant répertorie les dépendances pour les versions 2.3.2 à 2.3.6 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Authentification de l'appareil client	$\geq 2,2,0$ $< 2,5,0$	Stricte

2.3.0 and 2.3.1

Le tableau suivant répertorie les dépendances pour les versions 2.3.0 et 2.3.1 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Authentification de l'appareil client	$\geq 2,2,0$ $< 2,4,0$	Stricte

2.2.0

Le tableau suivant répertorie les dépendances pour la version 2.2.0 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Authentification de l'appareil client	$\geq 2,2,0$ $< 2,3,0$	Stricte

2.1.0

Le tableau suivant répertorie les dépendances pour la version 2.1.0 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Authentification de l'appareil client	$\geq 2,0,0$ $< 2,2,0$	Stricte

2.0.0 - 2.0.2

Le tableau suivant répertorie les dépendances pour les versions 2.0.0 à 2.0.2 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Authentification de l'appareil client	>=2,0.0 <2,10	Stricte

Pour plus d'informations sur les dépendances des composants, consultez la [référence de la recette des composants](#).

Configuration

Ce composant fournit les paramètres de configuration suivants que vous pouvez personnaliser lorsque vous déployez le composant.

`moquette`

(Facultatif) La configuration du [broker Moquette MQTT](#) à utiliser. Vous pouvez configurer un sous-ensemble d'options de configuration de Moquette dans ce composant. Pour plus d'informations, consultez les commentaires intégrés dans le fichier de [configuration de Moquette](#).

Cet objet contient les informations suivantes :

`ssl_port`

(Facultatif) Le port sur lequel le broker MQTT fonctionne.

Note

Si vous spécifiez un port différent et que vous utilisez le [composant pont MQTT](#) pour relayer les messages MQTT à d'autres courtiers, vous devez utiliser le pont MQTT v2.1.0 ou version ultérieure. Configurez-le pour utiliser le port sur lequel le broker MQTT fonctionne.

Si vous spécifiez un port différent et que vous utilisez le [composant de détection IP](#) pour gérer les points de terminaison du broker MQTT, vous devez utiliser le détecteur IP v2.1.0 ou version ultérieure. Configurez-le pour signaler le port sur lequel le broker MQTT fonctionne.

Par défaut : 8883

host

(Facultatif) Interface à laquelle le broker MQTT se lie. Par exemple, vous pouvez modifier ce paramètre afin que le broker MQTT se lie uniquement à un réseau local spécifique.

Par défaut : 0.0.0.0 (se lie à toutes les interfaces réseau)

startupTimeoutSeconds

(Facultatif) Durée maximale en secondes pendant laquelle le composant démarre. L'état du composant passe à BROKEN s'il dépasse ce délai d'attente.

Par défaut : 120

Exemple Exemple : mise à jour de la fusion de configurations

L'exemple de configuration suivant indique de faire fonctionner le broker MQTT sur le port 443.

```
{
  "moquette": {
    "ssl_port": "443"
  }
}
```

Fichier journal local

Ce composant utilise le même fichier journal que le composant [Greengrass nucleus](#).

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

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

Pour consulter les journaux de ce composant

- Exécutez la commande suivante sur le périphérique principal pour afficher le fichier journal de ce composant en temps réel. Remplacez `/greengrass/v2` ou `C:\greengrass\v2` par le chemin d'accès au dossier AWS IoT Greengrass racine.

Linux

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

Windows (PowerShell)

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

Journal des modifications

Le tableau suivant décrit les modifications apportées à chaque version du composant.

Version	Modifications
2.3.7	Version mise à jour pour la version 2.5.0 d' authentification des appareils clients .
2.3.6	Corrections de bogues et améliorations Correction et amélioration de bogues généraux
2.3.5	Corrections de bogues et améliorations Mise à jour de Moquette vers la version 0.17.
2.3.4	Corrections de bogues et améliorations Résout un problème à cause duquel les clients pouvaient rencontrer des erreurs de session non valides lors de l'envoi ou de la réception de messages, en raison d'un client dupliqué IDs. Ce problème a entraîné la fermeture de la session du client.

Version	Modifications
2.3.3	Nouvelles fonctionnalités Ajoute une nouvelle option <code>startupTimeoutSeconds</code> de configuration.
2.3.2	Version mise à jour pour la version 2.4.0 d' authentification du périphérique client .
2.3.1	Corrections de bogues et améliorations Corrige un problème de course dans lequel les clients peuvent être déconnectés après avoir tenté de se reconnecter, en raison d'une session non valide.
2.3.0	Ajoute la prise en charge des chaînes de certificats.
2.2.0	Version mise à jour pour la version 2.2.0 d' authentification du périphérique client .
2.1.0	Corrections de bogues et améliorations - Met à jour ce composant pour utiliser la version 0.16 de Moquette, qui améliore les performances et inclut plusieurs autres améliorations. - Résout un problème selon lequel le certificat du serveur MQTT local change plus souvent que prévu dans certains scénarios. Pour appliquer ce correctif, vous devez également utiliser la version 2.1.0 ou ultérieure du composant d'authentification du périphérique client.
2.0.2	Corrections de bogues et améliorations - Augmente la taille maximale des messages MQTT de 8 092 octets à 128 kilo-octets. La limite de charge utile effective des messages MQTT est légèrement inférieure, car la limite de taille des messages inclut les en-têtes de message. - Ajoute la prise en charge des valeurs entières dans le <code>ssl_port</code> paramètre.

Version	Modifications
2.0.1	Version mise à jour pour la version 2.4.0 de Greengrass Nucleus.
2.0.0	Première version.

Courtier MQTT 5 (EMQX)

Le composant broker EMQX MQTT (`aws.greengrass.clientdevices.mqtt.EMQX`) gère les messages MQTT entre les appareils clients et un périphérique principal de Greengrass. Ce composant fournit une version modifiée du broker [EMQX MQTT 5.0](#). Déployez ce broker MQTT pour utiliser les fonctionnalités de MQTT 5 dans la communication entre les appareils clients et un périphérique principal. Pour plus d'informations sur le choix d'un courtier MQTT, consultez [Choisissez un courtier MQTT](#).

Ce broker implémente le protocole MQTT 5.0. Il inclut la prise en charge des intervalles d'expiration des sessions et des messages, des propriétés utilisateur, des abonnements partagés, des alias de rubrique, etc. MQTT 5 est rétrocompatible avec MQTT 3.1.1, donc si vous exécutez le broker [Moquette MQTT 3.1.1](#), vous pouvez le remplacer par le broker EMQX MQTT 5, et les appareils clients peuvent continuer à se connecter et à fonctionner comme d'habitude.

Note

Les appareils clients sont des appareils IoT locaux qui se connectent à un appareil principal de Greengrass pour envoyer des messages MQTT et des données à traiter. Pour de plus amples informations, veuillez consulter [Interagissez avec les appareils IoT locaux](#).

Rubriques

- [Versions](#)
- [Type](#)
- [Système d'exploitation](#)
- [Prérequis](#)
- [Dépendances](#)
- [Configuration](#)

- [Fichier journal local](#)
- [Licences](#)
- [Journal des modifications](#)

Versions

Les versions de ce composant sont les suivantes :

- 2,0.x
- 1,2.x
- 1,1x
- 1,0 x

Type

Ce composant est un composant générique (`aws.greengrass.generic`). Le [noyau Greengrass](#) exécute les scripts de cycle de vie du composant.

Pour de plus amples informations, veuillez consulter [Types de composants](#).

Système d'exploitation

Ce composant peut être installé sur les appareils principaux qui exécutent les systèmes d'exploitation suivants :

- Linux
- Windows

Prérequis

Ce composant répond aux exigences suivantes :

- Le périphérique principal doit être capable d'accepter des connexions sur le port où fonctionne le broker MQTT. Ce composant exécute le broker MQTT sur le port 8883 par défaut. Vous pouvez spécifier un port différent lorsque vous configurez ce composant.

Si vous spécifiez un port différent et que vous utilisez le [composant pont MQTT](#) pour relayer les messages MQTT à d'autres courtiers, vous devez utiliser le pont MQTT v2.1.0 ou version ultérieure. Configurez-le pour utiliser le port sur lequel le broker MQTT fonctionne.

Si vous spécifiez un port différent et que vous utilisez le [composant de détection IP](#) pour gérer les points de terminaison du broker MQTT, vous devez utiliser le détecteur IP v2.1.0 ou version ultérieure. Configurez-le pour signaler le port sur lequel le broker MQTT fonctionne.

- Sur les appareils principaux Linux, Docker est installé et configuré sur le périphérique principal :
 - [Docker Engine](#) 1.9.1 ou version ultérieure installé sur le périphérique principal de Greengrass. La version 20.10 est la dernière version vérifiée pour fonctionner avec le logiciel AWS IoT Greengrass Core. Vous devez installer Docker directement sur le périphérique principal avant de déployer des composants qui exécutent des conteneurs Docker.
 - Le daemon Docker a démarré et s'est exécuté sur le périphérique principal avant que vous ne déployiez ce composant.
 - L'utilisateur du système qui exécute ce composant doit disposer des autorisations root ou administrateur. Vous pouvez également exécuter ce composant en tant qu'utilisateur du système dans le `docker` groupe et configurer l'`requiresPrivileges` option de ce composant `false` pour exécuter le courtier EMQX MQTT sans privilèges.
- Le composant broker EMQX MQTT est compatible pour s'exécuter dans un VPC.
- Le composant broker EMQX MQTT n'est pas pris en charge sur la plateforme. `armv7`

Dépendances

Lorsque vous déployez un composant, il déploie AWS IoT Greengrass également des versions compatibles de ses dépendances. Cela signifie que vous devez satisfaire aux exigences relatives au composant et à toutes ses dépendances pour réussir le déploiement du composant. Cette section répertorie les dépendances des [versions publiées](#) de ce composant et les contraintes de version sémantiques qui définissent les versions des composants pour chaque dépendance. Vous pouvez également consulter les dépendances de chaque version du composant dans la [AWS IoT Greengrass console](#). Sur la page de détails du composant, recherchez la liste des dépendances.

2.0.2 – 2.0.3

Le tableau suivant répertorie les dépendances pour les versions 2.0.2 et 2.0.3 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Authentification de l'appareil client	$\geq 2,2,0$ $< 2,6,0$	Flexible

2.0.1

Le tableau suivant répertorie les dépendances pour la version 2.0.1 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Authentification de l'appareil client	$\geq 2,2,0$ $< 2,6,0$	Stricte

2.0.0

Le tableau suivant répertorie les dépendances pour la version 2.0.0 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Authentification de l'appareil client	$\geq 2,2,0$ $< 2,5,0$	Stricte

1.2.2 – 1.2.3

Le tableau suivant répertorie les dépendances pour les versions 1.2.2 à 1.2.3 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Authentification de l'appareil client	$\geq 2,2,0$ $< 2,5,0$	Stricte

1.2.0 and 1.2.1

Le tableau suivant répertorie les dépendances pour les versions 1.2.0 et 1.2.1 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Authentification de l'appareil client	>=2,2,0 <2,4,0	Stricte

1.0.0 and 1.1.0

Le tableau suivant répertorie les dépendances pour les versions 1.0.0 et 1.1.0 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Authentification de l'appareil client	>=2,2,0 <2,3,0	Stricte

Pour plus d'informations sur les dépendances des composants, consultez la [référence de la recette des composants](#).

Configuration

2.0.0 - 2.0.3

Ce composant fournit les paramètres de configuration suivants que vous pouvez personnaliser lorsque vous déployez le composant.

Important

Si vous utilisez la version 2 du composant MQTT 5 broker (EMQX), vous devez mettre à jour votre fichier de configuration. Les fichiers de configuration de la version 1 ne fonctionnent pas avec la version 2.

Configuration EMQX

(Facultatif) La configuration du [broker EMQX MQTT](#) à utiliser. Vous pouvez définir les options de configuration EMQX dans ce composant.

Lorsque vous utilisez le broker EMQX, Greengrass utilise une configuration par défaut. Cette configuration est utilisée sauf si vous la modifiez à l'aide de ce champ.

La modification des paramètres de configuration suivants entraîne le redémarrage du composant du broker EMQX. Les autres modifications de configuration s'appliquent sans redémarrer le composant.

- `emqxConfig/cluster`
- `emqxConfig/node`
- `emqxConfig/rpc`

 Note

`aws.greengrass.clientdevices.mqtt.EMQX` vous permet de configurer des options sensibles à la sécurité. Il s'agit notamment des paramètres TLS, de l'authentification et des fournisseurs d'autorisation. Nous avons recommandé la configuration par défaut qui utilise l'authentification TLS mutuelle et le fournisseur d'authentification des appareils clients Greengrass.

Exemple Exemple : configuration par défaut

L'exemple suivant montre les valeurs par défaut définies pour le broker MQTT 5 (EMQX). Vous pouvez remplacer ces paramètres à l'aide du paramètre `emqxConfig` de configuration.

```
{
  "authorization": {
    "no_match": "deny",
    "sources": []
  },
  "node": {
    "cookie": "<placeholder>"
  },
  "listeners": {
    "ssl": {
      "default": {
        "ssl_options": {
          "keyfile": "{work:path}\\data\\key.pem",
          "certfile": "{work:path}\\data\\cert.pem",
          "cacertfile": null,
          "verify": "verify_peer",
          "versions": ["tlsv1.3", "tlsv1.2"],
          "fail_if_no_peer_cert": true
        }
      }
    }
  }
}
```

```

    }
  },
  "tcp": {
    "default": {
      "enabled": false
    }
  },
  "ws": {
    "default": {
      "enabled": false
    }
  },
  "wss": {
    "default": {
      "enabled": false
    }
  }
},
"plugins": {
  "states": [{"name_vsn": "gg-1.0.0", "enable": true}],
  "install_dir": "plugins"
}
}

```

Mode Auth

(Facultatif) Définit le fournisseur d'autorisation pour le courtier. Il peut avoir l'une des valeurs suivantes :

- **enabled**— (Par défaut) Utilisez le fournisseur d'authentification et d'autorisation Greengrass.
- **bypass_on_failure**— Utilisez le fournisseur d'authentification Greengrass, puis utilisez tous les fournisseurs d'authentification restants de la chaîne de fournisseurs EMQX si Greengrass refuse l'authentification ou l'autorisation.
- **bypass**— Le fournisseur Greengrass est désactivé. L'authentification et l'autorisation sont gérées par la chaîne de fournisseurs EMQX.

requiresPrivilege

(Facultatif) Sur les appareils principaux Linux, vous pouvez spécifier d'exécuter le broker EMQX MQTT sans droits root ou administrateur. Si vous définissez cette option sur `false`, l'utilisateur du système qui exécute ce composant doit être membre du `docker` groupe.

Par défaut : `true`

`startupTimeoutSeconds`

(Facultatif) Durée maximale en secondes pendant laquelle le broker EMQX MQTT démarre. L'état du composant passe à `BROKEN` s'il dépasse ce délai.

Par défaut : `90`

`ipcTimeoutSeconds`

(Facultatif) Durée maximale en secondes pendant laquelle le composant attend que le noyau Greengrass réponde aux demandes de communication interprocessus (IPC). Augmentez ce nombre si ce composant signale des erreurs de temporisation lorsqu'il vérifie si un appareil client est autorisé.

Par défaut : `5`

`crtLogLevel`

(Facultatif) Le niveau de journalisation de la bibliothèque AWS Common Runtime (CRT).

La valeur par défaut est le niveau de journalisation du broker EMQX MQTT (`in.log.level` `emqx`).

`restartIdentifier`

(Facultatif) Configurez cette option pour redémarrer le broker EMQX MQTT. Lorsque cette valeur de configuration change, ce composant redémarre le broker MQTT. Vous pouvez utiliser cette option pour forcer les appareils clients à se déconnecter.

`dockerOptions`

(Facultatif) Configurez cette option uniquement sur les systèmes d'exploitation Linux pour ajouter des paramètres à la ligne de commande Docker. Par exemple, pour mapper des ports supplémentaires, utilisez le paramètre `-p` Docker :

```
"-p 1883:1883"
```

Exemple Exemple : mise à jour d'un fichier de configuration v1.x vers la version 2.x

L'exemple suivant montre les modifications nécessaires pour mettre à jour un fichier de configuration v1.x vers la version 2.x.

Le fichier de configuration de la version 1.x :

```
{
  "emqx": {
    "listener.ssl.external": "443",
    "listener.ssl.external.max_connections": "1024000",
    "listener.ssl.external.max_conn_rate": "500",
    "listener.ssl.external.rate_limit": "50KB,5s",
    "listener.ssl.external.handshake_timeout": "15s",
    "log.level": "warning"
  },
  "mergeConfigurationFiles": {
    "etc/plugins/aws_greengrass_emqx_auth.conf": "auth_mode=enabled\nuse_greengrass_managed_certificates=true\n"
  }
}
```

Le fichier de configuration équivalent pour la v2 :

```
{
  "emqxConfig": {
    "listeners": {
      "ssl": {
        "default": {
          "bind": "8883",
          "max_connections": "1024000",
          "max_conn_rate": "500",
          "ssl_options": {
            "handshake_timeout": "15s"
          }
        }
      }
    },
    "log": {
      "console": {
        "enable": true,
        "level": "warning"
      }
    }
  },
  "authMode": "enabled"
}
```

Il n'existe aucun équivalent à l'entrée `listener.ssl.external.rate_limit` de configuration. L'option `use_greengrass_managed_certificates` de configuration a été supprimée.

Exemple Exemple : définir un nouveau port pour le courtier

Dans l'exemple suivant, le port sur lequel le broker MQTT opère passe du port par défaut 8883 au port 1234. Si vous utilisez Linux, incluez le `dockerOptions` champ.

```
{
  "emqxConfig": {
    "listeners": {
      "ssl": {
        "default": {
          "bind": 1234
        }
      }
    }
  },
  "dockerOptions": "-p 1234:1234"
}
```

Exemple Exemple : ajuster le niveau de journalisation du broker MQTT

L'exemple suivant change le niveau de journalisation du broker MQTT en `debug`. Vous pouvez choisir l'un des niveaux de journalisation suivants :

- `debug`
- `info`
- `notice`
- `warning`
- `error`
- `critical`
- `alert`
- `emergency`

Le niveau de journalisation par défaut est `warning`.

```
{
```

```
"emqxConfig": {
  "log": {
    "console": {
      "level": "debug"
    }
  }
}
```

Exemple Exemple : activer le tableau de bord EMQX

L'exemple suivant active le tableau de bord EMQX afin que vous puissiez surveiller et gérer votre courtier. Si vous utilisez Linux, incluez le `dockerOptions` champ.

```
{
  "emqxConfig": {
    "dashboard": {
      "listeners": {
        "http": {
          "bind": 18083
        }
      }
    }
  },
  "dockerOptions": "-p 18083:18083"
}
```

1.0.0 - 1.2.2

Ce composant fournit les paramètres de configuration suivants que vous pouvez personnaliser lorsque vous déployez le composant.

emqx

(Facultatif) La configuration du [broker EMQX MQTT](#) à utiliser. Vous pouvez configurer un sous-ensemble d'options de configuration EMQX dans ce composant.

Cet objet contient les informations suivantes :

`listener.ssl.external`

(Facultatif) Le port sur lequel le broker MQTT fonctionne.

 Note

Si vous spécifiez un port différent et que vous utilisez le [composant pont MQTT](#) pour relayer les messages MQTT à d'autres courtiers, vous devez utiliser le pont MQTT v2.1.0 ou version ultérieure. Configurez-le pour utiliser le port sur lequel le broker MQTT fonctionne.

Si vous spécifiez un port différent et que vous utilisez le [composant de détection IP](#) pour gérer les points de terminaison du broker MQTT, vous devez utiliser le détecteur IP v2.1.0 ou version ultérieure. Configurez-le pour signaler le port sur lequel le broker MQTT fonctionne.

Par défaut : 8883

`listener.ssl.external.max_connections`

(Facultatif) Le nombre maximum de connexions simultanées prises en charge par le broker MQTT.

Par défaut : 1024000

`listener.ssl.external.max_conn_rate`

(Facultatif) Le nombre maximum de nouvelles connexions par seconde que le broker MQTT peut recevoir.

Par défaut : 500

`listener.ssl.external.rate_limit`

(Facultatif) Limite de bande passante pour toutes les connexions au broker MQTT. Spécifiez la bande passante et la durée de cette bande passante séparées par une virgule (,) au format suivant :bandwidth,duration. Par exemple, vous pouvez spécifier de 50KB,5s limiter le broker MQTT à 50 kilo-octets (Ko) de données toutes les 5 secondes.

`listener.ssl.external.handshake_timeout`

(Facultatif) Durée pendant laquelle le broker MQTT attend pour terminer l'authentification d'une nouvelle connexion.

Par défaut : 15s

`mqtt.max_packet_size`

(Facultatif) Taille maximale d'un message MQTT.

Par défaut : 268435455 (256 Mo moins 1)

`log.level`

(Facultatif) Le niveau de journalisation du broker MQTT. Sélectionnez parmi les options suivantes :

- `debug`
- `info`
- `notice`
- `warning`
- `error`
- `critical`
- `alert`
- `emergency`

Le niveau de journalisation par défaut est `warning`.

`requiresPrivilege`

(Facultatif) Sur les appareils principaux Linux, vous pouvez spécifier d'exécuter le broker EMQX MQTT sans droits root ou administrateur. Si vous définissez cette option sur `false`, l'utilisateur du système qui exécute ce composant doit être membre du `docker` groupe.

Par défaut : `true`

`startupTimeoutSeconds`

(Facultatif) Durée maximale en secondes pendant laquelle le broker EMQX MQTT démarre. L'état du composant passe à `BROKEN` s'il dépasse ce délai.

Par défaut : 90

`ipcTimeoutSeconds`

(Facultatif) Durée maximale en secondes pendant laquelle le composant attend que le noyau Greengrass réponde aux demandes de communication interprocessus (IPC). Augmentez ce

nombre si ce composant signale des erreurs de temporisation lorsqu'il vérifie si un appareil client est autorisé.

Par défaut : 5

`crtLogLevel`

(Facultatif) Le niveau de journalisation de la bibliothèque AWS Common Runtime (CRT).

La valeur par défaut est le niveau de journalisation du broker EMQX MQTT (in). `log.level`
`emqx`

`restartIdentifier`

(Facultatif) Configurez cette option pour redémarrer le broker EMQX MQTT. Lorsque cette valeur de configuration change, ce composant redémarre le broker MQTT. Vous pouvez utiliser cette option pour forcer les appareils clients à se déconnecter.

`dockerOptions`

(Facultatif) Configurez cette option uniquement sur les systèmes d'exploitation Linux pour ajouter des paramètres à la ligne de commande Docker. Par exemple, pour mapper des ports supplémentaires, utilisez le paramètre `-p` Docker :

```
"-p 1883:1883"
```

`mergeConfigurationFiles`

(Facultatif) Configurez cette option pour ajouter ou remplacer les valeurs par défaut dans les fichiers de configuration EMQX spécifiés. Pour plus d'informations sur les fichiers de configuration et leurs formats, consultez la section [Configuration](#) dans la documentation d'EMQX 4.0. Les valeurs que vous spécifiez sont ajoutées au fichier de configuration.

L'exemple suivant met à jour le `etc/emqx.conf` fichier.

```
"mergeConfigurationFiles": {  
  "etc/emqx.conf": "broker.sys_interval=30s\nbroker.sys_heartbeat=10s"  
},
```

Outre les fichiers de configuration pris en charge par EMQX, Greengrass prend en charge un fichier qui configure le plugin d'authentification Greengrass pour EMQX appelé. `etc/`

`plugins/aws_greengrass_emqx_auth.conf` Il existe deux options prises en charge, `auth_mode` et `use_greengrass_managed_certificates`. Pour utiliser un autre fournisseur d'authentification, définissez l'`auth_mode` option sur l'une des options suivantes :

- `enabled`— (Par défaut) Utilisez le fournisseur d'authentification et d'autorisation Greengrass.
- `bypass_on_failure`— Utilisez le fournisseur d'authentification Greengrass, puis utilisez tous les fournisseurs d'authentification restants de la chaîne de fournisseurs EMQX si Greengrass refuse l'authentification ou l'autorisation.
- `bypass`— Le fournisseur Greengrass est désactivé. L'authentification et l'autorisation sont ensuite gérées par la chaîne de fournisseurs EMQX.

Dans l'`use_greengrass_managed_certificates` affirmative `true`, cette option indique que Greengrass gère les certificats TLS du courtier. Si `false`, cela indique que vous fournissez les certificats via une autre source.

L'exemple suivant met à jour les valeurs par défaut du fichier `etc/plugins/aws_greengrass_emqx_auth.conf` de configuration.

```
"mergeConfigurationFiles": {  
  "etc/plugins/aws_greengrass_emqx_auth.conf": "auth_mode=enabled\n  
  use_greengrass_managed_certificates=true\n"  
},
```

Note

`aws.greengrass.clientdevices.mqtt.EMQX` vous permet de configurer des options sensibles à la sécurité. Il s'agit notamment des paramètres TLS, de l'authentification et des fournisseurs d'autorisation. La configuration recommandée est la configuration par défaut qui utilise l'authentification TLS mutuelle et le fournisseur Greengrass Client Device Auth.

`replaceConfigurationFiles`

(Facultatif) Configurez cette option pour remplacer les fichiers de configuration EMQX spécifiés. Les valeurs que vous spécifiez remplacent l'intégralité du fichier de configuration existant. Vous ne pouvez pas spécifier le `etc/emqx.conf` fichier dans cette section. Vous devez utiliser `mergeConfigurationFile` pour modifier `etc/emqx.conf`.

Exemple Exemple : mise à jour de la fusion de configurations

L'exemple de configuration suivant indique de faire fonctionner le broker MQTT sur le port 443.

```
{
  "emqx": {
    "listener.ssl.external": "443",
    "listener.ssl.external.max_connections": "1024000",
    "listener.ssl.external.max_conn_rate": "500",
    "listener.ssl.external.rate_limit": "50KB,5s",
    "listener.ssl.external.handshake_timeout": "15s",
    "log.level": "warning"
  },
  "requiresPrivilege": "true",
  "startupTimeoutSeconds": "90",
  "ipcTimeoutSeconds": "5"
}
```

Fichier journal local

Ce composant utilise le fichier journal suivant.

Linux

```
/greengrass/v2/logs/aws.greengrass.clientdevices.mqtt.EMQX.log
```

Windows

```
C:\greengrass\v2\logs\aws.greengrass.clientdevices.mqtt.EMQX.log
```

Pour consulter les journaux de ce composant

- Exécutez la commande suivante sur le périphérique principal pour afficher le fichier journal de ce composant en temps réel. Remplacez */greengrass/v2* ou *C:\greengrass\v2* par le chemin d'accès au dossier AWS IoT Greengrass racine.

Linux

```
sudo tail -f /greengrass/v2/logs/aws.greengrass.clientdevices.mqtt.EMQX.log
```

Windows (PowerShell)

```
Get-Content C:\greengrass\v2\logs\aws.greengrass.clientdevices.mqtt.EMQX.log -  
Tail 10 -Wait
```

Licences

Sur les systèmes d'exploitation Windows, ce logiciel inclut du code distribué conformément aux [termes du contrat de licence logiciel Microsoft - Microsoft Visual Studio Community 2022](#). En téléchargeant ce logiciel, vous acceptez les termes du contrat de licence de ce code.

Ce composant 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 composant.

v2.x

Version	Modifications
2.0.3	Corrections de bogues et améliorations Résout le problème selon lequel EMQX ne démarre pas sous Windows si le chemin contient des espaces.
2.0.2	Corrections de bogues et améliorations Résout un problème selon lequel EMQX démarre avant que le composant d'authentification du périphérique client ne soit prêt.
2.0.1	Version mise à jour pour la version 2.5.0 d' authentification des appareils clients .
2.0.0	Cette version du broker MQTT 5 (EMQX) attend des paramètres de configuration différents de ceux de la version 1.x. Si vous utilisez une configuration autre que celle par défaut pour la version 1.x, vous devez mettre à jour la configuration du composant pour 2.x. Pour de plus amples informations, veuillez consulter Configuration .

Version	Modifications
	Nouvelles fonctionnalités • Met à niveau le broker MQTT vers EMQX 5.1.1. • Permet de modifier la configuration du broker sans redémarrer le composant. Mises à jour • Ajoute un nouveau champ de <code>emqxConfig</code> configuration qui remplace les champs <code>emqxmergeConfigurationFiles</code> , et <code>replaceConfigurationFiles</code> de configuration.

v1.x

Version	Modifications
1.2.3	Corrections de bogues et améliorations • Résout un problème empêchant les clients d'interagir avec EMQX après s'être authentifiés en déconnectant puis en réauthentifiant le client.
1.2.2	Version mise à jour pour la version 2.4.0 d' authentification du périphérique client .
1.2.1	Corrections de bogues et améliorations • Résout un problème selon lequel le composant ne démarrait pas sous Windows si Visual C++ Redistributable n'est pas déjà présent. • Met à jour EMQX vers la version 4.4.14.
1.2.0	Ajoute la prise en charge des chaînes de certificats.
1.1.0	Nouvelles fonctionnalités • Prend en charge les configurations EMQX, y compris les options de broker et les plug-ins. Corrections de bogues et améliorations • Met à jour EMQX vers la version 4.4.9.

Version	Modifications
1.0.1	Résout un problème lors de la prise de contact TLS qui empêchait certains clients MQTT de se connecter.
1.0.0	Première version.

Émetteur de télémétrie Nucleus

Le composant émetteur de télémétrie Nucleus (`aws.greengrass.telemetry.NucleusEmitter`) collecte les données de télémétrie relatives à l'état du système et les publie en permanence sur un sujet local et un sujet MQTT. AWS IoT Core Ce composant vous permet de recueillir les données télémétriques du système en temps réel sur vos principaux appareils Greengrass. Pour plus d'informations sur l'agent de télémétrie Greengrass qui publie les données de télémétrie du système sur Amazon, consultez. EventBridge [Collectez les données de télémétrie relatives à l'état du système à partir des principaux appareils AWS IoT Greengrass](#)

Par défaut, le composant émetteur de télémétrie Nucleus publie les données de télémétrie toutes les 60 secondes sur le sujet local suivant. `publish/subscribe`

```
$local/greengrass/telemetry
```

Le composant émetteur de télémétrie Nucleus ne publie pas dans un sujet AWS IoT Core MQTT par défaut. Vous pouvez configurer ce composant pour qu'il soit publié dans une rubrique AWS IoT Core MQTT lorsque vous le déployez. L'utilisation d'une rubrique MQTT pour publier des données sur le AWS Cloud est soumise à une [AWS IoT Core tarification](#).

AWS IoT Greengrass fournit plusieurs [composants communautaires](#) pour vous aider à analyser et à visualiser les données de télémétrie localement sur votre appareil principal à l'aide d'InfluxDB et Grafana. Ces composants utilisent les données de télémétrie du composant émetteur du noyau. Pour plus d'informations, consultez le fichier README du composant éditeur [InfluxDB](#).

Rubriques

- [Versions](#)
- [Type](#)
- [Système d'exploitation](#)
- [Dépendances](#)

- [Configuration](#)
- [Données de sortie](#)
- [Utilisation](#)
- [Fichier journal local](#)
- [Journal des modifications](#)

Versions

Les versions de ce composant sont les suivantes :

- 1,0 x

Type

Ce composant est un composant de plugin (`aws.greengrass.plugin`). Le [noyau Greengrass](#) exécute ce composant dans la même machine virtuelle Java (JVM) que le noyau. Le noyau redémarre lorsque vous modifiez la version de ce composant sur le périphérique principal.

Ce composant utilise le même fichier journal que le noyau Greengrass. Pour de plus amples informations, veuillez consulter [AWS IoT Greengrass Journaux de surveillance](#).

Pour de plus amples informations, veuillez consulter [Types de composants](#).

Système d'exploitation

Ce composant peut être installé sur les appareils principaux qui exécutent les systèmes d'exploitation suivants :

- Linux
- Windows

Dépendances

Lorsque vous déployez un composant, il déploie AWS IoT Greengrass également des versions compatibles de ses dépendances. Cela signifie que vous devez satisfaire aux exigences relatives au composant et à toutes ses dépendances pour réussir le déploiement du composant. Cette section répertorie les dépendances des [versions publiées](#) de ce composant et les contraintes de

version sémantiques qui définissent les versions des composants pour chaque dépendance. Vous pouvez également consulter les dépendances de chaque version du composant dans la [AWS IoT Greengrass console](#). Sur la page de détails du composant, recherchez la liste des dépendances.

1.0.12

Le tableau suivant répertorie les dépendances pour la version 1.0.12 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,4,0$ $< 2,17,0$	Flexible

1.0.11

Le tableau suivant répertorie les dépendances pour la version 1.0.11 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,4,0$ $< 2,16,0$	Stricte

1.0.10

Le tableau suivant répertorie les dépendances pour la version 1.0.10 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,4,0$ $< 2,15,0$	Stricte

1.0.9

Le tableau suivant répertorie les dépendances pour la version 1.0.9 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,4,0$ $< 2,14,0$	Stricte

1.0.8

Le tableau suivant répertorie les dépendances pour la version 1.0.8 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,4,0 <2,13,0	Stricte

1.0.7

Le tableau suivant répertorie les dépendances pour la version 1.0.7 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,4,0 <2,12,0	Stricte

1.0.6

Le tableau suivant répertorie les dépendances pour la version 1.0.6 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,4,0 <2,11,0	Stricte

1.0.5

Le tableau suivant répertorie les dépendances pour la version 1.0.5 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,4,0 <2,1,0	Stricte

1.0.4

Le tableau suivant répertorie les dépendances pour la version 1.0.4 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,4,0 <2,9,0	Stricte

1.0.3

Le tableau suivant répertorie les dépendances pour la version 1.0.3 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,4,0 <2,8,0	Stricte

1.0.2

Le tableau suivant répertorie les dépendances pour la version 1.0.2 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,4,0 <2,7,0	Stricte

1.0.1

Le tableau suivant répertorie les dépendances pour la version 1.0.1 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,4,0 <2,6,0	Stricte

1.0.0

Le tableau suivant répertorie les dépendances pour la version 1.0.0 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,4,0 <2,5,0	Stricte

Pour plus d'informations sur les dépendances des composants, consultez la [référence de la recette des composants](#).

Configuration

Ce composant fournit les paramètres de configuration suivants que vous pouvez personnaliser lorsque vous déployez le composant.

pubSubPublish

(Facultatif) Définit s'il faut publier des données de télémétrie dans le `$local/greengrass/telemetry` sujet. Les valeurs prises en charge sont `true` et `false`.

Par défaut : `true`

mqttTopic

(Facultatif) Rubrique AWS IoT Core MQTT dans laquelle ce composant publie les données de télémétrie.

Définissez cette valeur sur le sujet AWS IoT Core MQTT sur lequel vous souhaitez publier les données de télémétrie. Lorsque cette valeur est vide, l'émetteur du noyau ne publie pas les données de télémétrie dans le AWS Cloud

Note

L'utilisation d'une rubrique MQTT pour publier des données sur le AWS Cloud est soumise à une [AWS IoT Core tarification](#).

Par défaut : `""`

telemetryPublishIntervalMs

(Facultatif) Durée (en millisecondes) entre laquelle le composant publie les données de télémétrie. Si vous définissez cette valeur en dessous de la valeur minimale prise en charge, le composant utilise la valeur minimale à la place.

Note

Des intervalles de publication plus courts se traduisent par une augmentation de l'utilisation du processeur sur votre appareil principal. Nous vous recommandons

de commencer par l'intervalle de publication par défaut et de l'ajuster en fonction de l'utilisation du processeur de votre appareil.

Minimum : 500

Par défaut : 60000

Exemple Exemple : mise à jour de la fusion de configurations

L'exemple suivant montre un exemple de mise à jour de fusion de configuration qui permet de publier des données de télémétrie toutes les 5 secondes sur le `$local/greengrass/telemetry` sujet et le sujet `greengrass/myTelemetry` AWS IoT Core MQTT.

```
{
  "pubSubPublish": "true",
  "mqttTopic": "greengrass/myTelemetry",
  "telemetryPublishIntervalMs": 5000
}
```

Données de sortie

Ce composant publie les métriques de télémétrie sous forme de tableau JSON sur le sujet suivant.

Sujet local : `$local/greengrass/telemetry`

Vous pouvez éventuellement choisir de publier également des métriques de télémétrie dans un sujet AWS IoT Core MQTT. Pour plus d'informations sur les sujets, consultez les [rubriques MQTT](#) dans le Guide du AWS IoT Core développeur.

Exemple Exemple de données

```
[
  {
    "A": "Average",
    "N": "CpuUsage",
    "NS": "SystemMetrics",
    "TS": 1627597331445,
    "U": "Percent",
    "V": 26.21981271562346
  },
  {
```

```
"A": "Count",
"N": "TotalNumberOfFDs",
"NS": "SystemMetrics",
"TS": 1627597331445,
"U": "Count",
"V": 7316
},
{
  "A": "Count",
  "N": "SystemMemUsage",
  "NS": "SystemMetrics",
  "TS": 1627597331445,
  "U": "Megabytes",
  "V": 10098
},
{
  "A": "Count",
  "N": "NumberOfComponentsStarting",
  "NS": "GreengrassComponents",
  "TS": 1627597331446,
  "U": "Count",
  "V": 0
},
{
  "A": "Count",
  "N": "NumberOfComponentsInstalled",
  "NS": "GreengrassComponents",
  "TS": 1627597331446,
  "U": "Count",
  "V": 0
},
{
  "A": "Count",
  "N": "NumberOfComponentsStateless",
  "NS": "GreengrassComponents",
  "TS": 1627597331446,
  "U": "Count",
  "V": 0
},
{
  "A": "Count",
  "N": "NumberOfComponentsStopping",
  "NS": "GreengrassComponents",
  "TS": 1627597331446,
```

```
"U": "Count",
"V": 0
},
{
  "A": "Count",
  "N": "NumberOfComponentsBroken",
  "NS": "GreengrassComponents",
  "TS": 1627597331446,
  "U": "Count",
  "V": 0
},
{
  "A": "Count",
  "N": "NumberOfComponentsRunning",
  "NS": "GreengrassComponents",
  "TS": 1627597331446,
  "U": "Count",
  "V": 7
},
{
  "A": "Count",
  "N": "NumberOfComponentsErrored",
  "NS": "GreengrassComponents",
  "TS": 1627597331446,
  "U": "Count",
  "V": 0
},
{
  "A": "Count",
  "N": "NumberOfComponentsNew",
  "NS": "GreengrassComponents",
  "TS": 1627597331446,
  "U": "Count",
  "V": 0
},
{
  "A": "Count",
  "N": "NumberOfComponentsFinished",
  "NS": "GreengrassComponents",
  "TS": 1627597331446,
  "U": "Count",
  "V": 2
}
```

]

Le tableau de sortie contient une liste de métriques possédant les propriétés suivantes :

A

Type d'agrégation pour la métrique.

Pour la CpuUsage métrique, cette propriété est définie sur Average car la valeur publiée de la métrique est l'utilisation moyenne du processeur depuis le dernier événement de publication.

Pour toutes les autres mesures, l'émetteur du noyau n'agrège pas la valeur de la métrique, et cette propriété est définie sur Count.

N

Le nom de la métrique.

NS

L'espace de noms de la métrique.

TS

Horodatage du moment où les données ont été collectées.

U

Unité de la valeur métrique.

V

Valeur de la métrique .

L'émetteur du noyau publie les métriques suivantes :

Nom	Description	
Système		
SystemMemUsage	La quantité de mémoire actuellement utilisée par toutes les applications du périphérique principal de	

Nom	Description	
	Greengrass, y compris le système d'exploitation.	
CpuUsage	La quantité de processeur actuellement utilisée par toutes les applications du périphérique principal de Greengrass, y compris le système d'exploitation.	
TotalNumberOfFDs	Nombre de descripteurs de fichiers stockés par le système d'exploitation du périphérique principal Greengrass. Un descripteur de fichier identifie de manière unique un fichier ouvert.	
Noyau de Greengrass		
NumberOfComponentsRunning	Le nombre de composants qui s'exécutent sur le périphérique principal de Greengrass.	
NumberOfComponentsErrored	Nombre de composants présentant un état d'erreur sur le périphérique principal de Greengrass.	
NumberOfComponentsInstalled	Le nombre de composants installés sur le périphérique principal de Greengrass.	
NumberOfComponentsStarting	Le nombre de composants qui démarrent sur le périphérique principal de Greengrass.	

Nom	Description	
NumberOfComponents New	Le nombre de composants nouveaux sur le périphérique principal de Greengrass.	
NumberOfComponents Stopping	Le nombre de composants qui s'arrêtent sur le périphérique principal de Greengrass.	
NumberOfComponents Finished	Le nombre de composants terminés sur le périphérique principal Greengrass.	
NumberOfComponents Broken	Le nombre de composants cassés sur le périphérique principal de Greengrass.	
NumberOfComponents Stateless	Le nombre de composants qui sont apatrides sur le périphérique principal de Greengrass.	

Utilisation

Pour utiliser les données de télémétrie relatives à l'état du système, vous pouvez créer des composants personnalisés qui s'abonnent aux rubriques sur lesquelles l'émetteur du noyau publie les données de télémétrie, et qui réagissent à ces données selon les besoins. Le composant Nucleus Emitter offrant la possibilité de publier des données de télémétrie sur un sujet local, vous pouvez vous abonner à ce sujet et utiliser les données publiées pour agir localement sur votre appareil principal. L'appareil principal peut alors réagir aux données de télémétrie même lorsque sa connectivité au cloud est limitée.

Par exemple, vous pouvez configurer un composant qui écoute les données de télémétrie sur le `$local/greengrass/telemetry` sujet et envoie les données au composant du gestionnaire de flux pour qu'il diffuse vos données vers le. AWS Cloud Pour plus d'informations sur la création d'un tel composant, reportez-vous [Publier/souscrire des messages locaux](#) aux sections et [Créez des composants personnalisés utilisant le gestionnaire de flux](#).

Fichier journal local

Ce composant utilise le même fichier journal que le composant [Greengrass nucleus](#).

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

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

Pour consulter les journaux de ce composant

- Exécutez la commande suivante sur le périphérique principal pour afficher le fichier journal de ce composant en temps réel. Remplacez */greengrass/v2* ou *C:\greengrass\v2* par le chemin d'accès au dossier AWS IoT Greengrass racine.

Linux

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

Windows (PowerShell)

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

Journal des modifications

Le tableau suivant décrit les modifications apportées à chaque version du composant.

Version	Modifications
1.0.12	Version mise à jour pour la version 2.16.0 de Greengrass Nucleus.
1.0.11	Version mise à jour pour la version 2.15.0 de Greengrass Nucleus.
1.0.10	Version mise à jour pour la version 2.14.0 de Greengrass Nucleus.

Version	Modifications
1.0.9	Version mise à jour pour la version 2.13.0 de Greengrass Nucleus.
1.0.8	Version mise à jour pour la version 2.12.0 de Greengrass Nucleus.
1.0.7	Version mise à jour pour la version 2.11.0 de Greengrass Nucleus.
1.0.6	Version mise à jour pour la version 2.10.0 de Greengrass Nucleus.
1.0.5	Version mise à jour pour la version 2.9.0 de Greengrass Nucleus.
1.0.4	Version mise à jour pour la version 2.8.0 de Greengrass Nucleus.
1.0.3	Version mise à jour pour la version 2.7.0 de Greengrass Nucleus.
1.0.2	Version mise à jour pour la version 2.6.0 de Greengrass Nucleus.
1.0.1	Version mise à jour pour la version 2.5.0 de Greengrass Nucleus.
1.0.0	Première version.

Fournisseur PKCS #11

Le composant fournisseur PKCS #11 (`aws.greengrass.crypto.Pkcs11Provider`) vous permet de configurer le logiciel AWS IoT Greengrass principal pour utiliser un module de sécurité matériel (HSM) via l'interface [PKCS #11](#). Ce composant vous permet de stocker en toute sécurité les fichiers de certificats et de clés privées afin qu'ils ne soient pas exposés ou dupliqués dans le logiciel. Pour de plus amples informations, veuillez consulter [Intégration de sécurité matérielle](#).

Pour approvisionner un appareil principal Greengrass qui stocke son certificat et sa clé privée dans un HSM, vous devez spécifier ce composant en tant que plug-in de provisionnement lorsque vous installez le logiciel Core. AWS IoT Greengrass Pour de plus amples informations, veuillez consulter [Installation AWS IoT Greengrass du logiciel Core avec provisionnement manuel des ressources](#).

AWS IoT Greengrass fournit ce composant sous forme de fichier JAR que vous pouvez télécharger pour le spécifier en tant que plugin de provisionnement lors de l'installation. Vous pouvez télécharger la dernière version du fichier JAR du composant à l'adresse suivante : <https://d2s8p88vqu9w66.cloudfront.net/releases/Pkcs11Provider/aws.greengrass.crypto.pkcs11Provider-Latest.jar>.

Rubriques

- [Versions](#)
- [Type](#)
- [Système d'exploitation](#)
- [Prérequis](#)
- [Dépendances](#)
- [Configuration](#)
- [Fichier journal local](#)
- [Journal des modifications](#)

Versions

Les versions de ce composant sont les suivantes :

- 2,0.x

Type

Ce composant est un composant de plugin (`aws.greengrass.plugin`). Le [noyau Greengrass](#) exécute ce composant dans la même machine virtuelle Java (JVM) que le noyau. Le noyau redémarre lorsque vous modifiez la version de ce composant sur le périphérique principal.

Ce composant utilise le même fichier journal que le noyau Greengrass. Pour de plus amples informations, veuillez consulter [AWS IoT Greengrass Journaux de surveillance](#).

Pour de plus amples informations, veuillez consulter [Types de composants](#).

Système d'exploitation

Ce composant ne peut être installé que sur les appareils principaux de Linux.

Prérequis

Ce composant répond aux exigences suivantes :

- Module de sécurité matérielle qui prend en charge le schéma de signature [PKCS #1 v1.5](#) et les clés RSA d'une taille de clé RSA-2048 (ou supérieure) ou les clés ECC.

 Note

Pour utiliser un module de sécurité matériel avec des clés ECC, vous devez utiliser [Greengrass](#) nucleus v2.5.6 ou version ultérieure.

Pour utiliser un module de sécurité matériel et un [gestionnaire de secrets](#), vous devez utiliser un module de sécurité matériel doté de clés RSA.

- Une bibliothèque de fournisseur PKCS #11 que le logiciel AWS IoT Greengrass Core peut charger au moment de l'exécution (à l'aide de libdl) pour appeler les fonctions PKCS #11. La bibliothèque du fournisseur PKCS #11 doit implémenter les opérations d'API PKCS #11 suivantes :

- C_Initialize
- C_Finalize
- C_GetSlotList
- C_GetSlotInfo
- C_GetTokenInfo
- C_OpenSession
- C_GetSessionInfo
- C_CloseSession
- C_Login
- C_Logout
- C_GetAttributeValue
- C_FindObjectsInit
- C_FindObjects
- C_FindObjectsFinal
- C_DecryptInit
- C_Decrypt
- C_DecryptUpdate
- C_DecryptFinal
- C_SignInit
- C_Sign
- C_SignUpdate

- C_SignFinal
- C_GetMechanismList
- C_GetMechanismInfo
- C_GetInfo
- C_GetFunctionList
- Le module matériel doit être résolu par étiquette d'emplacement, tel que défini dans la spécification PKCS#11.
- Vous devez stocker la clé privée et le certificat dans le HSM dans le même emplacement, et ils doivent utiliser la même étiquette d'objet et le même ID d'objet, si le HSM prend en charge l'objet. IDs
- Le certificat et la clé privée doivent pouvoir être résolus par des libellés d'objets.
- La clé privée doit disposer des autorisations suivantes :
 - sign
 - decrypt
- (Facultatif) Pour utiliser le [composant secret manager](#), vous devez utiliser la version 2.1.0 ou ultérieure, et la clé privée doit disposer des autorisations suivantes :
 - unwrap
 - wrap
- (Facultatif) Si vous utilisez la TPM2 bibliothèque et exécutez le noyau de Greengrass en tant que service, vous devez fournir une variable d'environnement indiquant l'emplacement du magasin PKCS #11. L'exemple suivant est un fichier de service systemd contenant la variable d'environnement requise :

```
[Unit]
Description=Greengrass Core
After=network.target

[Service]
Type=simple
PIDFile=/var/run/greengrass.pid
Environment=TPM2_PKCS11_STORE=/path/to/store/directory
RemainAfterExit=no
Restart=on-failure
RestartSec=10
ExecStart=/bin/sh /greengrass/v2/alts/current/distro/bin/loader
```

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

Dépendances

Lorsque vous déployez un composant, il déploie AWS IoT Greengrass également des versions compatibles de ses dépendances. Cela signifie que vous devez satisfaire aux exigences relatives au composant et à toutes ses dépendances pour réussir le déploiement du composant. Cette section répertorie les dépendances des [versions publiées](#) de ce composant et les contraintes de version sémantiques qui définissent les versions des composants pour chaque dépendance. Vous pouvez également consulter les dépendances de chaque version du composant dans la [AWS IoT Greengrass console](#). Sur la page de détails du composant, recherchez la liste des dépendances.

2.0.11

Le tableau suivant répertorie les dépendances pour la version 2.0.11 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,5,3$ $< 2,17,0$	Flexible

2.0.10

Le tableau suivant répertorie les dépendances pour la version 2.0.10 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,5,3$ $< 2,16,0$	Flexible

2.0.9

Le tableau suivant répertorie les dépendances pour la version 2.0.9 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,5,3$ $< 2,15,0$	Flexible

2.0.8

Le tableau suivant répertorie les dépendances pour la version 2.0.8 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,5,3 <2,14,0	Flexible

2.0.7

Le tableau suivant répertorie les dépendances pour la version 2.0.7 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,5,3 <2,13,0	Flexible

2.0.6

Le tableau suivant répertorie les dépendances pour la version 2.0.6 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,5,3 <2,12,0	Flexible

2.0.5

Le tableau suivant répertorie les dépendances pour la version 2.0.5 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,5,3 <2,11,0	Flexible

2.0.4

Le tableau suivant répertorie les dépendances pour la version 2.0.4 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,5,3 <2,1,0	Flexible

2.0.3

Le tableau suivant répertorie les dépendances pour la version 2.0.3 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,5,3 <2,9,0	Flexible

2.0.2

Le tableau suivant répertorie les dépendances pour la version 2.0.2 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,5,3 <2,8,0	Flexible

2.0.1

Le tableau suivant répertorie les dépendances pour la version 2.0.1 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,5,3 <2,7,0	Flexible

2.0.0

Le tableau suivant répertorie les dépendances pour la version 2.0.0 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,5,3 <2,6,0	Flexible

Pour plus d'informations sur les dépendances des composants, consultez la [référence de la recette des composants](#).

Configuration

Ce composant fournit les paramètres de configuration suivants que vous pouvez personnaliser lorsque vous déployez le composant.

name

Nom de la configuration PKCS #11.

library

Le chemin de fichier absolu vers la bibliothèque de l'implémentation PKCS #11 que le logiciel AWS IoT Greengrass Core peut charger avec libdl.

slot

L'ID du slot qui contient la clé privée et le certificat de l'appareil. Cette valeur est différente de l'index ou de l'étiquette du slot.

userPin

Le code PIN de l'utilisateur à utiliser pour accéder au slot.

Exemple Exemple : mise à jour de la fusion de configurations

```
{
  "name": "softhsm_pkcs11",
  "library": "/usr/lib/softhsm/libsofthsm2.so",
  "slot": 1,
  "userPin": "1234"
}
```

Fichier journal local

Ce composant utilise le même fichier journal que le composant [Greengrass nucleus](#).

Linux

```
/greengrass/v2/logs/greengrass.log
```


Windows

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

Pour consulter les journaux de ce composant

- Exécutez la commande suivante sur le périphérique principal pour afficher le fichier journal de ce composant en temps réel. Remplacez `/greengrass/v2` ou `C:\greengrass\v2` par le chemin d'accès au dossier AWS IoT Greengrass racine.

Linux

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

Windows (PowerShell)

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

Journal des modifications

Le tableau suivant décrit les modifications apportées à chaque version du composant.

Version	Modifications
2.0.11	Version mise à jour pour la version 2.16.0 de Greengrass Nucleus.
2.0.10	Version mise à jour pour la version 2.15.0 de Greengrass Nucleus.
2.0.9	Version mise à jour pour la version 2.14.0 de Greengrass Nucleus.
2.0.8	Version mise à jour pour la version 2.13.0 de Greengrass Nucleus.
2.0.7	Version mise à jour pour la version 2.12.0 de Greengrass Nucleus.
2.0.6	Version mise à jour pour la version 2.11.0 de Greengrass Nucleus.
2.0.5	Version mise à jour pour la version 2.10.0 de Greengrass Nucleus.

Version	Modifications
2.0.4	Version mise à jour pour la version 2.9.0 de Greengrass Nucleus.
2.0.3	Version mise à jour pour la version 2.8.0 de Greengrass Nucleus.
2.0.2	Version mise à jour pour la version 2.7.0 de Greengrass Nucleus.
2.0.1	Version mise à jour pour la version 2.6.0 de Greengrass Nucleus.
2.0.0	Première version.

Directeur des services secrets

Le composant du gestionnaire de secrets (`aws.greengrass.SecretManager`) déploie les secrets depuis les appareils principaux AWS Secrets Manager de Greengrass. Utilisez ce composant pour utiliser en toute sécurité les informations d'identification, telles que les mots de passe, dans les composants personnalisés de vos principaux appareils Greengrass. Pour plus d'informations sur Secrets Manager, consultez [Qu'est-ce que c'est AWS Secrets Manager ?](#) dans le guide de AWS Secrets Manager l'utilisateur.

Pour accéder aux secrets de ce composant dans vos composants Greengrass personnalisés, utilisez l'[GetSecretValue](#) opération décrite dans le Kit SDK des appareils AWS IoT. Pour plus d'informations, consultez [Utilisez le Kit SDK des appareils AWS IoT pour communiquer avec le noyau de Greengrass, les autres composants et AWS IoT Core](#) et [Récupérez les valeurs secrètes](#).

Ce composant chiffre les secrets sur l'appareil principal afin de protéger vos informations d'identification et vos mots de passe jusqu'à ce que vous ayez besoin de les utiliser. Il utilise la clé privée de l'appareil principal pour chiffrer et déchiffrer les secrets.

Rubriques

- [Versions](#)
- [Type](#)
- [Système d'exploitation](#)
- [Exigences](#)
- [Dépendances](#)
- [Configuration](#)

- [Fichier journal local](#)
- [Journal des modifications](#)

Versions

Les versions de ce composant sont les suivantes :

- 2.2.x
- 2.1.x
- 2.0.x

Type

Ce composant est un composant de plugin (`aws.greengrass.plugin`). Le [noyau Greengrass](#) exécute ce composant dans la même machine virtuelle Java (JVM) que le noyau. Le noyau redémarre lorsque vous modifiez la version de ce composant sur le périphérique principal.

Ce composant utilise le même fichier journal que le noyau Greengrass. Pour de plus amples informations, veuillez consulter [AWS IoT Greengrass Journaux de surveillance](#).

Pour de plus amples informations, veuillez consulter [Types de composants](#).

Système d'exploitation

Ce composant peut être installé sur les appareils principaux qui exécutent les systèmes d'exploitation suivants :

- Linux
- Windows

Exigences

Ce composant répond aux exigences suivantes :

- Le [rôle d'appareil Greengrass](#) doit autoriser `secretsmanager:GetSecretValue` action, comme illustré dans l'exemple de politique IAM suivant.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "secretsmanager:GetSecretValue"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:secretsmanager:us-east-1:123456789012:secret:MySecret"
      ]
    }
  ]
}
```

Note

Si vous utilisez une AWS Key Management Service clé gérée par le client pour chiffrer des secrets, le rôle de l'appareil doit également autoriser l'action. `kms:Decrypt`

Pour plus d'informations sur les politiques IAM pour Secrets Manager, consultez ce qui suit dans le guide de l'AWS Secrets Manager utilisateur :

- [Authentification et contrôle d'accès pour AWS Secrets Manager](#)
- [Actions, ressources et clés de contexte que vous pouvez utiliser dans une politique IAM ou une politique secrète pour AWS Secrets Manager](#)
- Les composants personnalisés doivent définir une politique d'autorisation qui permet `aws:greengrass#GetSecretValue` d'obtenir les secrets que vous stockez avec ce composant. Dans cette politique d'autorisation, vous pouvez restreindre l'accès des composants à des secrets spécifiques. Pour plus d'informations, consultez la section [Autorisation IPC du gestionnaire secret](#).
- (Facultatif) Si vous stockez la clé privée et le certificat du périphérique principal dans un [module de sécurité matériel](#) (HSM), le HSM doit prendre en charge les clés RSA, la clé privée doit avoir l'`unwrapp` autorisation et la clé publique doit avoir l'autorisation. `wrap`

Points de terminaison et ports

Ce composant doit être capable d'effectuer des demandes sortantes vers les points de terminaison et les ports suivants, en plus des points de terminaison et des ports requis pour le fonctionnement de base. Pour de plus amples informations, veuillez consulter [Autoriser le trafic des appareils via un proxy ou un pare-feu](#).

Endpoint	Port	Obligatoire	Description
secretsmanager. <i>region</i> .amazonaws.com	443	Oui	Téléchargez les secrets sur l'appareil principal.

Dépendances

Lorsque vous déployez un composant, il déploie AWS IoT Greengrass également des versions compatibles de ses dépendances. Cela signifie que vous devez satisfaire aux exigences relatives au composant et à toutes ses dépendances pour réussir le déploiement du composant. Cette section répertorie les dépendances des [versions publiées](#) de ce composant et les contraintes de version sémantiques qui définissent les versions des composants pour chaque dépendance. Vous pouvez également consulter les dépendances de chaque version du composant dans la [AWS IoT Greengrass console](#). Sur la page de détails du composant, recherchez la liste des dépendances.

2.2.7

Le tableau suivant répertorie les dépendances pour la version 2.2.7 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,5,0 <2,17,0	Flexible

2.2.6

Le tableau suivant répertorie les dépendances pour la version 2.2.6 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,5,0 <2,16,0	Flexible

2.2.2 – 2.2.5

Le tableau suivant répertorie les dépendances pour les versions 2.2.2 à 2.2.5 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,5,0 <2,15,0	Flexible

2.2.0

Le tableau suivant répertorie les dépendances pour les versions 2.2.0 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,13,0 <2,14,0	Flexible

2.1.7 – 2.1.8

Le tableau suivant répertorie les dépendances pour les versions 2.1.7 et 2.1.8 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,5,0 <2,13,0	Flexible

2.1.6

Le tableau suivant répertorie les dépendances pour la version 2.1.6 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,5,0 <2,12,0	Flexible

2.1.5

Le tableau suivant répertorie les dépendances pour la version 2.1.5 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,5,0 <2,11,0	Flexible

2.1.4

Le tableau suivant répertorie les dépendances pour la version 2.1.4 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,5,0 <2,1,0	Flexible

2.1.3

Le tableau suivant répertorie les dépendances pour la version 2.1.3 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,5,0 <2,9,0	Flexible

2.1.2

Le tableau suivant répertorie les dépendances pour la version 2.1.2 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,5,0 <2,8,0	Flexible

2.1.1

Le tableau suivant répertorie les dépendances pour la version 2.1.1 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,5,0 <2,7,0	Flexible

2.1.0

Le tableau suivant répertorie les dépendances pour la version 2.1.0 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,5,0 <2,6,0	Flexible

2.0.9

Le tableau suivant répertorie les dépendances pour la version 2.0.9 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0,0 <2,5,0	Flexible

2.0.8

Le tableau suivant répertorie les dépendances pour la version 2.0.8 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0,0 <2,4,0	Flexible

2.0.7

Le tableau suivant répertorie les dépendances pour la version 2.0.7 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0,0 <2,3,0	Flexible

2.0.6

Le tableau suivant répertorie les dépendances pour la version 2.0.6 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0,0 <2,2.0	Flexible

2.0.4 and 2.0.5

Le tableau suivant répertorie les dépendances pour les versions 2.0.4 et 2.0.5 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0,3 <2,10	Flexible

Pour plus d'informations sur les dépendances des composants, consultez la [référence de la recette des composants](#).

Configuration

Ce composant fournit les paramètres de configuration suivants que vous pouvez personnaliser lorsque vous déployez le composant.

periodicRefreshIntervalMin (facultatif)

Intervalle en minutes pendant lequel ce composant synchronise les secrets configurés sur le périphérique principal avec les dernières valeurs secrètes du AWS Secrets Manager service. Si cet intervalle n'est pas configuré, le gestionnaire de secrets n'actualisera pas régulièrement les secrets configurés.

```
{
  "cloudSecrets": [
    {
      "arn": "arn:aws:secretsmanager:us-
west-2:123456789012:secret:MyGreengrassSecret-abcdef"
    }
  ],
  "periodicRefreshIntervalMin" : 60
}
```

```
}
```

cloudSecrets

Liste des secrets de Secrets Manager à déployer sur le périphérique principal. Vous pouvez spécifier des étiquettes pour définir les versions de chaque secret à déployer. Si vous ne spécifiez pas de version, ce composant déploie la version avec l'étiquette intermédiaire `AWSCURRENT` attachée. Pour plus d'informations, consultez la section [Étiquettes de mise en scène](#) dans le guide de AWS Secrets Manager l'utilisateur.

Le composant du gestionnaire de secrets met en cache les secrets localement. Si la valeur du secret change dans Secrets Manager, ce composant ne récupère pas automatiquement la nouvelle valeur. Pour mettre à jour la copie locale, attribuez une nouvelle étiquette au secret et configurez ce composant pour récupérer le secret identifié par le nouveau label.

Chaque objet contient les informations suivantes :

`arn`

L'ARN du secret à déployer. L'ARN du secret peut être un ARN complet ou partiel. Nous vous recommandons de spécifier un ARN complet plutôt qu'un ARN partiel. Pour plus d'informations, consultez la section [Recherche d'un secret à partir d'un ARN partiel](#). Voici un exemple d'ARN complet et d'ARN partiel :

- ARN complet : `arn:aws:secretsmanager:us-east-2:111122223333:secret:SecretName-abcdef`
- ARN partiel : `arn:aws:secretsmanager:us-east-2:111122223333:secret:SecretName`

`labels`

(Facultatif) Une liste d'étiquettes pour identifier les versions du secret à déployer sur le périphérique principal.

Chaque étiquette doit être une chaîne.

Exemple Exemple : mise à jour de la fusion de configurations

```
{  
  "cloudSecrets": [  
    {
```

```
"arn": "arn:aws:secretsmanager:us-west-2:123456789012:secret:MyGreengrassSecret-
abcdef"
}
]
```

Fichier journal local

Ce composant utilise le même fichier journal que le composant [Greengrass nucleus](#).

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

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

Pour consulter les journaux de ce composant

- Exécutez la commande suivante sur le périphérique principal pour afficher le fichier journal de ce composant en temps réel. Remplacez */greengrass/v2* ou *C:\greengrass\v2* par le chemin d'accès au dossier AWS IoT Greengrass racine.

Linux

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

Windows (PowerShell)

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

Journal des modifications

Le tableau suivant décrit les modifications apportées à chaque version du composant.

Version	Modifications
2.2.7	Version mise à jour pour la version 2.16.0 de Greengrass Nucleus.
2.2.6	Corrections de bogues et améliorations Résout un problème selon lequel le gestionnaire de secrets ne parvient pas à obtenir un secret en raison de la lenteur ou de l'absence de réponse du module Trusted Platform. Version mise à jour pour la version 2.15.0 de Greengrass Nucleus.
2.2.5	Corrections de bogues et améliorations Résout un problème selon lequel un secret n'est pas extrait du cache local AWS Cloud s'il n'est pas présent dans le cache local.
2.2.4	Corrections de bogues et améliorations Réduit la fréquence des écritures dans le magasin secret local. Le gestionnaire de secrets écrit désormais dans le magasin local uniquement lorsque les secrets sont mis à jour.
2.2.3	Corrections de bogues et améliorations Résout un problème selon lequel le gestionnaire de secrets efface les secrets conservés localement lorsque le périphérique principal est hors ligne et que le service de sécurité de l'appareil (tel qu'un HSM) n'est pas disponible.
2.2.2	Corrections de bogues et améliorations Résout un problème en raison duquel le gestionnaire de secrets ne télécharge pas les secrets configurés avec des arns partiels.
2.2.1	Corrections de bogues et améliorations Supporte le gestionnaire de secrets sur les versions 2.5.0 et supérieures de Nucleus.

Version	Modifications
2.2.0	Nouvelles fonctionnalités Ajoute la prise en charge de l'actualisation périodique des secrets configurés par le biais d'une nouvelle clé de configuration de composant. Ajoute la prise en charge d'un nouveau paramètre de demande dans la requête <code>GetSecretValue</code> IPC pour actualiser les secrets par demande
2.1.8	Corrections de bogues et améliorations Résout un problème selon lequel le gestionnaire de secrets n'acceptait pas un ARN partiel.
2.1.7	Version mise à jour pour la version 2.12.0 de Greengrass Nucleus.
2.1.6	Version mise à jour pour la version 2.11.0 de Greengrass Nucleus.
2.1.5	Version mise à jour pour la version 2.10.0 de Greengrass Nucleus.
2.1.4	Corrections de bogues et améliorations Résout un problème en raison duquel les secrets mis en cache étaient supprimés lors du déploiement du gestionnaire de secrets et du redémarrage du noyau de Greengrass. Version mise à jour pour la version 2.9.0 de Greengrass Nucleus.
2.1.3	Version mise à jour pour la version 2.8.0 de Greengrass Nucleus.
2.1.2	Version mise à jour pour la version 2.7.0 de Greengrass Nucleus.
2.1.1	Version mise à jour pour la version 2.6.0 de Greengrass Nucleus.

Version	Modifications
2.1.0	Nouvelles fonctionnalités - Ajoute la prise en charge de l'intégration de la sécurité matérielle. Le composant du gestionnaire de secrets peut chiffrer et déchiffrer des secrets à l'aide d'une clé privée que vous stockez dans un module de sécurité matériel (HSM). Pour de plus amples informations, veuillez consulter Intégration de sécurité matérielle. Corrections de bogues et améliorations - Version mise à jour pour la version 2.5.0 de Greengrass Nucleus.
2.0.9	Version mise à jour pour la version 2.4.0 de Greengrass Nucleus.
2.0.8	Version mise à jour pour la version 2.3.0 de Greengrass Nucleus.
2.0.7	Version mise à jour pour la version 2.2.0 de Greengrass Nucleus.
2.0.6	Version mise à jour pour la version 2.1.0 de Greengrass Nucleus.
2.0.5	Améliorations Ajoutez le support pour les régions et AWS GovCloud (US) les régions de AWS Chine.
2.0.4	Première version.

Tunneling sécurisé

Avec le `aws.greengrass.SecureTunneling` composant, vous pouvez établir une communication bidirectionnelle sécurisée avec un appareil principal de Greengrass situé derrière des pare-feux restreints.

Par exemple, imaginez que vous avez un appareil central Greengrass derrière un pare-feu qui interdit toutes les connexions entrantes. Le tunneling sécurisé utilise MQTT pour transférer un jeton d'accès à l'appareil, puis WebSockets pour établir une connexion SSH avec l'appareil via le pare-feu. Avec ce tunnel AWS IoT géré, vous pouvez ouvrir la connexion SSH requise pour votre appareil. Pour plus d'informations sur l'utilisation du tunneling AWS IoT sécurisé pour se connecter à des

appareils distants, consultez la section relative au [tunneling AWS IoT sécurisé](#) dans le guide du développeur.AWS IoT

Ce composant s'abonne au courtier de messages AWS IoT Core MQTT sur le `$aws/things/greengrass-core-device/tunnels/notify` sujet pour recevoir des notifications de tunneling sécurisé.

Rubriques

- [Versions](#)
- [Type](#)
- [Système d'exploitation](#)
- [Prérequis](#)
- [Dépendances](#)
- [Configuration](#)
- [Fichier journal local](#)
- [Licences](#)
- [Utilisation](#)
- [Consultez aussi](#)
- [Journal des modifications](#)

Versions

Les versions de ce composant sont les suivantes :

- 1,1x
- 1,0 x

Type

Ce composant est un composant générique (`aws.greengrass.generic`). Le [noyau Greengrass](#) exécute les scripts de cycle de vie du composant.

Pour de plus amples informations, veuillez consulter [Types de composants](#).

Système d'exploitation

Ce composant ne peut être installé que sur les appareils principaux de Linux.

Architectures :

- Armv 71
- Bras (8) AArch64
- x86_64

Prérequis

Ce composant répond aux exigences suivantes :

- Au moins 32 Mo d'espace disque disponible pour le composant de tunneling sécurisé. Cette exigence n'inclut pas le logiciel principal de Greengrass ni les autres composants exécutés sur le même appareil.
- Minimum de 16 Mo de RAM disponible pour le composant de tunneling sécurisé. Cette exigence n'inclut pas le logiciel principal de Greengrass ni les autres composants exécutés sur le même appareil. Pour de plus amples informations, veuillez consulter [Contrôlez l'allocation de mémoire grâce aux options JVM](#).
- La version 2.25 ou supérieure de la bibliothèque GNU C (glibc) avec un noyau Linux 3.2 ou supérieur est requise pour le composant de tunneling sécurisé version 1.0.12 et supérieure. Les versions du système d'exploitation et des bibliothèques ayant dépassé la date de fin de vie de leur support à long terme ne sont pas prises en charge. Vous devez utiliser un système d'exploitation et des bibliothèques bénéficiant d'un support à long terme.
- Le système d'exploitation et le moteur d'exécution Java doivent être installés en 64 bits.
- [Python](#) 3.5 ou version ultérieure installé sur le périphérique principal de Greengrass et ajouté à la variable d'environnement PATH.
- `libcrypto.so.1.1` installé sur le périphérique principal de Greengrass et ajouté à la variable d'environnement PATH.
- Ouvrez le trafic sortant sur le port 443 de l'appareil principal de Greengrass.
- Activez le support pour le service de communication que vous souhaitez utiliser pour communiquer avec l'appareil principal de Greengrass. Par exemple, pour ouvrir une connexion SSH à l'appareil, vous devez activer SSH sur cet appareil.

Points de terminaison et ports

Ce composant doit être capable d'effectuer des demandes sortantes vers les points de terminaison et les ports suivants, en plus des points de terminaison et des ports requis pour le fonctionnement de base. Pour de plus amples informations, veuillez consulter [Autoriser le trafic des appareils via un proxy ou un pare-feu](#).

Endpoint	Port	Obligatoire	Description
<code>data.tunneling.iot</code> <code>. <i>region</i>.amazonaws.com</code>	443	Oui	Établissez des tunnels sécurisés.

Dépendances

Lorsque vous déployez un composant, il déploie AWS IoT Greengrass également des versions compatibles de ses dépendances. Cela signifie que vous devez satisfaire aux exigences relatives au composant et à toutes ses dépendances pour réussir le déploiement du composant. Cette section répertorie les dépendances des [versions publiées](#) de ce composant et les contraintes de version sémantiques qui définissent les versions des composants pour chaque dépendance. Vous pouvez également consulter les dépendances de chaque version du composant dans la [AWS IoT Greengrass console](#). Sur la page de détails du composant, recherchez la liste des dépendances.

1.0.19 – 1.1.3

Le tableau suivant répertorie les dépendances pour les versions 1.0.19 à 1.1.3 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	<code>>=2,0.0 <3,0.0</code>	Flexible

1.0.18

Le tableau suivant répertorie les dépendances pour la version 1.0.18 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0,0 <2,13,0	Flexible

1.0.16 – 1.0.17

Le tableau suivant répertorie les dépendances pour les versions 1.0.16 à 1.0.17 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0,0 <2,12,0	Flexible

1.0.14 – 1.0.15

Le tableau suivant répertorie les dépendances pour les versions 1.0.14 à 1.0.15 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0,0 <2,11,0	Flexible

1.0.11 – 1.0.13

Le tableau suivant répertorie les dépendances pour les versions 1.0.11 à 1.0.13 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0,0 <2,1,0	Flexible

1.0.10

Le tableau suivant répertorie les dépendances pour la version 1.0.10 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0,0 <2,9,0	Flexible

1.0.9

Le tableau suivant répertorie les dépendances pour la version 1.0.9 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,8.0	Flexible

1.0.8

Le tableau suivant répertorie les dépendances pour la version 1.0.8 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,7.0	Flexible

1.0.5 - 1.0.7

Le tableau suivant répertorie les dépendances pour les versions 1.0.5 à 1.0.7 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,6.0	Flexible

1.0.4

Le tableau suivant répertorie les dépendances pour la version 1.0.4 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,5.0	Flexible

1.0.3

Le tableau suivant répertorie les dépendances pour la version 1.0.3 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,4.0	Flexible

1.0.2

Le tableau suivant répertorie les dépendances pour la version 1.0.2 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,3.0	Flexible

1.0.1

Le tableau suivant répertorie les dépendances pour la version 1.0.1 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,2.0	Flexible

1.0.0

Le tableau suivant répertorie les dépendances pour la version 1.0.0 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0,3 <2,10	Flexible

Pour plus d'informations sur les dépendances des composants, consultez la [référence de la recette des composants](#).

Configuration

Ce composant fournit les paramètres de configuration suivants que vous pouvez personnaliser lorsque vous déployez le composant.

OS_DIST_INFO

(Facultatif) Le système d'exploitation de votre appareil principal. Par défaut, le composant tente d'identifier automatiquement le système d'exploitation exécuté sur votre appareil principal. Si le composant ne démarre pas avec la valeur par défaut, utilisez cette valeur pour spécifier le système d'exploitation. Pour obtenir la liste des systèmes d'exploitation pris en charge pour ce composant, consultez [Exigences relatives aux dispositifs](#).

Cette valeur peut être l'une des suivantes : `autobuntu`, `amzn2`, `raspberrypi`.

Par défaut : `auto`

accessControl

(Facultatif) Objet contenant la [politique d'autorisation](#) permettant au composant de s'abonner à la rubrique relative aux notifications relatives au tunneling sécurisé.

Note

Ne modifiez pas ce paramètre de configuration si votre déploiement cible un groupe d'objets. Si votre déploiement cible un appareil principal individuel et que vous souhaitez limiter son abonnement au sujet de l'appareil, spécifiez le nom de l'objet du périphérique principal. Dans la `resources` valeur de la politique d'autorisation de l'appareil, remplacez le caractère générique du sujet MQTT par le nom de l'objet de l'appareil.

```
{
  "aws.greengrass.ipc.mqttproxy": {
    "aws.iot.SecureTunneling:mqttproxy:1": {
      "policyDescription": "Access to tunnel notification pubsub topic",
      "operations": [
        "aws.greengrass#SubscribeToIoTCore"
      ],
      "resources": [
        "$aws/things/+/tunnels/notify"
      ]
    }
  }
}
```

Exemple Exemple : mise à jour de la fusion de configurations

L'exemple de configuration suivant indique d'autoriser ce composant à ouvrir des tunnels sécurisés sur un périphérique principal nommé **MyGreengrassCore** qui exécute Ubuntu.

```
{
  "OS_DIST_INFO": "ubuntu",
  "accessControl": {
    "aws.greengrass.ipc.mqttproxy": {
      "aws.iot.SecureTunneling:mqttproxy:1": {
        "policyDescription": "Access to tunnel notification pubsub topic",
        "operations": [
          "aws.greengrass#SubscribeToIoTCore"
        ],
        "resources": [
          "$aws/things/MyGreengrassCore/tunnels/notify"
        ]
      }
    }
  }
}
```

Fichier journal local

Ce composant utilise le fichier journal suivant.

```
/greengrass/v2/logs/aws.greengrass.SecureTunneling.log
```

Pour consulter les journaux de ce composant

- Exécutez la commande suivante sur le périphérique principal pour afficher le fichier journal de ce composant en temps réel. Remplacez */greengrass/v2* par le chemin d'accès au dossier AWS IoT Greengrass racine.

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

Licences

Ce composant inclut les logiciels/licences tiers suivants :

- [AWS IoT Device Client/Apache](#) License 2.0
- [Kit SDK des appareils AWS IoT pour Java](#)/Licence Apache 2.0
- [Licence gson/Apache](#) 2.0
- [Log4j](#) /Licence Apache 2.0
- [licence slf4j /Apache](#) 2.0

Utilisation

Pour utiliser le composant de tunneling sécurisé sur votre appareil, procédez comme suit :

1. Déployez le composant de tunneling sécurisé sur votre appareil.
2. Ouvrez la [AWS IoT console](#). Dans le menu de gauche, choisissez Actions à distance, puis Tunnels sécurisés.
3. Créez un tunnel vers votre appareil Greengrass.
4. Téléchargez le jeton d'accès à la source.
5. Utilisez le proxy local avec le jeton d'accès à la source pour vous connecter à votre destination. Pour plus d'informations, consultez la section [Comment utiliser le proxy local](#) dans le Guide du AWS IoT développeur.

Consultez aussi

- [AWS IoT tunneling sécurisé dans le](#) guide du développeur AWS IoT
- [Comment utiliser le proxy local](#) dans le guide du AWS IoT développeur

Journal des modifications

Le tableau suivant décrit les modifications apportées à chaque version du composant.

Version	Modifications
1.1.3	Corrections de bogues et améliorations • Met à niveau le client de AWS IoT périphérique sous-jacent invoqué par le composant de la version 1.10.0 à la version 1.10.1.

Version	Modifications
	• Résout les problèmes de compatibilité de la bibliothèque GNU C (glibc) dans la version 1.1.2 du composant de tunneling sécurisé.
1.1.2	 Warning Cette version n'est plus disponible. Les améliorations apportées à cette version sont disponibles dans les versions ultérieures de ce composant. Corrections de bogues et améliorations • Met à niveau le client de AWS IoT périphérique sous-jacent invoqué par le composant de la version 1.9.0 à la version 1.10.0. • Résout le problème de transfert de charge utile qui empêchait les utilisateurs de transférer des fichiers volumineux des appareils principaux de Greengrass V2 vers le périphérique source via le tunnel sécurisé.
1.1.1	Corrections de bogues et améliorations • Ajoute une configuration pour prendre en charge Greengrass Nucleus Lite.
1.1.0	Nouvelles fonctionnalités • Ajoute des supports de recettes pour Greengrass Nucleus Lite.

Version	Modifications
1,019	Corrections de bogues et améliorations • Met à niveau le client de AWS IoT périphérique sous-jacent invoqué par le composant de la version 1.8.0 à la version 1.9.0. • Augmente la limite de tunnels simultanés à 20 tunnels au niveau du composant. • Augmente le délai d'expiration du AWS IoT Greengrass Core IPC par défaut de 3 secondes à 10 secondes.  Warning Si vous utilisez le proxy local de tunneling sécurisé comme client source du tunnel, ne mettez pas à jour votre composant vers cette version tant que vous n'avez pas également mis à niveau le proxy local vers la version 3.1.1 ou ultérieure.
10,18	Version mise à jour pour la version 2.12.0 de Greengrass Nucleus.
1,017	Corrections de bogues et améliorations • Résout le problème de nettoyage des threads qui empêchait les utilisateurs de créer des tunnels. Ce composant nettoiera désormais un thread soit une fois qu'il aura reçu le CloseTunnel signal, soit si le tunnel est expiré au bout de 12 heures.
1,0,16	Version mise à jour pour la version 2.11.0 de Greengrass Nucleus.
1.0.15	Corrections de bogues et améliorations • Résout un problème de démarrage pour les utilisateurs qui n'ont pas de répertoire personnel sur l'appareil. Le composant de tunneling sécurisé démarre désormais sans créer de répertoire pour les documents fantômes.
10,14	Version mise à jour pour la version 2.10.0 de Greengrass Nucleus.

Version	Modifications
1,0,13	Corrections de bogues et améliorations • Résout un problème selon lequel un processus client orphelin empêchait plusieurs tunnels de cibler l'appareil.
1.0.12	Corrections de bogues et améliorations • Ajoute le support pour x86_64 (AMD64) et ARMv8 (Aarch64) lors de l'exécution sur le système d'exploitation Raspberry Pi.
1.0.11	Version mise à jour pour la version 2.9.0 de Greengrass Nucleus.
1.0.10	Version mise à jour pour la version 2.8.0 de Greengrass Nucleus.
1.0.9	Version mise à jour pour la version 2.7.0 de Greengrass Nucleus.
1.0.8	Version mise à jour pour la version 2.6.0 de Greengrass Nucleus.
1.0.7	Corrections de bogues et améliorations • Résout un problème de déconnexion du composant lorsque vous transférez des fichiers volumineux via SCP.
1.0.6	Cette version contient des corrections de bogues.
1.0.5	Version mise à jour pour la version 2.5.0 de Greengrass Nucleus.
1.0.4	Version mise à jour pour la version 2.4.0 de Greengrass Nucleus.
1.0.3	Version mise à jour pour la version 2.3.0 de Greengrass Nucleus.
1.0.2	Version mise à jour pour la version 2.2.0 de Greengrass Nucleus.
1.0.1	Version mise à jour pour la version 2.1.0 de Greengrass Nucleus.
1.0.0	Première version.

Shadow Manager

Le composant Shadow Manager (`aws.greengrass.ShadowManager`) active le service fantôme local sur votre appareil principal. Le service local d'ombre permet aux composants d'utiliser la communication entre processus pour [interagir avec les ombres locales](#). Le composant Shadow Manager gère le stockage des documents d'ombre locaux et gère également la synchronisation des états d'ombre locaux avec le service AWS IoT Device Shadow.

Pour plus d'informations sur la manière dont les appareils Greengrass Core peuvent interagir avec les ombres, consultez. [Interagissez avec les ombres de l'appareil](#)

Rubriques

- [Versions](#)
- [Type](#)
- [Système d'exploitation](#)
- [Prérequis](#)
- [Dépendances](#)
- [Configuration](#)
- [Fichier journal local](#)
- [Journal des modifications](#)

Versions

Les versions de ce composant sont les suivantes :

- 2.3.x
- 2.2.x
- 2,1x
- 2,0.x

Type

Ce composant est un composant de plugin (`aws.greengrass.plugin`). Le [noyau Greengrass](#) exécute ce composant dans la même machine virtuelle Java (JVM) que le noyau. Le noyau redémarre lorsque vous modifiez la version de ce composant sur le périphérique principal.

Ce composant utilise le même fichier journal que le noyau Greengrass. Pour de plus amples informations, veuillez consulter [AWS IoT Greengrass Journaux de surveillance](#).

Pour de plus amples informations, veuillez consulter [Types de composants](#).

Système d'exploitation

Ce composant peut être installé sur les appareils principaux qui exécutent les systèmes d'exploitation suivants :

- Linux
- Windows

Prérequis

Ce composant répond aux exigences suivantes :

- (Facultatif) Pour synchroniser les ombres avec le service AWS IoT Device Shadow, la AWS IoT politique du périphérique principal de Greengrass doit autoriser les actions de politique AWS IoT Core parallèle suivantes :
 - `iot:GetThingShadow`
 - `iot:UpdateThingShadow`
 - `iot:DeleteThingShadow`

Pour plus d'informations sur ces AWS IoT Core politiques, voir les [actions AWS IoT Core politiques](#) dans le Guide du AWS IoT développeur.

Pour plus d'informations sur la AWS IoT politique minimale, voir [AWS IoT Politique minimale pour les appareils AWS IoT Greengrass V2 principaux](#)

- Le composant Shadow Manager est compatible pour s'exécuter dans un VPC.

Dépendances

Lorsque vous déployez un composant, il déploie AWS IoT Greengrass également des versions compatibles de ses dépendances. Cela signifie que vous devez satisfaire aux exigences relatives au composant et à toutes ses dépendances pour réussir le déploiement du composant. Cette section répertorie les dépendances des [versions publiées](#) de ce composant et les contraintes de

version sémantiques qui définissent les versions des composants pour chaque dépendance. Vous pouvez également consulter les dépendances de chaque version du composant dans la [AWS IoT Greengrass console](#). Sur la page de détails du composant, recherchez la liste des dépendances.

2.3.12

Le tableau suivant répertorie les dépendances pour la version 2.3.12 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,5,0$ $< 2,17,0$	Flexible

2.3.11

Le tableau suivant répertorie les dépendances pour la version 2.3.11 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,5,0$ $< 2,16,0$	Flexible

2.3.10

Le tableau suivant répertorie les dépendances pour la version 2.3.10 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,5,0$ $< 2,15,0$	Flexible

2.3.9

Le tableau suivant répertorie les dépendances pour la version 2.3.9 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,5,0$ $< 2,14,0$	Flexible

2.3.5 – 2.3.8

Le tableau suivant répertorie les dépendances pour les versions 2.3.5 à 2.3.8 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,5,0$ $< 2,13,0$	Flexible

2.3.3 and 2.3.4

Le tableau suivant répertorie les dépendances pour les versions 2.3.3 et 2.3.4 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,5,0$ $< 2,12,0$	Flexible

2.3.2

Le tableau suivant répertorie les dépendances pour la version 2.3.2 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,5,0$ $< 2,11,0$	Flexible

2.3.0 and 2.3.1

Le tableau suivant répertorie les dépendances pour les versions 2.3.0 et 2.3.1 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,5,0$ $< 2,1,0$	Flexible

2.2.3 and 2.2.4

Le tableau suivant répertorie les dépendances pour les versions 2.2.3 et 2.2.4 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,2,0 <3,0,0	Flexible

2.2.2

Le tableau suivant répertorie les dépendances pour la version 2.2.2 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,2,0 <2,9,0	Flexible

2.2.1

Le tableau suivant répertorie les dépendances pour la version 2.2.1 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,2,0 <2,8,0	Flexible

2.1.1 and 2.2.0

Le tableau suivant répertorie les dépendances pour les versions 2.1.1 et 2.2.0 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,2,0 <2,7,0	Flexible

2.0.5 - 2.1.0

Le tableau suivant répertorie les dépendances pour les versions 2.0.5 à 2.1.0 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,2,0 <2,6,0	Flexible

2.0.3 and 2.0.4

Le tableau suivant répertorie les dépendances pour les versions 2.0.3 et 2.0.4 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,2,0 <2,5,0	Flexible

2.0.1 and 2.0.2

Le tableau suivant répertorie les dépendances pour les versions 2.0.1 et 2.0.2 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,2,0 <2,4,0	Flexible

2.0.0

Le tableau suivant répertorie les dépendances pour la version 2.0.0 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,2,0 <2,3,0	Flexible

Pour plus d'informations sur les dépendances des composants, consultez la [référence de la recette des composants](#).

Configuration

Ce composant fournit les paramètres de configuration suivants que vous pouvez personnaliser lorsque vous déployez le composant.

2.3.x

strategy

(Facultatif) Stratégie utilisée par ce composant pour synchroniser les ombres entre AWS IoT Core et le périphérique principal.

Cet objet contient les informations suivantes.

type

(Facultatif) Type de stratégie utilisé par ce composant pour synchroniser les ombres entre AWS IoT Core et le périphérique principal. Sélectionnez parmi les options suivantes :

- `realTime`— Synchronisez les ombres avec AWS IoT Core chaque mise à jour des ombres.
- `periodic`— Synchronisez les ombres avec AWS IoT Core un intervalle régulier que vous spécifiez avec le paramètre `delay` de configuration.

Par défaut : `realTime`

delay

(Facultatif) Intervalle en secondes avec lequel ce composant synchronise les ombres AWS IoT Core, lorsque vous spécifiez la stratégie de `periodic` synchronisation.

Note

Ce paramètre est obligatoire si vous spécifiez la stratégie de `periodic` synchronisation.

synchronize

(Facultatif) Les paramètres de synchronisation qui déterminent la manière dont les ombres sont synchronisées avec. AWS Cloud

Note

Vous devez créer une mise à jour de configuration avec cette propriété pour synchroniser les ombres avec AWS Cloud.

Cet objet contient les informations suivantes.

coreThing

(Facultatif) Les ombres de l'appareil principal à synchroniser. Cet objet contient les informations suivantes.

classic

(Facultatif) Par défaut, le gestionnaire de shadow synchronise l'état local du shadow classique de votre appareil principal avec le AWS Cloud. Si vous ne souhaitez pas synchroniser le shadow classique de l'appareil, réglez-le sur `false`.

Par défaut : `true`

namedShadows

(Facultatif) La liste des ombres du périphérique principal nommées à synchroniser. Vous devez indiquer le nom exact des ombres.



Warning

Le AWS IoT Greengrass service utilise le `AWSThingsShadow` nom shadow pour gérer les déploiements qui ciblent des appareils principaux individuels. Cette ombre nommée est réservée à l'usage du AWS IoT Greengrass service. Ne mettez pas à jour ou ne supprimez pas cette ombre nommée.

shadowDocumentsMap

(Facultatif) Les ombres supplémentaires de l'appareil à synchroniser. L'utilisation de ce paramètre de configuration facilite la définition de documents fictifs. Nous vous recommandons d'utiliser ce paramètre à la place de l'`shadowDocument` objet.



Note

Si vous spécifiez un `shadowDocumentsMap` objet, vous ne devez pas le spécifier. `shadowDocuments`

Chaque objet contient les informations suivantes :

thingName

La configuration d'ombre *thingName* pour cette configuration d'ombre.

`classic`

(Facultatif) Si vous ne souhaitez pas synchroniser le shadow classique de l'`thingName` appareil, réglez-le sur `false`.

`namedShadows`

La liste des ombres nommées que vous souhaitez synchroniser. Vous devez indiquer le nom exact des ombres.

`shadowDocuments`

(Facultatif) La liste des ombres supplémentaires de l'appareil à synchroniser. Nous vous recommandons d'utiliser le `shadowDocumentsMap` paramètre à la place.

Note

Si vous spécifiez un `shadowDocuments` objet, vous ne devez pas le spécifier. `shadowDocumentsMap`

Chaque objet de cette liste contient les informations suivantes.

`thingName`

Nom de l'objet de l'appareil pour lequel les ombres doivent être synchronisées.

`classic`

(Facultatif) Si vous ne souhaitez pas synchroniser le shadow classique de l'`thingName` appareil, réglez-le sur `false`.

Par défaut : `true`

`namedShadows`

(Facultatif) La liste des ombres de périphériques nommées que vous souhaitez synchroniser. Vous devez indiquer le nom exact des ombres.

`direction`

(Facultatif) La direction dans laquelle synchroniser les ombres entre le service d'ombre local et le AWS Cloud. Vous pouvez configurer cette option pour réduire la bande passante et les connexions au AWS Cloud. Sélectionnez parmi les options suivantes :

- `betweenDeviceAndCloud`— Synchronise les ombres entre le service parallèle local et le AWS Cloud.
- `deviceToCloud`— Envoyez des mises à jour instantanées depuis le service parallèle local vers le AWS Cloud, et ignorez les mises à jour instantanées depuis le AWS Cloud.
- `cloudToDevice`— Recevez des mises à jour fictives depuis le AWS Cloud, et n'envoyez pas de mises à jour fictives depuis le service parallèle local vers le AWS Cloud.

Par défaut : `BETWEEN_DEVICE_AND_CLOUD`

`rateLimits`

(Facultatif) Les paramètres qui déterminent les limites de débit pour les demandes de service parallèle.

Cet objet contient les informations suivantes.

`maxOutboundSyncUpdatesPerSecond`

(Facultatif) Nombre maximal de demandes de synchronisation par seconde transmises par l'appareil.

Par défaut : 100 requêtes/seconde

`maxTotalLocalRequestsRate`

(Facultatif) Le nombre maximum de demandes IPC locales par seconde envoyées au périphérique principal.

Par défaut : 200 requêtes/seconde

`maxLocalRequestsPerSecondPerThing`

(Facultatif) Le nombre maximum de demandes IPC locales par seconde envoyées pour chaque objet IoT connecté.

Par défaut : 20 requests/second pour chaque objet

Note

Ces paramètres de limites de débit définissent le nombre maximal de demandes par seconde pour le service parallèle local. Le nombre maximum de demandes par

seconde pour le service AWS IoT Device Shadow dépend de votre Région AWS. Pour plus d'informations, consultez les limites de l'[API AWS IoT Device Shadow Service](#) dans le Référence générale d'Amazon Web Services.

shadowDocumentSizeLimitBytes

(Facultatif) Taille maximale autorisée de chaque document d'état JSON pour les ombres locales.

Si vous augmentez cette valeur, vous devez également augmenter la limite de ressources pour le document d'état JSON pour les ombres des nuages. Pour plus d'informations, consultez les limites de l'[API AWS IoT Device Shadow Service](#) dans le Référence générale d'Amazon Web Services.

Par défaut : 8192 octets

Maximum : 30720 octets

Exemple Exemple : mise à jour de la fusion de configurations

L'exemple suivant montre un exemple de mise à jour de fusion de configuration avec tous les paramètres de configuration disponibles pour le composant Shadow Manager.

```
{
  "strategy":{
    "type":"periodic",
    "delay":300
  },
  "synchronize":{
    "shadowDocumentsMap":{
      "MyDevice1":{
        "classic":false,
        "namedShadows":[
          "MyShadowA",
          "MyShadowB"
        ]
      },
      "MyDevice2":{
        "classic":true,
        "namedShadows":[]
      }
    }
  }
}
```

```
    }  
    },  
    "direction": "betweenDeviceAndCloud"  
  },  
  "rateLimits": {  
    "maxOutboundSyncUpdatesPerSecond": 100,  
    "maxTotalLocalRequestsRate": 200,  
    "maxLocalRequestsPerSecondPerThing": 20  
  },  
  "shadowDocumentSizeLimitBytes": 8192  
}
```

2.2.x

strategy

(Facultatif) Stratégie utilisée par ce composant pour synchroniser les ombres entre AWS IoT Core et le périphérique principal.

Cet objet contient les informations suivantes.

type

(Facultatif) Type de stratégie utilisé par ce composant pour synchroniser les ombres entre AWS IoT Core et le périphérique principal. Sélectionnez parmi les options suivantes :

- **realTime**— Synchronisez les ombres avec AWS IoT Core chaque mise à jour des ombres.
- **periodic**— Synchronisez les ombres avec AWS IoT Core un intervalle régulier que vous spécifiez avec le paramètre `delay` de configuration.

Par défaut : **realTime**

delay

(Facultatif) Intervalle en secondes avec lequel ce composant synchronise les ombres AWS IoT Core, lorsque vous spécifiez la stratégie de **periodic** synchronisation.

Note

Ce paramètre est obligatoire si vous spécifiez la stratégie de **periodic** synchronisation.

synchronize

(Facultatif) Les paramètres de synchronisation qui déterminent la manière dont les ombres sont synchronisées avec AWS Cloud

Note

Vous devez créer une mise à jour de configuration avec cette propriété pour synchroniser les ombres avec AWS Cloud.

Cet objet contient les informations suivantes.

coreThing

(Facultatif) Les ombres de l'appareil principal à synchroniser. Cet objet contient les informations suivantes.

classic

(Facultatif) Par défaut, le gestionnaire de shadow synchronise l'état local du shadow classique de votre appareil principal avec le AWS Cloud. Si vous ne souhaitez pas synchroniser le shadow classique de l'appareil, réglez-le sur `false`.

Par défaut : `true`

namedShadows

(Facultatif) La liste des ombres du périphérique principal nommées à synchroniser. Vous devez indiquer le nom exact des ombres.

Warning

Le AWS IoT Greengrass service utilise le `AWSManagedGreengrassV2Deployment` nom shadow pour gérer les déploiements qui ciblent des appareils principaux individuels. Cette ombre nommée est réservée à l'usage du AWS IoT Greengrass service. Ne mettez pas à jour ou ne supprimez pas cette ombre nommée.

shadowDocumentsMap

(Facultatif) Les ombres supplémentaires de l'appareil à synchroniser. L'utilisation de ce paramètre de configuration facilite la définition de documents fictifs. Nous vous recommandons d'utiliser ce paramètre à la place de l'`shadowDocument` objet.

Note

Si vous spécifiez un `shadowDocumentsMap` objet, vous ne devez pas le spécifier. `shadowDocuments`

Chaque objet contient les informations suivantes :

thingName

La configuration d'ombre *thingName* pour cette configuration d'ombre.

`classic`

(Facultatif) Si vous ne souhaitez pas synchroniser le shadow classique de l'`thingName` appareil, réglez-le sur `false`.

`namedShadows`

La liste des ombres nommées que vous souhaitez synchroniser. Vous devez indiquer le nom exact des ombres.

`shadowDocuments`

(Facultatif) La liste des ombres supplémentaires de l'appareil à synchroniser. Nous vous recommandons d'utiliser le `shadowDocumentsMap` paramètre à la place.

Note

Si vous spécifiez un `shadowDocuments` objet, vous ne devez pas le spécifier. `shadowDocumentsMap`

Chaque objet de cette liste contient les informations suivantes.

`thingName`

Nom de l'objet de l'appareil pour lequel les ombres doivent être synchronisées.

`classic`

(Facultatif) Si vous ne souhaitez pas synchroniser le shadow classique de l'thingNameappareil, réglez-le sur `false`.

Par défaut : `true`

`namedShadows`

(Facultatif) La liste des ombres de périphériques nommées que vous souhaitez synchroniser. Vous devez indiquer le nom exact des ombres.

`direction`

(Facultatif) La direction dans laquelle synchroniser les ombres entre le service d'ombre local et le AWS Cloud. Vous pouvez configurer cette option pour réduire la bande passante et les connexions au AWS Cloud. Sélectionnez parmi les options suivantes :

- `betweenDeviceAndCloud`— Synchronise les ombres entre le service parallèle local et le AWS Cloud.
- `deviceToCloud`— Envoyez des mises à jour instantanées depuis le service parallèle local vers le AWS Cloud, et ignorez les mises à jour instantanées depuis le AWS Cloud.
- `cloudToDevice`— Recevez des mises à jour fictives depuis le AWS Cloud, et n'envoyez pas de mises à jour fictives depuis le service parallèle local vers le AWS Cloud.

Par défaut : `BETWEEN_DEVICE_AND_CLOUD`

`rateLimits`

(Facultatif) Les paramètres qui déterminent les limites de débit pour les demandes de service parallèle.

Cet objet contient les informations suivantes.

`maxOutboundSyncUpdatesPerSecond`

(Facultatif) Nombre maximal de demandes de synchronisation par seconde transmises par l'appareil.

Par défaut : 100 requêtes/seconde

maxTotalLocalRequestsRate

(Facultatif) Le nombre maximum de demandes IPC locales par seconde envoyées au périphérique principal.

Par défaut : 200 requêtes/seconde

maxLocalRequestsPerSecondPerThing

(Facultatif) Le nombre maximum de demandes IPC locales par seconde envoyées pour chaque objet IoT connecté.

Par défaut : 20 requests/second pour chaque objet

Note

Ces paramètres de limites de débit définissent le nombre maximal de demandes par seconde pour le service parallèle local. Le nombre maximum de demandes par seconde pour le service AWS IoT Device Shadow dépend de votre Région AWS. Pour plus d'informations, consultez les limites de l'[API AWS IoT Device Shadow Service](#) dans le Référence générale d'Amazon Web Services.

shadowDocumentSizeLimitBytes

(Facultatif) Taille maximale autorisée de chaque document d'état JSON pour les ombres locales.

Si vous augmentez cette valeur, vous devez également augmenter la limite de ressources pour le document d'état JSON pour les ombres des nuages. Pour plus d'informations, consultez les limites de l'[API AWS IoT Device Shadow Service](#) dans le Référence générale d'Amazon Web Services.

Par défaut : 8192 octets

Maximum : 30720 octets

Exemple Exemple : mise à jour de la fusion de configurations

L'exemple suivant montre un exemple de mise à jour de fusion de configuration avec tous les paramètres de configuration disponibles pour le composant Shadow Manager.

```
{
  "strategy":{
    "type":"periodic",
    "delay":300
  },
  "synchronize":{
    "shadowDocumentsMap":{
      "MyDevice1":{
        "classic":false,
        "namedShadows":[
          "MyShadowA",
          "MyShadowB"
        ]
      },
      "MyDevice2":{
        "classic":true,
        "namedShadows":[]
      }
    },
    "direction":"betweenDeviceAndCloud"
  },
  "rateLimits":{
    "maxOutboundSyncUpdatesPerSecond":100,
    "maxTotalLocalRequestsRate":200,
    "maxLocalRequestsPerSecondPerThing":20
  },
  "shadowDocumentSizeLimitBytes":8192
}
```

2.1.x

strategy

(Facultatif) Stratégie utilisée par ce composant pour synchroniser les ombres entre AWS IoT Core et le périphérique principal.

Cet objet contient les informations suivantes.

type

(Facultatif) Type de stratégie utilisé par ce composant pour synchroniser les ombres entre AWS IoT Core et le périphérique principal. Sélectionnez parmi les options suivantes :

- `realTime`— Synchronisez les ombres avec AWS IoT Core chaque mise à jour des ombres.
- `periodic`— Synchronisez les ombres avec AWS IoT Core un intervalle régulier que vous spécifiez avec le paramètre `delay` de configuration.

Par défaut : `realTime`

`delay`

(Facultatif) Intervalle en secondes avec lequel ce composant synchronise les ombres AWS IoT Core, lorsque vous spécifiez la stratégie de `periodic` synchronisation.

 Note

Ce paramètre est obligatoire si vous spécifiez la stratégie de `periodic` synchronisation.

`synchronize`

(Facultatif) Les paramètres de synchronisation qui déterminent la manière dont les ombres sont synchronisées avec AWS Cloud.

 Note

Vous devez créer une mise à jour de configuration avec cette propriété pour synchroniser les ombres avec AWS Cloud.

Cet objet contient les informations suivantes.

`coreThing`

(Facultatif) Les ombres de l'appareil principal à synchroniser. Cet objet contient les informations suivantes.

`classic`

(Facultatif) Par défaut, le gestionnaire de shadow synchronise l'état local du shadow classique de votre appareil principal avec le AWS Cloud. Si vous ne souhaitez pas synchroniser le shadow classique de l'appareil, réglez-le sur `false`.

Par défaut : `true`

`namedShadows`

(Facultatif) La liste des ombres du périphérique principal nommées à synchroniser. Vous devez indiquer le nom exact des ombres.

 Warning

Le AWS IoT Greengrass service utilise le `AWSThingsShadowV2Deployment` nom shadow pour gérer les déploiements qui ciblent des appareils principaux individuels. Cette ombre nommée est réservée à l'usage du AWS IoT Greengrass service. Ne mettez pas à jour ou ne supprimez pas cette ombre nommée.

`shadowDocumentsMap`

(Facultatif) Les ombres supplémentaires de l'appareil à synchroniser. L'utilisation de ce paramètre de configuration facilite la définition de documents fictifs. Nous vous recommandons d'utiliser ce paramètre à la place de l'`shadowDocument` objet.

 Note

Si vous spécifiez un `shadowDocumentsMap` objet, vous ne devez pas le spécifier. `shadowDocuments`

Chaque objet contient les informations suivantes :

thingName

La configuration d'ombre ***thingName*** pour cette configuration d'ombre.

`classic`

(Facultatif) Si vous ne souhaitez pas synchroniser le shadow classique de l'`thingName` appareil, réglez-le sur `false`.

`namedShadows`

La liste des ombres nommées que vous souhaitez synchroniser. Vous devez indiquer le nom exact des ombres.

shadowDocuments

(Facultatif) La liste des ombres supplémentaires de l'appareil à synchroniser. Nous vous recommandons d'utiliser le `shadowDocumentsMap` paramètre à la place.

Note

Si vous spécifiez un `shadowDocuments` objet, vous ne devez pas le spécifier.
`shadowDocumentsMap`

Chaque objet de cette liste contient les informations suivantes.

`thingName`

Nom de l'objet de l'appareil pour lequel les ombres doivent être synchronisées.

`classic`

(Facultatif) Si vous ne souhaitez pas synchroniser le shadow classique de l'`thingName` appareil, réglez-le sur `false`.

Par défaut : `true`

`namedShadows`

(Facultatif) La liste des ombres de périphériques nommées que vous souhaitez synchroniser. Vous devez indiquer le nom exact des ombres.

`rateLimits`

(Facultatif) Les paramètres qui déterminent les limites de débit pour les demandes de service parallèle.

Cet objet contient les informations suivantes.

`maxOutboundSyncUpdatesPerSecond`

(Facultatif) Nombre maximal de demandes de synchronisation par seconde transmises par l'appareil.

Par défaut : 100 requêtes/seconde

maxTotalLocalRequestsRate

(Facultatif) Le nombre maximum de demandes IPC locales par seconde envoyées au périphérique principal.

Par défaut : 200 requêtes/seconde

maxLocalRequestsPerSecondPerThing

(Facultatif) Le nombre maximum de demandes IPC locales par seconde envoyées pour chaque objet IoT connecté.

Par défaut : 20 requests/second pour chaque objet

Note

Ces paramètres de limites de débit définissent le nombre maximal de demandes par seconde pour le service parallèle local. Le nombre maximum de demandes par seconde pour le service AWS IoT Device Shadow dépend de votre Région AWS. Pour plus d'informations, consultez les limites de l'[API AWS IoT Device Shadow Service](#) dans le Référence générale d'Amazon Web Services.

shadowDocumentSizeLimitBytes

(Facultatif) Taille maximale autorisée de chaque document d'état JSON pour les ombres locales.

Si vous augmentez cette valeur, vous devez également augmenter la limite de ressources pour le document d'état JSON pour les ombres des nuages. Pour plus d'informations, consultez les limites de l'[API AWS IoT Device Shadow Service](#) dans le Référence générale d'Amazon Web Services.

Par défaut : 8192 octets

Maximum : 30720 octets

Exemple Exemple : mise à jour de la fusion de configurations

L'exemple suivant montre un exemple de mise à jour de fusion de configuration avec tous les paramètres de configuration disponibles pour le composant Shadow Manager.

```
{
  "strategy":{
    "type":"periodic",
    "delay":300
  },
  "synchronize":{
    "shadowDocumentsMap":{
      "MyDevice1":{
        "classic":false,
        "namedShadows":[
          "MyShadowA",
          "MyShadowB"
        ]
      },
      "MyDevice2":{
        "classic":true,
        "namedShadows":[]
      }
    },
    "direction":"betweenDeviceAndCloud"
  },
  "rateLimits":{
    "maxOutboundSyncUpdatesPerSecond":100,
    "maxTotalLocalRequestsRate":200,
    "maxLocalRequestsPerSecondPerThing":20
  },
  "shadowDocumentSizeLimitBytes":8192
}
```

2.0.x

synchronize

(Facultatif) Les paramètres de synchronisation qui déterminent la manière dont les ombres sont synchronisées avec AWS Cloud

Note

Vous devez créer une mise à jour de configuration avec cette propriété pour synchroniser les ombres avec AWS Cloud.

Cet objet contient les informations suivantes.

`coreThing`

(Facultatif) Les ombres de l'appareil principal à synchroniser. Cet objet contient les informations suivantes.

`classic`

(Facultatif) Par défaut, le gestionnaire de shadow synchronise l'état local du shadow classique de votre appareil principal avec le AWS Cloud. Si vous ne souhaitez pas synchroniser le shadow classique de l'appareil, réglez-le sur `false`.

Par défaut : `true`

`namedShadows`

(Facultatif) La liste des ombres du périphérique principal nommées à synchroniser. Vous devez indiquer le nom exact des ombres.



Warning

Le AWS IoT Greengrass service utilise le `AWSThingsShadow` nom shadow pour gérer les déploiements qui ciblent des appareils principaux individuels. Cette ombre nommée est réservée à l'usage du AWS IoT Greengrass service. Ne mettez pas à jour ou ne supprimez pas cette ombre nommée.

`shadowDocumentsMap`

(Facultatif) Les ombres supplémentaires de l'appareil à synchroniser. L'utilisation de ce paramètre de configuration facilite la définition de documents fictifs. Nous vous recommandons d'utiliser ce paramètre à la place de l'`shadowDocument` objet.



Note

Si vous spécifiez un `shadowDocumentsMap` objet, vous ne devez pas le spécifier.
`shadowDocuments`

Chaque objet contient les informations suivantes :

thingName

La configuration d'ombre *thingName* pour cette configuration d'ombre.

`classic`

(Facultatif) Si vous ne souhaitez pas synchroniser le shadow classique de l'*thingName* appareil, réglez-le sur `false`.

`namedShadows`

La liste des ombres nommées que vous souhaitez synchroniser. Vous devez indiquer le nom exact des ombres.

`shadowDocuments`

(Facultatif) La liste des ombres supplémentaires de l'appareil à synchroniser. Nous vous recommandons d'utiliser le `shadowDocumentsMap` paramètre à la place.

Note

Si vous spécifiez un `shadowDocuments` objet, vous ne devez pas le spécifier.
`shadowDocumentsMap`

Chaque objet de cette liste contient les informations suivantes.

`thingName`

Nom de l'objet de l'appareil pour lequel les ombres doivent être synchronisées.

`classic`

(Facultatif) Si vous ne souhaitez pas synchroniser le shadow classique de l'*thingName* appareil, réglez-le sur `false`.

Par défaut : `true`

`namedShadows`

(Facultatif) La liste des ombres de périphériques nommées que vous souhaitez synchroniser. Vous devez indiquer le nom exact des ombres.

`rateLimits`

(Facultatif) Les paramètres qui déterminent les limites de débit pour les demandes de service parallèle.

Cet objet contient les informations suivantes.

`maxOutboundSyncUpdatesPerSecond`

(Facultatif) Nombre maximal de demandes de synchronisation par seconde transmises par l'appareil.

Par défaut : 100 requêtes/seconde

`maxTotalLocalRequestsRate`

(Facultatif) Le nombre maximum de demandes IPC locales par seconde envoyées au périphérique principal.

Par défaut : 200 requêtes/seconde

`maxLocalRequestsPerSecondPerThing`

(Facultatif) Le nombre maximum de demandes IPC locales par seconde envoyées pour chaque objet IoT connecté.

Par défaut : 20 requests/second pour chaque objet

 Note

Ces paramètres de limites de débit définissent le nombre maximal de demandes par seconde pour le service parallèle local. Le nombre maximum de demandes par seconde pour le service AWS IoT Device Shadow dépend de votre Région AWS. Pour plus d'informations, consultez les limites de l'[API AWS IoT Device Shadow Service](#) dans le Référence générale d'Amazon Web Services.

`shadowDocumentSizeLimitBytes`

(Facultatif) Taille maximale autorisée de chaque document d'état JSON pour les ombres locales.

Si vous augmentez cette valeur, vous devez également augmenter la limite de ressources pour le document d'état JSON pour les ombres des nuages. Pour plus d'informations, consultez les limites de l'[API AWS IoT Device Shadow Service](#) dans le Référence générale d'Amazon Web Services.

Par défaut : 8192 octets

Maximum : 30720 octets

Exemple Exemple : mise à jour de la fusion de configurations

L'exemple suivant montre un exemple de mise à jour de fusion de configuration avec tous les paramètres de configuration disponibles pour le composant Shadow Manager.

```
{
  "synchronize": {
    "coreThing": {
      "classic": true,
      "namedShadows": [
        "MyCoreShadowA",
        "MyCoreShadowB"
      ]
    },
    "shadowDocuments": [
      {
        "thingName": "MyDevice1",
        "classic": false,
        "namedShadows": [
          "MyShadowA",
          "MyShadowB"
        ]
      },
      {
        "thingName": "MyDevice2",
        "classic": true,
        "namedShadows": []
      }
    ]
  },
  "rateLimits": {
    "maxOutboundSyncUpdatesPerSecond": 100,
    "maxTotalLocalRequestsRate": 200,
    "maxLocalRequestsPerSecondPerThing": 20
  },
  "shadowDocumentSizeLimitBytes": 8192
}
```

Fichier journal local

Ce composant utilise le même fichier journal que le composant [Greengrass nucleus](#).

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

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

Pour consulter les journaux de ce composant

- Exécutez la commande suivante sur le périphérique principal pour afficher le fichier journal de ce composant en temps réel. Remplacez `/greengrass/v2` ou `C:\greengrass\v2` par le chemin d'accès au dossier AWS IoT Greengrass racine.

Linux

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

Windows (PowerShell)

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

Journal des modifications

Le tableau suivant décrit les modifications apportées à chaque version du composant.

Version	Modifications
2.3.12	Corrections de bogues et améliorations Résout un problème en raison duquel les déploiements étaient bloqués lorsque plus de 1 024 devices shadow étaient configurés.
2.3.11	Version mise à jour pour la version 2.15.0 de Greengrass Nucleus.

Version	Modifications
2.3.10	Version mise à jour pour la version 2.14.0 de Greengrass Nucleus.
2.3.9	Version mise à jour pour la version 2.13.0 de Greengrass Nucleus.
2.3.8	Corrections de bogues et améliorations • Résout un problème selon lequel le gestionnaire fantôme créait une situation de blocage lors de la connexion du client MQTT.
2.3.7	Corrections de bogues et améliorations • Résout un problème selon lequel le Shadow Manager enregistre régulièrement une <code>NullPointerException</code> erreur lors d'une synchronisation du Shadow Manager.
2.3.6	Corrections de bogues et améliorations • Résout un problème selon lequel les propriétés d'ombre supprimées par le biais de AWS Cloud mises à jour alors que l'appareil est hors ligne continuent d'exister dans l'ombre locale une fois la connectivité rétablie.
2.3.5	Version mise à jour pour la version 2.12.0 de Greengrass Nucleus.
2.3.4	Corrections de bogues et améliorations • Ajoute la prise en charge des documents à état fictif nuls et vides.
2.3.3	Version mise à jour pour la version 2.11.0 de Greengrass Nucleus.
2.3.2	Corrections de bogues et améliorations • Résout un problème selon lequel Shadow Manager passe à l'<code>BROKEN</code> état lorsque la base de données parallèle locale est corrompue. • Version mise à jour pour la version 2.10.0 de Greengrass Nucleus.

Version	Modifications
2.3.1	Corrections de bogues et améliorations • Résout un problème susceptible d'empêcher la synchronisation des mises à jour de Cloud Shadow. • Résout un problème selon lequel les modifications apportées à la configuration de synchronisation des ombres nommées ne s'appliquent qu'à une seule ombre nommée.
2.3.0	Corrections de bogues et améliorations • Résout un problème susceptible d'empêcher les ombres de se synchroniser lorsque la clé privée de l'appareil Greengrass est stockée dans un module de sécurité matériel.
2.2.4	Corrections de bogues et améliorations • Résout un problème en raison duquel la validation de la taille de l'ombre n'était pas cohérente avec celle du cloud lors de la mise à jour du document fantôme local. • Résout un problème selon lequel le gestionnaire fantôme arrête d'écouter les mises à jour de configuration si un déploiement effectue une RESET opération sur les nœuds de configuration.
2.2.3	Version mise à jour pour la version 2.9.0 de Greengrass Nucleus.
2.2.2	Version mise à jour pour la version 2.8.0 de Greengrass Nucleus.
2.2.1	Version mise à jour pour la version 2.7.0 de Greengrass Nucleus.

Version	Modifications
2.2.0	Nouvelles fonctionnalités - Ajoute la prise en charge du service parallèle local via l' <code>publish/s</code> <code>unsubscribe</code> interface locale. Vous pouvez désormais communiquer avec le courtier de <code>publish/subscribe</code> messages local sur des sujets MQTT fictifs afin d'obtenir, de mettre à jour et de supprimer des ombres sur le périphérique principal. Cette fonctionnalité vous permet de connecter des appareils clients au service parallèle local en utilisant le pont MQTT pour relayer des messages sur des sujets cachés entre les appareils clients et l'interface de publication/d'abonnement locale. Cette fonctionnalité nécessite la version 2.6.0 ou ultérieure du composant Greengrass nucleus. Pour connecter les appareils clients au service parallèle local, vous devez également utiliser la version 2.2.0 ou ultérieure du composant pont MQTT. - Ajoute l'<code>direction</code> option que vous pouvez configurer pour personnaliser la direction afin de synchroniser les ombres entre le service d'ombre local et le AWS Cloud. Vous pouvez configurer cette option pour réduire la bande passante et les connexions au AWS Cloud.
2.1.1	Corrections de bogues et améliorations - Résout un problème selon lequel la profondeur maximale des <code>reported sections desired</code> et du document d'état fantôme du périphérique JSON était de 4 niveaux au lieu de 5 niveaux. - Version mise à jour pour la version 2.6.0 de Greengrass Nucleus.
2.1.0	Nouvelles fonctionnalités Prend en charge les intervalles périodiques de synchronisation parallèle, afin que vous puissiez configurer le périphérique principal afin de réduire l'utilisation de la bande passante et les frais.
2.0.6	Cette version contient des corrections de bogues et des améliorations.
2.0.5	Version mise à jour pour la version 2.5.0 de Greengrass Nucleus.

Version	Modifications
2.0.4	Corrections de bogues et améliorations • Résout un problème en raison duquel Shadow Manager supprimait les versions nouvellement créées de toutes les ombres précédemment supprimées. • Met à jour l'opération <code>DeleteThingShadow</code> IPC pour incrémenter la version fantôme lorsqu'elle est appelée.
2.0.3	Version mise à jour pour la version 2.4.0 de Greengrass Nucleus.
2.0.2	Corrections de bogues et améliorations • Correction d'un problème en raison duquel le gestionnaire des ombres ne reconnaissait pas la <code>delta</code> propriété lors de la synchronisation des états des ombres depuis AWS IoT Core. • Correction d'un problème qui provoquait parfois une fusion incorrecte des demandes de synchronisation pour une ombre.
2.0.1	Version mise à jour pour la version 2.3.0 de Greengrass Nucleus.
2.0.0	Première version.

Amazon SNS

Le composant Amazon SNS (`aws.greengrass.SNS`) publie des messages sur une rubrique Amazon Simple Notification Service (Amazon SNS). Vous pouvez utiliser ce composant pour envoyer des événements depuis les appareils principaux de Greengrass vers des serveurs Web, des adresses e-mail et d'autres abonnés aux messages. Pour plus d'informations, consultez [Qu'est-ce qu'Amazon SNS ?](#) dans le guide du développeur Amazon Simple Notification Service.

Pour publier sur une rubrique Amazon SNS avec ce composant, publiez un message dans le sujet auquel ce composant est abonné. Par défaut, ce composant s'abonne à la rubrique de [publication/d'abonnement sns/message locale](#). Vous pouvez spécifier d'autres sujets, y compris les sujets AWS IoT Core MQTT, lorsque vous déployez ce composant.

Dans votre composant personnalisé, vous souhaitez peut-être implémenter une logique de filtrage ou de formatage pour traiter les messages provenant d'autres sources avant de les publier dans ce

composant. Cela vous permet de centraliser votre logique de traitement des messages sur un seul composant.

Note

Ce composant fournit des fonctionnalités similaires à celles du connecteur Amazon SNS dans AWS IoT Greengrass la version 1. Pour plus d'informations, consultez le [connecteur Amazon SNS](#) dans le guide du développeur AWS IoT Greengrass V1.

Rubriques

- [Versions](#)
- [Type](#)
- [Système d'exploitation](#)
- [Exigences](#)
- [Dépendances](#)
- [Configuration](#)
- [Données d'entrée](#)
- [Données de sortie](#)
- [Fichier journal local](#)
- [Licences](#)
- [Journal des modifications](#)

Versions

Les versions de ce composant sont les suivantes :

- 2.1.x
- 2.0.x

Type

Ce composant est un composant Lambda () `aws.greengrass.lambda`. [Le noyau Greengrass exécute la fonction Lambda de ce composant à l'aide du composant Lambda Launcher.](#)

Pour de plus amples informations, veuillez consulter [Types de composants](#).

Système d'exploitation

Ce composant ne peut être installé que sur les appareils principaux de Linux.

Exigences

Ce composant répond aux exigences suivantes :

- Votre appareil principal doit répondre aux exigences pour exécuter les fonctions Lambda. Si vous souhaitez que le périphérique principal exécute des fonctions Lambda conteneurisées, le périphérique doit répondre aux exigences requises. Pour de plus amples informations, veuillez consulter [Exigences relatives à la fonction Lambda](#).
- [Python](#) version 3.7 installé sur le périphérique principal et ajouté à la variable d'environnement PATH.
- Une rubrique Amazon SNS Pour plus d'informations, consultez [Création d'une rubrique Amazon SNS](#) dans le Guide du développeur Amazon Simple Notification Service.
- Le [rôle d'appareil Greengrass](#) doit autoriser l'`sns:PublishAction`, comme illustré dans l'exemple de politique IAM suivant.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "sns:Publish"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:sns:us-east-1:123456789012:topic-name"
      ]
    }
  ]
}
```

Vous pouvez remplacer dynamiquement le sujet par défaut dans la charge utile du message d'entrée pour ce composant. Si votre application utilise cette fonctionnalité, la politique IAM doit

inclure tous les sujets 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 recevoir les données de sortie de ce composant, vous devez fusionner la mise à jour de configuration suivante pour l'[ancien composant routeur d'abonnement](#) (`aws.greengrass.LegacySubscriptionRouter`) lorsque vous déployez ce composant. Cette configuration indique le sujet dans lequel ce composant publie les réponses.

Legacy subscription router v2.1.x

```
{
  "subscriptions": {
    "aws-greengrass-sns": {
      "id": "aws-greengrass-sns",
      "source": "component:aws.greengrass.SNS",
      "subject": "sns/message/status",
      "target": "cloud"
    }
  }
}
```

Legacy subscription router v2.0.x

```
{
  "subscriptions": {
    "aws-greengrass-sns": {
      "id": "aws-greengrass-sns",
      "source": "arn:aws:lambda:region:aws:function:aws-greengrass-sns:version",
      "subject": "sns/message/status",
      "target": "cloud"
    }
  }
}
```

- *region* Remplacez-le par Région AWS celui que vous utilisez.
- *version* Remplacez-le par la version de la fonction Lambda exécutée par ce composant. Pour trouver la version de la fonction Lambda, vous devez consulter la recette de la version de ce composant que vous souhaitez déployer. Ouvrez la page de détails de ce composant dans la [AWS IoT Greengrass console](#) et recherchez la paire clé-valeur de la fonction Lambda. Cette paire clé-valeur contient le nom et la version de la fonction Lambda.

⚠ Important

Vous devez mettre à jour la version de la fonction Lambda sur l'ancien routeur d'abonnement chaque fois que vous déployez ce composant. Cela garantit que vous utilisez la bonne version de la fonction Lambda pour la version du composant que vous déployez.

Pour de plus amples informations, veuillez consulter [Créer des déploiements](#).

- Le composant Amazon SNS est compatible pour s'exécuter dans un VPC. Pour déployer ce composant dans un VPC, les éléments suivants sont requis.
 - Le composant Amazon SNS doit disposer d'une connectivité `sns.region.amazonaws.com` dont le point de terminaison VPC est de `com.amazonaws.us-east-1.sns`

Points de terminaison et ports

Ce composant doit être capable d'effectuer des demandes sortantes vers les points de terminaison et les ports suivants, en plus des points de terminaison et des ports requis pour le fonctionnement de base. Pour de plus amples informations, veuillez consulter [Autoriser le trafic des appareils via un proxy ou un pare-feu](#).

Endpoint	Port	Obligatoire	Description
<code>sns.region.amazonaws.com</code>	443	Oui	Publiez des messages sur Amazon SNS.

Dépendances

Lorsque vous déployez un composant, il déploie AWS IoT Greengrass également des versions compatibles de ses dépendances. Cela signifie que vous devez satisfaire aux exigences relatives au composant et à toutes ses dépendances pour réussir le déploiement du composant. Cette

section répertorie les dépendances des [versions publiées](#) de ce composant et les contraintes de version sémantiques qui définissent les versions des composants pour chaque dépendance. Vous pouvez également consulter les dépendances de chaque version du composant dans la [AWS IoT Greengrass console](#). Sur la page de détails du composant, recherchez la liste des dépendances.

2.1.10

Le tableau suivant répertorie les dépendances pour la version 2.1.10 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,0,0 < 2,16,0$	Stricte
Lanceur Lambda	$\wedge 2,0,0$	Stricte
Environnements d'exécution (runtimes) Lambda	$\wedge 2,0,0$	Flexible
Service d'échange de jetons	$\wedge 2,0,0$	Stricte

2.1.9

Le tableau suivant répertorie les dépendances pour la version 2.1.9 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,0,0 < 2,15,0$	Stricte
Lanceur Lambda	$\wedge 2,0,0$	Stricte
Environnements d'exécution (runtimes) Lambda	$\wedge 2,0,0$	Flexible
Service d'échange de jetons	$\wedge 2,0,0$	Stricte

2.1.8

Le tableau suivant répertorie les dépendances pour la version 2.1.8 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,14.0	Stricte
Lanceur Lambda	^2,0.0	Stricte
Environnements d'exécution (runtimes) Lambda	^2,0.0	Flexible
Service d'échange de jetons	^2,0.0	Stricte

2.1.7

Le tableau suivant répertorie les dépendances pour la version 2.1.7 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0,0 <2,13,0	Stricte
Lanceur Lambda	^2,0.0	Stricte
Environnements d'exécution (runtimes) Lambda	^2,0.0	Flexible
Service d'échange de jetons	^2,0.0	Stricte

2.1.6

Le tableau suivant répertorie les dépendances pour la version 2.1.6 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,12.0	Stricte
Lanceur Lambda	^2,0.0	Stricte
Environnements d'exécution (runtimes) Lambda	^2,0.0	Flexible

Dépendance	Versions compatibles	Type de dépendance
Service d'échange de jetons	^2,0.0	Stricte

2.1.5

Le tableau suivant répertorie les dépendances pour la version 2.1.5 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,11.0	Stricte
Lanceur Lambda	^2,0.0	Stricte
Environnements d'exécution (runtimes) Lambda	^2,0.0	Flexible
Service d'échange de jetons	^2,0.0	Stricte

2.1.4

Le tableau suivant répertorie les dépendances pour la version 2.1.4 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0,0 <2,1,0	Stricte
Lanceur Lambda	^2,0.0	Stricte
Environnements d'exécution (runtimes) Lambda	^2,0.0	Flexible
Service d'échange de jetons	^2,0.0	Stricte

2.1.3

Le tableau suivant répertorie les dépendances pour la version 2.1.3 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,9.0	Stricte
Lanceur Lambda	^2,0.0	Stricte
Environnements d'exécution (runtimes) Lambda	^2,0.0	Flexible
Service d'échange de jetons	^2,0.0	Stricte

2.1.2

Le tableau suivant répertorie les dépendances pour la version 2.1.2 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,8.0	Stricte
Lanceur Lambda	^2,0.0	Stricte
Environnements d'exécution (runtimes) Lambda	^2,0.0	Flexible
Service d'échange de jetons	^2,0.0	Stricte

2.1.1

Le tableau suivant répertorie les dépendances pour la version 2.1.1 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,7.0	Stricte
Lanceur Lambda	^2,0.0	Stricte
Environnements d'exécution (runtimes) Lambda	^2,0.0	Flexible

Dépendance	Versions compatibles	Type de dépendance
Service d'échange de jetons	^2,0.0	Stricte

2.0.8 - 2.1.0

Le tableau suivant répertorie les dépendances pour les versions 2.0.8 et 2.1.0 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,6.0	Stricte
Lanceur Lambda	^2,0.0	Stricte
Environnements d'exécution (runtimes) Lambda	^2,0.0	Flexible
Service d'échange de jetons	^2,0.0	Stricte

2.0.7

Le tableau suivant répertorie les dépendances pour la version 2.0.7 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,5.0	Stricte
Lanceur Lambda	^2,0.0	Stricte
Environnements d'exécution (runtimes) Lambda	^2,0.0	Flexible
Service d'échange de jetons	^2,0.0	Stricte

2.0.6

Le tableau suivant répertorie les dépendances pour la version 2.0.6 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,4.0	Stricte
Lanceur Lambda	^2,0.0	Stricte
Environnements d'exécution (runtimes) Lambda	^2,0.0	Flexible
Service d'échange de jetons	^2,0.0	Stricte

2.0.5

Le tableau suivant répertorie les dépendances pour la version 2.0.5 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,3.0	Stricte
Lanceur Lambda	^2,0.0	Stricte
Environnements d'exécution (runtimes) Lambda	^2,0.0	Flexible
Service d'échange de jetons	^2,0.0	Stricte

2.0.4

Le tableau suivant répertorie les dépendances pour la version 2.0.4 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,2.0	Stricte
Lanceur Lambda	^2,0.0	Stricte
Environnements d'exécution (runtimes) Lambda	^2,0.0	Flexible

Dépendance	Versions compatibles	Type de dépendance
Service d'échange de jetons	^2,0.0	Stricte

2.0.3

Le tableau suivant répertorie les dépendances pour la version 2.0.3 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0,3 <2,10	Stricte
Lanceur Lambda	>=10,0	Stricte
Environnements d'exécution (runtimes) Lambda	>=10,0	Flexible
Service d'échange de jetons	>=10,0	Stricte

Pour plus d'informations sur les dépendances des composants, consultez la [référence de la recette des composants](#).

Configuration

Ce composant fournit les paramètres de configuration suivants que vous pouvez personnaliser lorsque vous déployez le composant.

Note

La configuration par défaut de ce composant inclut les paramètres de la fonction Lambda. Nous vous recommandons de modifier uniquement les paramètres suivants pour configurer ce composant sur vos appareils.

lambdaParams

Objet contenant les paramètres de la fonction Lambda de ce composant. Cet objet contient les informations suivantes :

EnvironmentVariables

Objet contenant les paramètres de la fonction Lambda. Cet objet contient les informations suivantes :

DEFAULT_SNS_ARN

L'ARN de la rubrique Amazon SNS par défaut dans laquelle ce composant publie des messages. Vous pouvez remplacer le sujet de destination par la `sns_topic_arn` propriété dans la charge utile du message d'entrée.

containerMode

(Facultatif) Mode de conteneurisation de ce composant. Sélectionnez l'une des options suivantes :

- `NoContainer`— Le composant ne s'exécute pas dans un environnement d'exécution isolé.
- `GreengrassContainer`— Le composant s'exécute dans un environnement d'exécution isolé à l'intérieur du AWS IoT Greengrass conteneur.

Valeur par défaut : `GreengrassContainer`

containerParams

(Facultatif) Objet contenant les paramètres du conteneur pour ce composant. Le composant utilise ces paramètres si vous le spécifiez `GreengrassContainer` pour `containerMode`.

Cet objet contient les informations suivantes :

memorySize

(Facultatif) Quantité de mémoire (en kilo-octets) à allouer au composant.

La valeur par défaut est de 512 Mo (525 312 Ko).

pubsubTopics

(Facultatif) Objet contenant les rubriques auxquelles le composant s'abonne pour recevoir des messages. Vous pouvez spécifier chaque sujet et indiquer si le composant est abonné à des sujets MQTT provenant de sujets locaux AWS IoT Core ou locaux publish/subscribe .

Cet objet contient les informations suivantes :

0— Il s'agit d'un index de tableau sous forme de chaîne.

Un objet qui contient les informations suivantes :

type

(Facultatif) Type de publish/subscribe messagerie utilisé par ce composant pour s'abonner aux messages. Sélectionnez l'une des options suivantes :

- **PUB_SUB** – Abonnez-vous aux messages locaux de publication/abonnement. Si vous choisissez cette option, le sujet ne peut pas contenir de caractères génériques MQTT. Pour plus d'informations sur la façon d'envoyer des messages à partir d'un composant personnalisé lorsque vous spécifiez cette option, consultez [Publier/souscrire des messages locaux](#).
- **IOT_CORE**— Abonnez-vous aux messages AWS IoT Core MQTT. Si vous choisissez cette option, le sujet peut contenir des caractères génériques MQTT. Pour plus d'informations sur la façon d'envoyer des messages à partir de composants personnalisés lorsque vous spécifiez cette option, consultez [Publier/souscrire AWS IoT Core des messages MQTT](#).

Valeur par défaut : PUB_SUB

topic

(Facultatif) Rubrique à laquelle le composant s'abonne pour recevoir des messages. Si vous spécifiez `IotCore` pour `type`, vous pouvez utiliser des caractères génériques MQTT (+et#) dans cette rubrique.

Exemple Exemple : mise à jour de fusion de configuration (mode conteneur)

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "DEFAULT_SNS_ARN": "arn:aws:sns:us-west-2:123456789012:mytopic"
    }
  },
  "containerMode": "GreengrassContainer"
}
```

Exemple Exemple : mise à jour de fusion de configuration (pas de mode conteneur)

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "DEFAULT_SNS_ARN": "arn:aws:sns:us-west-2:123456789012:mytopic"
    }
  }
}
```

```
}  
},  
"containerMode": "NoContainer"  
}
```

Données d'entrée

Ce composant accepte les messages relatifs au sujet suivant et publie le message tel quel sur le sujet Amazon SNS cible. Par défaut, ce composant s'abonne aux publish/subscribe messages locaux. Pour plus d'informations sur la façon de publier des messages vers ce composant à partir de vos composants personnalisés, consultez [Publier/souscrire des messages locaux](#).

Rubrique par défaut (publication/abonnement local) : sns/message

Le message accepte les propriétés suivantes. Les messages d'entrée doivent être au format JSON.

request

Informations sur le message à envoyer à la rubrique Amazon SNS.

Type : object qui contient les informations suivantes :

message

Le contenu du message sous forme de chaîne.

Pour envoyer un objet JSON, sérialisez-le sous forme de chaîne et spécifiez json la message_structure propriété.

Type : string

subject

(Facultatif) Objet du message.

Type : string

Le sujet peut être du texte ASCII comportant jusqu'à 100 caractères. Il doit commencer par une lettre, un chiffre ou un signe de ponctuation. Il ne peut pas inclure de sauts de ligne ni de caractères de contrôle.

sns_topic_arn

(Facultatif) L'ARN de la rubrique Amazon SNS dans laquelle ce composant publie le message. Spécifiez cette propriété pour remplacer la rubrique Amazon SNS par défaut.

Type : string

message_structure

(Facultatif) Structure du message. Spécifiez json l'envoi d'un message JSON que vous sérialiserez sous forme de chaîne dans la content propriété.

Type : string

Valeurs valides : json

id

ID arbitraire de la demande. Utilisez cette propriété pour associer une demande d'entrée à une réponse de sortie. Lorsque vous spécifiez cette propriété, le composant définit la id propriété de l'objet de réponse sur cette valeur.

Type : string

Note

La taille du message peut être maximale de 256 Ko.

Exemple Exemple d'entrée : message de type chaîne

```
{
  "request": {
    "subject": "Message subject",
    "message": "Message data",
    "sns_topic_arn": "arn:aws:sns:region:account-id:topic2-name"
  },
  "id": "request123"
}
```

Exemple Exemple d'entrée : message JSON

```
{
  "request": {
    "subject": "Message subject",
    "message": "{ \"default\": \"Message data\" }",
    "message_structure": "json"
  }
}
```



```
},  
  "id": "request123"  
}
```

Données de sortie

Ce composant publie par défaut les réponses sous forme de données de sortie sur le sujet MQTT suivant. Vous devez spécifier cette rubrique `subject` dans la configuration de l'[ancien composant routeur d'abonnement](#). Pour plus d'informations sur la façon de s'abonner à des messages sur ce sujet dans vos composants personnalisés, consultez [Publier/souscrire AWS IoT Core des messages MQTT](#).

Rubrique par défaut (AWS IoT Core MQTT) : `sns/message/status`

Exemple Exemple de sortie : réussite

```
{  
  "response": {  
    "sns_message_id": "f80a81bc-f44c-56f2-a0f0-d5af6a727c8a",  
    "status": "success"  
  },  
  "id": "request123"  
}
```

Exemple Exemple de sortie : échec

```
{  
  "response" : {  
    "error": "InvalidInputException",  
    "error_message": "SNS Topic Arn is invalid",  
    "status": "fail"  
  },  
  "id": "request123"  
}
```

Fichier journal local

Ce composant utilise le fichier journal suivant.

```
/greengrass/v2/logs/aws.greengrass.SNS.log
```

Pour consulter les journaux de ce composant

- Exécutez la commande suivante sur le périphérique principal pour afficher le fichier journal de ce composant en temps réel. Remplacez `/greengrass/v2` par le chemin d'accès au dossier AWS IoT Greengrass racine.

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

Licences

Ce composant inclut les logiciels/licences tiers suivants :

- [AWS SDK pour 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 composant 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 composant.

Version	Modifications
2.1.10	Version mise à jour pour la version 2.15.0 de Greengrass Nucleus.
2.1.9	Version mise à jour pour la version 2.14.0 de Greengrass Nucleus.
2.1.8	Version mise à jour pour la version 2.13.0 de Greengrass Nucleus.
2.1.7	Version mise à jour pour la version 2.12.0 de Greengrass Nucleus.

Version	Modifications
2.1.6	Version mise à jour pour la version 2.11.0 de Greengrass Nucleus.
2.1.5	Version mise à jour pour la version 2.10.0 de Greengrass Nucleus.
2.1.4	Version mise à jour pour la version 2.9.0 de Greengrass Nucleus.
2.1.3	Version mise à jour pour la version 2.8.0 de Greengrass Nucleus.
2.1.2	Version mise à jour pour la version 2.7.0 de Greengrass Nucleus.
2.1.1	Version mise à jour pour la version 2.6.0 de Greengrass Nucleus.
2.1.0	Nouvelles fonctionnalités • Ajoute la prise en charge des configurations de proxy réseau HTTPS. Pour plus d'informations, consultez Connexion au port 443 ou via un proxy réseau et Permettre au périphérique principal de faire confiance à un proxy HTTPS.
2.0.8	Version mise à jour pour la version 2.5.0 de Greengrass Nucleus.
2.0.7	Version mise à jour pour la version 2.4.0 de Greengrass Nucleus.
2.0.6	Version mise à jour pour la version 2.3.0 de Greengrass Nucleus.
2.0.5	Version mise à jour pour la version 2.2.0 de Greengrass Nucleus.
2.0.4	Version mise à jour pour la version 2.1.0 de Greengrass Nucleus.
2.0.3	Première version.

Gestionnaire de flux

Le composant du gestionnaire de flux (`aws.greengrass.StreamManager`) vous permet de traiter les flux de données à transférer vers les appareils principaux AWS Cloud de Greengrass.

Pour plus d'informations sur la configuration et l'utilisation du gestionnaire de flux dans les composants personnalisés, consultez [Gérez les flux de données sur les appareils principaux de Greengrass](#).

Rubriques

- [Versions](#)
- [Type](#)
- [Système d'exploitation](#)
- [Prérequis](#)
- [Dépendances](#)
- [Configuration](#)
- [Fichier journal local](#)
- [Journal des modifications](#)

Versions

Les versions de ce composant sont les suivantes :

- 2.2.x
- 2,1x
- 2,0.x

Note

Si vous utilisez le gestionnaire de flux pour exporter des données vers le cloud, vous ne pouvez pas mettre à niveau la version 2.0.7 du composant du gestionnaire de flux vers une version comprise entre v2.0.8 et v2.0.11. Si vous déployez le gestionnaire de flux pour la première fois, nous vous recommandons vivement de déployer la dernière version du composant du gestionnaire de flux.

Type

Ce composant est un composant générique (`aws.greengrass.generic`). Le [noyau Greengrass](#) exécute les scripts de cycle de vie du composant.

Pour de plus amples informations, veuillez consulter [Types de composants](#).

Système d'exploitation

Ce composant peut être installé sur les appareils principaux qui exécutent les systèmes d'exploitation suivants :

- Linux
- Windows

Prérequis

Ce composant répond aux exigences suivantes :

- Le [rôle d'échange de jetons](#) doit autoriser l'accès aux AWS Cloud destinations que vous utilisez avec le gestionnaire de flux. Pour en savoir plus, consultez :
 - [the section called “AWS IoT Analytics chaînes”](#)
 - [the section called “Flux de données Amazon Kinesis”](#)
 - [the section called “AWS IoT SiteWise propriétés des actifs”](#)
 - [the section called “Objets Amazon S3”](#)
- Le composant du gestionnaire de flux peut être exécuté dans un VPC. Pour déployer ce composant dans un VPC, les éléments suivants sont requis.
 - Le composant du gestionnaire de flux doit être connecté au AWS service sur lequel vous publiez des données.
 - Amazon S3 : `com.amazonaws.region.s3`
 - Amazon Kinesis Data Streams : `com.amazonaws.region.kinesis-streams`
 - AWS IoT SiteWise: `com.amazonaws.region.iotsitewise.data`
 - Si vous publiez des données sur Amazon S3 dans la us-east-1 région, ce composant essaiera d'utiliser le point de terminaison global S3 par défaut ; toutefois, ce point de terminaison n'est pas disponible via le point de terminaison de l'interface Amazon S3 VPC. Pour plus d'informations, consultez [Restrictions et limites de AWS PrivateLink pour Amazon S3](#). Pour résoudre ce problème, vous pouvez choisir l'une des options suivantes.
 - Configurez le composant du gestionnaire de flux pour utiliser le point de terminaison S3 régional de la us-east-1 région, en le configurant - `Daws.s3UseUsEast1RegionalEndpoint=regional` dans `JVM_ARGS`.
 - Créez un point de terminaison VPC Amazon S3 Gateway au lieu d'un point de terminaison VPC d'interface Amazon S3. Les points de terminaison de la passerelle S3 prennent en

charge l'accès au point de terminaison global S3. Pour plus d'informations, consultez la section [Créer un point de terminaison de passerelle](#).

Points de terminaison et ports

Ce composant doit être capable d'effectuer des demandes sortantes vers les points de terminaison et les ports suivants, en plus des points de terminaison et des ports requis pour le fonctionnement de base. Pour de plus amples informations, veuillez consulter [Autoriser le trafic des appareils via un proxy ou un pare-feu](#).

Endpoint	Port	Obligatoire	Description
iotanalytics. <i>region</i> .amazonaws.com	443	Non	Obligatoire si vous publiez des données sur AWS IoT Analytics.
kinesis. <i>region</i> .amazonaws.com	443	Non	Obligatoire si vous publiez des données sur Firehose.
data.iots itewise. <i>region</i> .amazonaws.com	443	Non	Obligatoire si vous publiez des données sur AWS IoT SiteWise.
*.s3.amazonaws.com	443	Non	Obligatoire si vous publiez des

Endpoint	Port	Obligatoire	Description
			données dans des compartiments S3.
			Vous pouvez le * remplacer par le nom de chaque compartiment dans lequel vous publiez des données.

Dépendances

Lorsque vous déployez un composant, il déploie AWS IoT Greengrass également des versions compatibles de ses dépendances. Cela signifie que vous devez satisfaire aux exigences relatives au composant et à toutes ses dépendances pour réussir le déploiement du composant. Cette section répertorie les dépendances des [versions publiées](#) de ce composant et les contraintes de version sémantiques qui définissent les versions des composants pour chaque dépendance. Vous pouvez également consulter les dépendances de chaque version du composant dans la [AWS IoT Greengrass console](#). Sur la page de détails du composant, recherchez la liste des dépendances.

2.1.3 – 2.2.1

Le tableau suivant répertorie les dépendances pour les versions 2.1.3 à 2.2.1 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <3,0.0	Flexible
Service d'échange de jetons	>=0,0.0	Stricte

2.1.11 – 2.1.12

Le tableau suivant répertorie les dépendances pour les versions 2.1.11 à 2.1.10 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,0,0$ $< 2,13,0$	Flexible
Service d'échange de jetons	$\geq 0,0,0$	Stricte

2.1.9 – 2.1.10

Le tableau suivant répertorie les dépendances pour les versions 2.1.9 à 2.1.10 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,0,0$ $< 2,12,0$	Flexible
Service d'échange de jetons	$\geq 0,0,0$	Stricte

2.1.5 – 2.1.8

Le tableau suivant répertorie les dépendances pour les versions 2.1.5 à 2.1.8 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,0,0$ $< 2,11,0$	Flexible
Service d'échange de jetons	$\geq 0,0,0$	Stricte

2.1.2 – 2.1.4

Le tableau suivant répertorie les dépendances pour les versions 2.1.2 à 2.1.4 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	$\geq 2,0,0$ $< 2,1,0$	Flexible

Dépendance	Versions compatibles	Type de dépendance
Service d'échange de jetons	>=0,0.0	Stricte

2.1.1

Le tableau suivant répertorie les dépendances pour la version 2.1.1 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,9.0	Flexible
Service d'échange de jetons	>=0,0.0	Stricte

2.1.0

Le tableau suivant répertorie les dépendances pour la version 2.1.0 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,8.0	Flexible
Service d'échange de jetons	>=0,0.0	Stricte

2.0.15

Le tableau suivant répertorie les dépendances pour la version 2.0.15 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,7.0	Flexible
Service d'échange de jetons	>=0,0.0	Stricte

2.0.13 and 2.0.14

Le tableau suivant répertorie les dépendances pour les versions 2.0.13 et 2.0.14 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,6.0	Flexible
Service d'échange de jetons	>=0,0.0	Stricte

2.0.11 and 2.0.12

Le tableau suivant répertorie les dépendances pour les versions 2.0.11 et 2.0.12 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,5.0	Flexible
Service d'échange de jetons	>=0,0.0	Stricte

2.0.10

Le tableau suivant répertorie les dépendances pour la version 2.0.10 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,4.0	Flexible
Service d'échange de jetons	>=0,0.0	Stricte

2.0.9

Le tableau suivant répertorie les dépendances pour la version 2.0.9 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,3.0	Flexible
Service d'échange de jetons	>=0,0.0	Stricte

2.0.8

Le tableau suivant répertorie les dépendances pour la version 2.0.8 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0.0 <2,2.0	Flexible
Service d'échange de jetons	>=0,0.0	Stricte

2.0.7

Le tableau suivant répertorie les dépendances pour la version 2.0.7 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,0,3 <2,10	Flexible
Service d'échange de jetons	>=0,0.0	Stricte

Pour plus d'informations sur les dépendances des composants, consultez la [référence de la recette des composants](#).

Configuration

Ce composant fournit les paramètres de configuration suivants que vous pouvez personnaliser lorsque vous déployez le composant.

STREAM_MANAGER_STORE_ROOT_DIR

(Facultatif) Le chemin absolu du répertoire local utilisé pour stocker les flux. Cette valeur doit commencer par une barre oblique (par exemple, /data).

Vous devez spécifier un dossier existant, et l'[utilisateur du système qui exécute le composant du gestionnaire de flux](#) doit être autorisé à lire et à écrire dans ce dossier. Par exemple, vous pouvez exécuter les commandes suivantes pour créer et configurer un dossier `/var/greengrass/streams`, que vous spécifiez comme dossier racine du gestionnaire de flux. Ces commandes permettent à l'utilisateur du système par défaut de lire et d'écrire dans ce dossier. `ggc_user`

```
sudo mkdir /var/greengrass/streams
sudo chown ggc_user /var/greengrass/streams
sudo chmod 700 /var/greengrass/streams
```

Par défaut : `/greengrass/v2/work/aws.greengrass.StreamManager`

`STREAM_MANAGER_SERVER_PORT`

(Facultatif) Le numéro de port local à utiliser pour communiquer avec le gestionnaire de flux.

Vous pouvez spécifier `0` d'utiliser un port disponible de manière aléatoire.

Par défaut : `8088`

`STREAM_MANAGER_AUTHENTICATE_CLIENT`

(Facultatif) Vous pouvez obliger les clients à s'authentifier avant de pouvoir interagir avec le gestionnaire de flux. Le SDK Stream Manager contrôle l'interaction entre les clients et le gestionnaire de flux. Ce paramètre détermine quels clients peuvent appeler le SDK Stream Manager pour travailler avec des flux. Pour plus d'informations, consultez la section [Authentification du client du gestionnaire de flux](#).

Si vous le spécifiez `true`, le SDK Stream Manager n'autorise que les composants Greengrass en tant que clients.

Si vous le spécifiez `false`, le SDK Stream Manager permet à tous les processus du périphérique principal d'être des clients.

Par défaut : `true`

`STREAM_MANAGER_EXPORTER_MAX_BANDWIDTH`

(Facultatif) Bande passante maximale moyenne (en kilobits par seconde) que le gestionnaire de flux peut utiliser pour exporter des données.

Par défaut : pas de limite

STREAM_MANAGER_EXPORTER_THREAD_POOL_SIZE

(Facultatif) Le nombre maximum de threads actifs que le gestionnaire de flux peut utiliser pour exporter des données.

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. Essayez donc de limiter le nombre de flux d'exportation sur les appareils limités.

Par défaut : 5 fils

STREAM_MANAGER_EXPORTER_S3_DESTINATION_MULTIPART_UPLOAD_MIN_PART_SIZE_BYTES

(Facultatif) Taille minimale (en octets) d'une partie dans un téléchargement 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 en plusieurs parties.

Note

Le gestionnaire de flux utilise la `sizeThresholdForMultipartUploadBytes` propriété `streams` pour déterminer s'il convient d'exporter vers Amazon S3 sous forme de téléchargement en une ou plusieurs parties. AWS IoT Greengrass les composants peuvent définir ce seuil lorsqu'ils créent un flux exporté vers Amazon S3.

Par défaut : 5242880 (5 Mo). Il s'agit également de la valeur minimale.

LOG_LEVEL

(Facultatif) Le niveau de journalisation du composant. Choisissez parmi les niveaux de journalisation suivants, listés ici par ordre de niveau :

- TRACE
- DEBUG
- INFO

- WARN
- ERROR

Par défaut : INFO

JVM_ARGS

(Facultatif) Les arguments personnalisés de la machine virtuelle Java à transmettre au gestionnaire de flux au démarrage. Séparez les arguments multiples 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.

startupTimeoutSeconds

(Facultatif) Durée maximale en secondes pendant laquelle le composant démarre. L'état du composant passe à `ERRORED` s'il dépasse ce délai d'attente.

Par défaut : 120

Exemple Exemple : mise à jour de la fusion de configurations

L'exemple de configuration suivant indique d'utiliser un port autre que le port par défaut.

```
{  
  "STREAM_MANAGER_SERVER_PORT": "18088"  
}
```

Fichier journal local

Ce composant utilise le fichier journal suivant.

Linux

```
/greengrass/v2/logs/aws.greengrass.StreamManager.log
```

Windows

```
C:\greengrass\v2\logs\aws.greengrass.StreamManager.log
```

Pour consulter les journaux de ce composant

- Exécutez la commande suivante sur le périphérique principal pour afficher le fichier journal de ce composant en temps réel. Remplacez `/greengrass/v2` ou `C:\greengrass\v2` par le chemin d'accès au dossier AWS IoT Greengrass racine.

Linux

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

Windows (PowerShell)

```
Get-Content C:\greengrass\v2\logs\aws.greengrass.StreamManager.log -Tail 10 - Wait
```

Journal des modifications

Le tableau suivant décrit les modifications apportées à chaque version du composant.

Version	Modifications
2.2.1	Corrections de bugs et améliorations - Résout un problème en raison duquel le gestionnaire de flux ne parvient pas à exporter les messages vers les destinations Kinesis Data Streams. - Améliorations supplémentaires apportées aux performances des exportations du gestionnaire de flux vers les destinations Kinesis Data Streams.
2.2.0	Nouvelles fonctionnalités - Ajoute une nouvelle clé de configuration pour l'expiration du délai de démarrage. La valeur par défaut est de 120 secondes. - Ajoute des supports de recettes pour Greengrass Nucleus Lite.
2.1.13	Corrections de bugs et améliorations Supporte les points de terminaison FIPS pour AWS IoT SiteWise

Version	Modifications
2.1.12	Corrections de bugs et améliorations Met à jour l'ordre dans lequel les informations d'identification sont utilisées afin que les informations d'identification Greengrass soient préférées pour les demandes de AWS service.
2.1.11	Version mise à jour pour la version 2.12.0 de Greengrass Nucleus.
2.1.10	Corrections de bugs et améliorations Résout un problème selon lequel la configuration du proxy HTTPS ne fait pas confiance à la chaîne de certificats de l'autorité de certification (CA) Greengrass.
2.1.9	Version mise à jour pour la version 2.11.0 de Greengrass Nucleus.
2.1.8	Corrections de bugs et améliorations Résout un problème selon lequel le gestionnaire de flux réessaie indéfiniment SiteWise les exportations échouant avec. <code>InvalidRequestException</code>
2.1.7	Corrections de bugs et améliorations Résout un problème où le gestionnaire de flux ne lisait pas correctement la configuration du proxy.
2.1.6	Corrections de bugs et améliorations Résout un problème susceptible de provoquer un crash au démarrage sur certains ARMv8 processeurs, notamment le Jetson Nano.
2.1.5	Version mise à jour pour la version 2.10.0 de Greengrass Nucleus.

Version	Modifications
2.1.4	Corrections de bugs et améliorations • Résout un problème selon lequel les entrées pour le même actif immobilier avec le même horodatage au sein d'un même lot sont renvoyées par l' SiteWise API, ce qui oblige le gestionnaire <code>ConflictOperationException</code> de flux à réessayer en permanence. • Actualise le délai de connexion par défaut de 3 secondes à 1 minute.
2.1.3	Corrections de bugs et améliorations Résout un problème de démarrage sur le système d'exploitation Windows lors de l'exécution en tant qu'utilisateur du SYSTÈME.
2.1.2	Corrections de bugs et améliorations • Résout un problème sur les systèmes d'exploitation Windows qui utilisent une langue autre que l'anglais. • Version mise à jour pour la version 2.9.0 de Greengrass Nucleus.
2.1.1	Version mise à jour pour la version 2.8.0 de Greengrass Nucleus.
2.1.0	Nouvelles fonctionnalités • Met à jour ce composant pour envoyer automatiquement des métriques de télémétrie à Amazon. EventBridge Pour de plus amples informations, veuillez consulter Collectez les données de télémétrie relatives à l'état du système à partir des principaux appareils AWS IoT Greengrass. Cette fonctionnalité nécessite la version 2.7.0 ou ultérieure du composant Greengrass nucleus. • Version mise à jour pour la version 2.7.0 de Greengrass Nucleus.
2.0.15	Version mise à jour pour la version 2.6.0 de Greengrass Nucleus.
2.0.14	Cette version contient des corrections de bogues et des améliorations.
2.0.13	Version mise à jour pour la version 2.5.0 de Greengrass Nucleus.

Version	Modifications
2.0.12	Corrections de bugs et améliorations Résout un problème qui empêchait la mise à niveau du gestionnaire de flux v2.0.7 vers une version comprise entre v2.0.8 et v2.0.11. Si vous utilisez le gestionnaire de flux pour exporter des données vers le cloud, vous pouvez désormais passer à la version 2.0.12.
2.0.11	Version mise à jour pour la version 2.4.0 de Greengrass Nucleus.
2.0.10	Version mise à jour pour la version 2.3.0 de Greengrass Nucleus.
2.0.9	Version mise à jour pour la version 2.2.0 de Greengrass Nucleus.
2.0.8	Version mise à jour pour la version 2.1.0 de Greengrass Nucleus.
2.0.7	Première version.

Redirecteur de journal du système

Le redirecteur de journal système (`aws.greengrass.SystemLogForwarder`) télécharge les journaux système actifs directement sur Amazon à CloudWatch l'aide de l'API CloudWatch HTTPS.

Important

Ce composant ne transmettra que les journaux `systemd-journald` générés pendant l'exécution. [Pour plus d'informations sur les journaux `systemd-journald`, consultez `systemd-journald` et `journalctl`.](#)

Note

Ce composant nécessite des autorisations spécifiques pour créer et gérer des groupes de CloudWatch journaux et des flux.

Rubriques

- [Versions](#)
- [Type](#)
- [Système d'exploitation](#)
- [Exigences](#)
- [Points de terminaison et ports](#)
- [Dépendances](#)
- [Configuration](#)
- [Journal des modifications](#)

Versions

Les versions de ce composant sont les suivantes :

- 2.1.x
- 2.0.x

Type

Ce composant est un composant générique (`aws.greengrass.generic`). Le [noyau Greengrass](#) exécute les scripts de cycle de vie du composant.

Pour de plus amples informations, veuillez consulter [Types de composants](#).

Système d'exploitation

Ce composant doit être installé sur les systèmes Linux basés sur Systemd.

Exigences

Ce composant répond aux exigences suivantes :

Le composant nécessite un accès pour créer des groupes de journaux et de flux, CloudWatch ainsi que l'autorisation d'effectuer l'appel PutLogEvents HTTP. Vous devez au minimum ajouter les autorisations politiques suivantes à l'alias de rôle de votre appareil Greengrass :

JSON

```
{
```

```

    "Version": "2012-10-17",
    "Statement": [
      {
        "Effect": "Allow",
        "Action": ["logs:CreateLogGroup"],
        "Resource": "arn:aws:logs:us-east-1:111122223333:log-group:greengrass/systemLogs:"
      },
      {
        "Effect": "Allow",
        "Action": ["logs:CreateLogStream", "logs:PutLogEvents"],
        "Resource": "arn:aws:logs:us-east-1:111122223333:log-group:greengrass/systemLogs:log-stream:${credentials-iot:ThingName}"
      }
    ]
  }
}

```

Note

Pour plus d'informations, consultez la page [Github](#) du System Log Forwarder.

Points de terminaison et ports

Ce composant doit être capable d'effectuer des demandes sortantes vers les points de terminaison et les ports suivants, en plus des points de terminaison et des ports requis pour le fonctionnement de base. Pour de plus amples informations, veuillez consulter [Autoriser le trafic des appareils via un proxy ou un pare-feu](#).

Endpoint	Port	Obligatoire	Description
logs. <i>region</i> .amazonaws.com	443	Non	Obligatoire si vous écrivez des journaux dans CloudWatch Logs.

Dépendances

Lorsque vous déployez un composant, il déploie AWS IoT Greengrass également des versions compatibles de ses dépendances. Cela signifie que vous devez satisfaire aux exigences relatives au composant et à toutes ses dépendances pour réussir le déploiement du composant. Cette section répertorie les dépendances des [versions publiées](#) de ce composant et les contraintes de version sémantiques qui définissent les versions des composants pour chaque dépendance. Vous pouvez également consulter les dépendances de chaque version du composant dans la [AWS IoT Greengrass console](#). Sur la page de détails du composant, recherchez la liste des dépendances.

2.1.x

Le tableau suivant répertorie les dépendances pour la version 2.1.x de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Service d'échange de jetons	$\geq 2,0,0$	Stricte
Greengrass Nucleus Lite	$\geq 2,3,0$	Flexible

2.0.x

Le tableau suivant répertorie les dépendances pour la version 2.0.x de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Service d'échange de jetons	$\geq 2,0,0$	Stricte

Configuration

Ce composant fournit les paramètres de configuration suivants que vous pouvez personnaliser lorsque vous déployez le composant.

2.0.x-2.1.x

`maxUploadIntervalSec`

Période maximale pendant laquelle le redirecteur des journaux du système tentera de télécharger les journaux. Étant donné que le redirecteur de journaux télécharge les journaux

lorsque la mémoire est pleine, il se peut qu'il soit chargé plus souvent que la cadence maximale configurée.

`maxRetriesCount`

Nombre de fois où le redirecteur de journal du système tente de retenter une erreur HTTP transitoire.

`bufferCapacity`

Taille de la mémoire tampon en anneau pour le stockage des journaux en mémoire.

`logGroup`

Le chemin du journal dans CloudWatch.

`logStream`

Le CloudWatch LogStream.

`filters`

Carte des configurations de filtre pour le périphérique principal.

`services`

Liste des filtres de nom de service que System Log Forwarder utilisera pour déterminer quels journaux seront téléchargés. Un journal ne sera chargé que si le service dont il provient correspond à au moins un des filtres de cette liste. Les filtres de cette liste peuvent être soit une chaîne à laquelle le nom du service doit correspondre entièrement, soit une chaîne se terminant par *, ce qui signifie que le préfixe doit correspondre.

Valeur par défaut : `[ggl.*]`

Important

Un journal ne sera chargé que si le service dont il provient correspond à au moins un des filtres de cette liste.

Note

L'utilisation de la valeur * inclura tous les services disponibles.

Exemple Exemple de configuration :

L'exemple ci-dessous filtrera les journaux en fonction de tous les services inclus dans Greengrass Nucleus Lite.

```
{
  "maxUploadIntervalSec": 300,
  "maxRetriesCount": 3,
  "bufferCapacity": 1048576,
  "logGroup": "greengrass/systemLogs",
  "logStream": "deviceName",
  "filters": {
    "services": ["gg1.*"]
  }
}
```

Journal des modifications

Le tableau suivant décrit les modifications apportées à chaque version du composant.

Version	Modifications
2.1.0	Corrections de bogues et améliorations • Met à jour la recette du composant pour qu'il supporte correctement le noyau de Greengrass. • Résultat de journalisation amélioré lorsqu'il n'y a aucun journal à télécharger. • Correction et amélioration de bogues généraux
2.0.1	Corrections de bogues et améliorations Met à jour la recette du composant pour prendre correctement en charge les systèmes aarch64 (arm64).
2.0.0	Première version.

Agent Systems Manager

Le composant AWS Systems Manager Agent (`aws.greengrass.SystemsManagerAgent`) installe l'agent Systems Manager afin que vous puissiez gérer les appareils principaux avec Systems Manager. Systems Manager est un AWS service que vous pouvez utiliser pour visualiser et contrôler votre infrastructure AWS, y compris les EC2 instances Amazon, les serveurs sur site et les machines virtuelles (VMs), ainsi que les appareils périphériques. Systems Manager vous permet de visualiser les données opérationnelles, d'automatiser les tâches opérationnelles et de garantir la sécurité et la conformité. Pour plus d'informations, voir [Qu'est-ce que c'est AWS Systems Manager ?](#) et [À propos de Systems Manager Agent](#) dans le guide de AWS Systems Manager l'utilisateur.

Les outils et fonctionnalités de Systems Manager sont appelés fonctionnalités. Les appareils principaux de Greengrass prennent en charge toutes les fonctionnalités de Systems Manager. Pour plus d'informations sur ces fonctionnalités et sur la façon d'utiliser Systems Manager pour gérer les appareils principaux, consultez la section [Fonctionnalités de Systems Manager](#) dans le Guide de AWS Systems Manager l'utilisateur.

Rubriques

- [Versions](#)
- [Type](#)
- [Système d'exploitation](#)
- [Prérequis](#)
- [Dépendances](#)
- [Configuration](#)
- [Fichier journal local](#)
- [Consultez aussi](#)
- [Journal des modifications](#)

Versions

Les versions de ce composant sont les suivantes :

- 1.3.x
- 1,2.x
- 1,1x

- 1,0 x

Type

Ce composant est un composant générique (`aws.greengrass.generic`). Le [noyau Greengrass](#) exécute les scripts de cycle de vie du composant.

Pour de plus amples informations, veuillez consulter [Types de composants](#).

Système d'exploitation

Ce composant ne peut être installé que sur les appareils principaux de Linux.

Prérequis

Ce composant répond aux exigences suivantes :

- Un appareil principal Greengrass qui fonctionne sur une plate-forme Linux 64 bits : Armv8 () AArch64 ou x86_64.
- Vous devez disposer d'un rôle de service Gestion des identités et des accès AWS (IAM) que Systems Manager peut assumer. Ce rôle doit inclure la politique SSMManged InstanceCore gérée par [Amazon](#) ou une politique personnalisée qui définit des autorisations équivalentes. Pour plus d'informations, voir [Créer un rôle de service IAM pour les appareils Edge](#) dans le Guide de l'AWS Systems Manager utilisateur.

Lorsque vous déployez ce composant, vous devez spécifier le nom de ce rôle pour le paramètre `SSMRegistrationRole` de configuration.

- Le [rôle d'appareil Greengrass](#) doit autoriser les actions `ssm:AddTagsToResource` et `ssm:RegisterManagedInstance`. Le rôle de périphérique doit également autoriser `iam:PassRole` pour le rôle de service IAM qui répond à l'exigence précédente. L'exemple de politique IAM suivant accorde ces autorisations.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "iam:PassRole"
      ],
      "Effect": "Allow",
```

```
    "Resource": [
      "arn:aws:iam::account-id:role/SSMServiceRole"
    ],
  },
  {
    "Action": [
      "ssm:AddTagsToResource",
      "ssm:RegisterManagedInstance"
    ],
    "Effect": "Allow",
    "Resource": "*"
  }
]
```

Endpoints et ports

Ce composant doit être capable d'effectuer des demandes sortantes vers les points de terminaison et les ports suivants, en plus des points de terminaison et des ports requis pour le fonctionnement de base. Pour de plus amples informations, veuillez consulter [Autoriser le trafic des appareils via un proxy ou un pare-feu](#).

Endpoint	Port	Obligatoire	Description
ec2messages. <i>region</i> .amazonaws.com	443	Oui	Communiquez avec le service Systems Manager dans le AWS Cloud.
ssm. <i>region</i> .amazonaws.com	443	Oui	Enregistrez le périphérique principal en tant que nœud

Endpoint	Port	Obligatoire	Description
			géré par Systems Manager.
ssmmessages. <i>region</i> .amazonaws.com	443	Oui	Communiquez avec Session Manager, une fonctionnalité de Systems Manager, dans le AWS Cloud.

Pour plus d'informations, voir [Référence : ec2messages, ssmmessages et autres appels d'API](#) dans le guide de l'utilisateur AWS Systems Manager

Dépendances

Lorsque vous déployez un composant, il déploie AWS IoT Greengrass également des versions compatibles de ses dépendances. Cela signifie que vous devez satisfaire aux exigences relatives au composant et à toutes ses dépendances pour réussir le déploiement du composant. Cette section répertorie les dépendances des [versions publiées](#) de ce composant et les contraintes de version sémantiques qui définissent les versions des composants pour chaque dépendance. Vous pouvez également consulter les dépendances de chaque version du composant dans la [AWS IoT Greengrass console](#). Sur la page de détails du composant, recherchez la liste des dépendances.

Le tableau suivant répertorie les dépendances pour les versions 1.0.0 à 1.3.0 de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Service d'échange de jetons	>=2,0.0 <3,0.0	Stricte

Pour plus d'informations sur les dépendances des composants, consultez la [référence de la recette des composants](#).

Configuration

Ce composant fournit les paramètres de configuration suivants que vous pouvez personnaliser lorsque vous déployez le composant.

SSMRegistrationRole

Le rôle de service IAM que Systems Manager peut assumer et qui inclut la politique `SSMManaged InstanceCore` gérée par [Amazon](#) ou une politique personnalisée qui définit des autorisations équivalentes. Pour plus d'informations, voir [Créer un rôle de service IAM pour les appareils Edge](#) dans le Guide de l'AWS Systems Manager utilisateur.

SSMOverrideExistingRegistration

(Facultatif) Si le périphérique principal exécute déjà l'agent Systems Manager enregistré avec une activation hybride, vous pouvez annuler l'enregistrement de l'agent Systems Manager existant sur le périphérique. Définissez cette option sur `true` pour enregistrer le périphérique principal en tant que nœud géré à l'aide de l'agent Systems Manager fourni par ce composant.

Note

Cette option s'applique uniquement aux appareils enregistrés avec une activation hybride. Si le périphérique principal s'exécute sur une EC2 instance Amazon avec l'agent Systems Manager installé et un rôle de profil d'instance configuré, l'ID de nœud géré existant de l'EC2 instance Amazon commence par `i-`. Lorsque vous installez le composant Systems Manager Agent, celui-ci enregistre un nouveau nœud géré dont l'ID commence par `mi-` au lieu de `i-`. Vous pouvez ensuite utiliser le nœud géré dont l'identifiant commence par `mi-` pour gérer le périphérique principal avec Systems Manager.

Par défaut : `false`

SSMResourceTags

(Facultatif) Les balises à ajouter au nœud géré par Systems Manager que ce composant crée pour le périphérique principal. Vous pouvez utiliser ces balises pour gérer des groupes d'appareils principaux avec Systems Manager. Par exemple, vous pouvez exécuter une commande sur tous les appareils dotés d'une étiquette que vous spécifiez.

Spécifiez une liste dans laquelle chaque balise est un objet avec a Key et Value a. Par exemple, la valeur suivante pour SSMResourceTags indique à ce composant de définir la **Owner** balise **richard-roe** sur le nœud géré du périphérique principal.

```
[
  {
    "Key": "Owner",
    "Value": "richard-roe"
  }
]
```

Ce composant ignore ces balises si le nœud géré existe déjà et SSMOverrideExistingRegistration existe false déjà.

Exemple Exemple : mise à jour de la fusion de configurations

L'exemple de configuration suivant indique d'utiliser un rôle de service nommé SSMServiceRole pour permettre au périphérique principal de s'enregistrer et de communiquer avec Systems Manager.

```
{
  "SSMRegistrationRole": "SSMServiceRole",
  "SSMOverrideExistingRegistration": false,
  "SSMResourceTags": [
    {
      "Key": "Owner",
      "Value": "richard-roe"
    },
    {
      "Key": "Team",
      "Value": "solar"
    }
  ]
}
```

Fichier journal local

Le logiciel Systems Manager Agent écrit les journaux dans un dossier situé en dehors du dossier racine de Greengrass. Pour plus d'informations, consultez la section [Visualisation des journaux de l'agent Systems Manager](#) dans le guide de AWS Systems Manager l'utilisateur.

Le composant Systems Manager Agent utilise des scripts shell pour installer, démarrer et arrêter l'agent Systems Manager. Vous trouverez le résultat de ces scripts dans le fichier journal suivant.

```
/greengrass/v2/logs/aws.greengrass.SystemsManagerAgent.log
```

Pour consulter les journaux de ce composant

- Exécutez la commande suivante sur le périphérique principal pour afficher le fichier journal de ce composant en temps réel. Remplacez `/greengrass/v2` par le chemin d'accès au dossier AWS IoT Greengrass racine.

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

Consultez aussi

- [Gérez les appareils principaux de Greengrass avec AWS Systems Manager](#)
- [Qu'est-ce que AWS Systems Manager ?](#) dans le Guide de l'utilisateur AWS Systems Manager
- [À propos de Systems Manager Agent](#) dans le guide de AWS Systems Manager l'utilisateur

Journal des modifications

Le tableau suivant décrit les modifications apportées à chaque version du composant.

Version	Modifications
1.3.0	Corrections de bugs et améliorations Met à jour ce composant pour obtenir la version 3.3.1142.0 de l'agent.
1.2.4	Corrections de bugs et améliorations Met à jour ce composant pour obtenir la version 3.2.2303.0 de l'agent.
1.2.3	Corrections de bugs et améliorations • Ajoute de nouvelles tentatives pour l'installation du composant Agent avec Snap on Greengrass.

Version	Modifications
	• Met à jour la configuration du composant Agent pour utiliser uniquement l'identité Onprem dans Greengrass. • Met à jour ce composant pour mettre à jour l'agent uniquement lorsque la version de l'agent installée ne correspond pas à la version du composant Greengrass SSM Agent.
1.1.0	Cette version contient des corrections de bugs et des améliorations.
1.0.0	Première version.

Service d'échange de jetons

Le composant de service d'échange de jetons (`aws.greengrass.TokenExchangeService`) fournit des AWS informations d'identification que vous pouvez utiliser pour interagir avec les AWS services de vos composants personnalisés.

Le service d'échange de jetons fonctionne comme un serveur local. Ce serveur local se connecte au fournisseur AWS IoT d'informations d'identification à l'aide de l'alias de AWS IoT rôle que vous configurez dans le composant central du [noyau de Greengrass](#). Le composant fournit deux variables d'environnement, `AWS_CONTAINER_CREDENTIALS_FULL_URI` et `AWS_CONTAINER_AUTHORIZATION_TOKEN`. `AWS_CONTAINER_CREDENTIALS_FULL_URI` définit l'URI de ce serveur local. Lorsqu'un composant crée un client AWS SDK, le client reconnaît cette variable d'environnement URI et utilise le jeton contenu dans le `AWS_CONTAINER_AUTHORIZATION_TOKEN` pour se connecter au service d'échange de jetons et récupérer les AWS informations d'identification. Cela permet aux appareils principaux de Greengrass d'appeler les opérations de AWS service. Pour plus d'informations sur l'utilisation de ce composant dans des composants personnalisés, consultez [Interagissez avec les AWS services](#).

Important

Support pour acquérir des AWS informations d'identification de cette manière a été ajouté AWS SDKs le 13 juillet 2016. Votre composant doit utiliser une version du AWS SDK créée à cette date ou après cette date. Pour plus d'informations, consultez la section [Utilisation d'un AWS SDK compatible](#) dans le manuel Amazon Elastic Container Service Developer Guide.

 Note

Les composants utilisant la chaîne d'informations d'identification AWS par défaut peuvent découvrir et utiliser d'autres sources d'informations d'identification, si elles existent. Au sein d'Amazon Amazon Elastic Container Service (Amazon ECS), les `AWS_CONTAINER_CREDENTIALS_RELATIVE_URI` informations d'identification du conteneur seront prioritaires `AWS_CONTAINER_CREDENTIALS_FULL_URI`, ce qui signifie que les informations d'identification du service d'échange de jetons peuvent ne pas fonctionner au sein d'Amazon Elastic Container Service (Amazon ECS). Pour en savoir plus, consultez les [fournisseurs d'identifiants standardisés](#)

Rubriques

- [Versions](#)
- [Type](#)
- [Système d'exploitation](#)
- [Dépendances](#)
- [Configuration](#)
- [Fichier journal local](#)
- [Journal des modifications](#)

Versions

Les versions de ce composant sont les suivantes :

- 2.0.x

Type

Ce composant est un composant générique (`aws.greengrass.generic`). Le [noyau Greengrass](#) exécute les scripts de cycle de vie du composant.

Pour de plus amples informations, veuillez consulter [Types de composants](#).

Système d'exploitation

Ce composant peut être installé sur les appareils principaux qui exécutent les systèmes d'exploitation suivants :

- Linux
- Windows

Dépendances

Ce composant n'a aucune dépendance.

Configuration

Ce composant fournit les paramètres de configuration suivants que vous pouvez personnaliser lorsque vous déployez le composant.

`port`

Port à utiliser pour les connexions au service d'échange de jetons. Le service d'échange de jetons redémarrera une fois la configuration du port modifiée.

`credentialRetryInSec`

Spécifie les intervalles entre les tentatives en secondes lorsque le service Token Exchange rencontre des erreurs de demande d'informations d'identification.

`clientError`

Intervalle entre les tentatives en secondes pour les erreurs du client (4xx codes d'état HTTP).

Par défaut: 120

Valeurs valides : de 10 à 42900

`serverError`

Intervalle entre les tentatives en secondes pour les erreurs du serveur (5xx codes d'état HTTP).

Par défaut: 60

Valeurs valides : de 10 à 42900

unknownError

Intervalle entre les tentatives en secondes pour les erreurs inconnues (erreurs de connexion et codes d'état HTTP en dehors des plages 4xx et 5xx).

Par défaut: 300

Valeurs valides : de 10 à 42900

Exemple Exemple : mise à jour de la fusion de configurations

```
{
  "port": 2020,
  "credentialRetryInSec": {
    "clientError": 30,
    "serverError": 45,
    "unknownError": 60
  }
}
```

Fichier journal local

Ce composant utilise le même fichier journal que le composant [Greengrass nucleus](#).

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

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

Pour consulter les journaux de ce composant

- Exécutez la commande suivante sur le périphérique principal pour afficher le fichier journal de ce composant en temps réel. Remplacez */greengrass/v2* ou *C:\greengrass\v2* par le chemin d'accès au dossier AWS IoT Greengrass racine.

Linux

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

Windows (PowerShell)

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

Journal des modifications

Le tableau suivant décrit les modifications apportées à chaque version du composant.

Version	Modifications
2.0.3	Première version.

Collecteur IoT SiteWise OPC UA

Le composant collecteur IoT SiteWise OPC UA (`aws.iot.SiteWiseEdgeCollectorOpcua`) permet aux AWS IoT SiteWise passerelles de collecter des données à partir de serveurs OPC UA locaux.

Avec ce composant, les AWS IoT SiteWise passerelles peuvent se connecter à plusieurs serveurs OPC UA. Pour plus d'informations sur les AWS IoT SiteWise passerelles, consultez la section [Utilisation AWS IoT SiteWise du périphérique dans le](#) Guide de l'AWS IoT SiteWise utilisateur.

Rubriques

- [Versions](#)
- [Type](#)
- [Système d'exploitation](#)
- [Prérequis](#)
- [Dépendances](#)
- [Configuration](#)
- [Données d'entrée](#)

- [Données de sortie](#)
- [Fichier journal local](#)
- [Licences](#)
- [Journal des modifications](#)
- [Consultez aussi](#)

Versions

Les versions de ce composant sont les suivantes :

- 3.1.x
- 3,0. x
- 2,6. x
- 2,5.x
- 2.4.x
- 2.3.x
- 2.2.x
- 2,1x
- 2,0.x

Type

Ce composant est un composant générique (`aws.greengrass.generic`). Le [noyau Greengrass](#) exécute les scripts de cycle de vie du composant.

Pour de plus amples informations, veuillez consulter [Types de composants](#).

Système d'exploitation

Ce composant peut être installé sur les appareils principaux qui exécutent les systèmes d'exploitation suivants :

- Linux
- Windows

Prérequis

Ce composant répond aux exigences suivantes :

- L'appareil principal de Greengrass doit fonctionner sur l'une des plateformes suivantes :
 - Système d'exploitation : Ubuntu 20.04 ou version ultérieure
architecture : x86_64 (AMD64) ou ARMv8 (Aarch64)
 - Système d'exploitation : Red Hat Enterprise Linux (RHEL) 8
architecture : x86_64 (AMD64) ou ARMv8 (Aarch64)
 - Système d'exploitation : Amazon Linux 2
architecture : x86_64 (AMD64) ou ARMv8 (Aarch64)
 - Système d'exploitation : Debian 11
architecture : x86_64 (AMD64) ou ARMv8 (Aarch64)
 - Système d'exploitation : Windows Server 2019 ou version ultérieure
architecture : x86_64 (AMD64)
- Le périphérique principal Greengrass doit autoriser la connectivité réseau sortante aux serveurs OPC UA.

Dépendances

Lorsque vous déployez un composant, il déploie AWS IoT Greengrass également des versions compatibles de ses dépendances. Cela signifie que vous devez satisfaire aux exigences relatives au composant et à toutes ses dépendances pour réussir le déploiement du composant. Cette section répertorie les dépendances des [versions publiées](#) de ce composant et les contraintes de version sémantiques qui définissent les versions des composants pour chaque dépendance. Vous pouvez également consulter les dépendances de chaque version du composant dans la [AWS IoT Greengrass console](#). Sur la page de détails du composant, recherchez la liste des dépendances.

Le tableau suivant répertorie les dépendances de toutes les versions de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,3,0 <3,0,0	Stricte

Dépendance	Versions compatibles	Type de dépendance
Gestionnaire de flux	>2,0,10<3,0,0	Stricte
Directeur des services secrets	>=2,0,8 <3,0,0	Stricte

Pour plus d'informations sur les dépendances des composants, consultez la [référence de la recette des composants](#).

Configuration

Ce composant ne possède aucun paramètre de configuration.

Vous pouvez utiliser la AWS IoT SiteWise console ou l'API pour configurer le composant collecteur IoT SiteWise OPC UA. Pour plus d'informations, voir [Étape 4 : Ajouter des sources de données \(facultatif\)](#) dans le Guide de AWS IoT SiteWise l'utilisateur.

Données d'entrée

Ce composant accepte uniquement les données dans les formats suivants, tous les autres seront ignorés et rejetés. Le tableau ci-dessous met en correspondance les types de données OPC UA avec leurs SiteWise équivalents.

SiteWise type de données	Type de données OPC UA	Description
STRING	String	Chaîne d'une longueur maximale de 1024 octets.
	Guid	
	XmlElement	
INTEGER	SByte	Un entier signé de 32 bits dont la plage est comprise -2,147,483,648 to 2,147,483,647 entre.
	Byte	
	Int16	
	UInt16	
	Int32	

SiteWise type de données	Type de données OPC UA	Description
DOUBLE	UInt32*	Nombre à virgule flottante avec plage de -10^{100} to 10^{100} valeurs et IEEE 754 double précision.
	Int64*	
	UInt32*	
	Int64*	
	Float	
	Double	
BOOLEAN	Boolean	true ou false.

* Pour les types de données OPC UA UInt32 et Int64, son type de SiteWise données sera INTEGER s'il SiteWise est capable de représenter sa valeur, sinon il le sera DOUBLE.

Données de sortie

Ce composant écrit BatchPutAssetPropertyValue des messages dans le gestionnaire de AWS IoT Greengrass flux. Pour plus d'informations, consultez [BatchPutAssetPropertyValue](#) dans la Référence d'API AWS IoT SiteWise .

Fichier journal local

Ce composant utilise le fichier journal suivant.

Linux

```
/greengrass/v2/logs/aws.iot.SiteWiseEdgeCollectorOpcua.log
```

Windows

```
C:\greengrass\v2\logs\aws.iot.SiteWiseEdgeCollectorOpcua.log
```

Pour consulter les journaux de ce composant

- Exécutez la commande suivante sur le périphérique principal pour afficher le fichier journal de ce composant en temps réel. Remplacez `/greengrass/v2` ou `C:\greengrass\v2` par le chemin d'accès au dossier AWS IoT Greengrass racine.

Linux

```
sudo tail -f /greengrass/v2/logs/aws.iot.SiteWiseEdgeCollectorOpcua.log
```

Windows (PowerShell)

```
Get-Content C:\greengrass\v2\logs\aws.iot.SiteWiseEdgeCollectorOpcua.log -Tail 10 -Wait
```

Licences

Ce composant 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 composant.

Version	Modifications
3.1.0	Nouvelles fonctionnalités - Ajoute la configurabilité du processus de découverte des nœuds pour chaque source de données. - Ajoute la configurabilité du déclencheur de modification des données par défaut pour chaque source de données. Corrections de bogues et améliorations - Corrige les failles de sécurité.
3.0.3	Corrections de bogues et améliorations Résout un problème à l'origine de la troncature de fichiers et d'un échec de synchronisation lors de la synchronisation de configurations volumineuses.

Version	Modifications
	• Résout un problème qui entraînait des retards dans le processus de démarrage en raison de la latence de l' AWS IoT Core abonnement. • Résout un problème qui déclenchait une découverte complète lors de déconnexions momentanées des serveurs OPC UA.
3.0.2	Corrections de bogues et améliorations • Corrige la <code>OpcUaCollector.IncomingValuesCount</code> métrique pour inclure avec précision les points de capture dans les calculs.
3.0.1	Corrections de bogues et améliorations • Corrige les failles de sécurité. • Résout un problème lié à la gestion des horodatages OPC UA. Ajoute une sélection d'horodatage préférée et améliore la logique de repli lorsque les horodatages préférés ne sont pas disponibles. • Résout un problème selon lequel les configurations du collecteur OPC UA pour les flux classiques et la passerelle V2 ne passaient pas par défaut sur Stream Manager lorsqu'aucune destination n'est spécifiée.
3.0.0	Nouvelles fonctionnalités Ajout de la prise en charge des passerelles V3 compatibles MQTT, en plus des flux classiques et des passerelles V2 (anciennement connues sous le nom de passerelles Edge auto-hébergées). SiteWise Pour plus d'informations, voir Passerelles V3 compatibles MQTT pour Edge. AWS IoT SiteWise • Permet de publier les données OPC UA sur le broker AWS IoT Greengrass MQTT 5 (EMQX). Corrections de bogues et améliorations • Corrige les failles de sécurité.

Version	Modifications
2.6.0	Nouvelles fonctionnalités • Ajout de la prise en charge de l'ingestion des valeurs Null et NaN si l'ingestion est activée dans AWS IoT SiteWise. Pour afficher ou modifier la configuration Null et NaN dans AWS IoT SiteWise, consultez le DescribeStorageConfiguration et PutStorageConfiguration APIs.
2.5.1	Corrections de bogues et améliorations • Corrige un bogue selon lequel les futures tâches de capture d'écran sont annulées en cas d'erreur lors de l'exécution d'une tâche de capture instantanée. • Corrige un bogue selon lequel les mises à jour de configuration de la source de données ne sont conservées qu'après le redémarrage du collecteur OPC UA, en cas de perte de connexion au serveur OPC UA de la source de données.
2.5.0	Nouvelles fonctionnalités • Ajoute une option de source de données pour convertir des tableaux et des DateTime valeurs simples en chaînes. • Ajoute une option de groupe de propriétés permettant de sélectionner un horodatage source ou serveur lors de la collecte de données à partir d'un serveur OPC UA. Corrections de bogues et améliorations • Réinitialise la configuration d'interrogation par défaut pour utiliser l'horodatage de la source.
2.4.2	Corrections de bogues et améliorations • Résout les problèmes lors de la découverte du serveur OPC UA dans lesquels un nœud peut être découvert plusieurs fois. • Corrige la fonctionnalité de capture instantanée afin de garantir que l'horodatage est nouveau pour chaque point de données de capture instantanée.

Version	Modifications
2.4.1	Corrections de bogues et améliorations • Résout les problèmes liés à la prise en charge des proxys. • Résout un problème en raison duquel le nettoyage des threads échouait et provoquait un blocage des données.
2.4.0	Nouvelles fonctionnalités • Ajoute un journal des événements pour faciliter l'identification et la résolution des problèmes. Corrections de bogues et améliorations • Résout un problème lié au client OPC UA qui provoquait des erreurs de certificat lors de la connexion à un serveur OPC UA utilisant la version 1.05 de la spécification OPC UA.
2.3.0	Nouvelles fonctionnalités • Prend en charge la configuration du proxy HTTP Greengrass Nucleus sous Linux. Corrections de bogues et améliorations • Résout un problème de sécurité (CVE-2019-19135).
2.2.0	Nouvelles fonctionnalités • Prend en charge l'installation du pack de collecte de données sur ARMv8 l'architecture Linux. • Configuration minimale requise pour Linux ARMv8 : • Mémoire : 4 Go • Processeur : ARM Cortex-A72 ou spécification équivalente Corrections de bogues et améliorations • Améliore la journalisation des métriques dans le processus de découverte des nœuds. • Améliore la gestion des types de données non pris en charge. • Améliore l'enregistrement des erreurs de flux de données.

Version	Modifications
2.1.3	Nouvelles fonctionnalités • Ajoute la prise en charge de Windows Server 2019 ou version ultérieure. Corrections de bogues et améliorations • Améliore les messages d'erreur lorsque vous déployez ce composant sur des appareils non pris en charge.
2.1.1	Nouvelles fonctionnalités • Permet de configurer les propriétés d'abonnement suivantes : • DataChangeTrigger- Vous pouvez définir la condition qui déclenche une alerte de modification des données. • QueueSize- La profondeur de la file d'attente sur un serveur OPC-UA pour une métrique particulière où les notifications relatives aux éléments surveillés sont mises en file d'attente. • PublishingIntervalMilliseconds- Intervalle (en millisecondes) d'un cycle de publication spécifié lors de la création d'un abonnement. • SnapshotFrequencyMilliseconds - Vous pouvez configurer le paramètre de délai d'expiration de la fréquence des instantanés pour garantir qu' AWS IoT SiteWise Edge ingère un flux constant de données. • Cette version prend en charge l'ingestion de données de BAD qualité et filtre les données en fonction des qualités de données suivantes : • UNCERTAIN données de qualité • BADdonnées de qualité Corrections de bogues et améliorations • Améliorations des statistiques relatives aux clients. • Corrige le codage de sécurité qui causait parfois des problèmes lors de la connexion aux serveurs sur lesquels le chiffrement était activé. • Résout un problème en raison duquel le groupe de propriétés n'a pas pu être mis à jour.
2.0.3	Corrections de bogues et améliorations.

Version	Modifications
2.0.2	Corrections de bogues et améliorations apportées à la synchronisation des priorités des actifs avec Edge.
2.0.1	Première version.

Consultez aussi

- [Qu'est-ce que c'est AWS IoT SiteWise ?](#) dans le guide de AWS IoT SiteWise l'utilisateur.

Simulateur de source de données IoT SiteWise OPC UA

Le composant simulateur de source de données IoT SiteWise OPC UA (`aws.iot.SiteWiseEdgeOpcuaDataSourceSimulator`) démarre un serveur OPC UA local qui génère des échantillons de données. Utilisez ce serveur OPC UA pour simuler une source de données lue par le [composant collecteur IoT SiteWise OPC UA](#) sur une AWS IoT SiteWise passerelle. Vous pouvez ensuite explorer les AWS IoT SiteWise fonctionnalités à l'aide de ces exemples de données. Pour plus d'informations sur les AWS IoT SiteWise passerelles, consultez la section [Utilisation AWS IoT SiteWise du périphérique dans le](#) Guide de l'AWS IoT SiteWise utilisateur.

Rubriques

- [Versions](#)
- [Type](#)
- [Système d'exploitation](#)
- [Prérequis](#)
- [Dépendances](#)
- [Configuration](#)
- [Fichier journal local](#)
- [Journal des modifications](#)
- [Consultez aussi](#)

Versions

Les versions de ce composant sont les suivantes :

- 1,0 x

Type

Ce composant est un composant générique (`aws.greengrass.generic`). Le [noyau Greengrass](#) exécute les scripts de cycle de vie du composant.

Pour de plus amples informations, veuillez consulter [Types de composants](#).

Système d'exploitation

Ce composant peut être installé sur les appareils principaux qui exécutent les systèmes d'exploitation suivants :

- Linux
- Windows

Prérequis

Ce composant répond aux exigences suivantes :

- Le périphérique principal de Greengrass doit pouvoir utiliser le port 4840 sur l'hôte local. Le serveur OPC UA local de ce composant s'exécute sur ce port.

Dépendances

Lorsque vous déployez un composant, il déploie AWS IoT Greengrass également des versions compatibles de ses dépendances. Cela signifie que vous devez satisfaire aux exigences relatives au composant et à toutes ses dépendances pour réussir le déploiement du composant. Cette section répertorie les dépendances des [versions publiées](#) de ce composant et les contraintes de version sémantiques qui définissent les versions des composants pour chaque dépendance. Vous pouvez également consulter les dépendances de chaque version du composant dans la [AWS IoT Greengrass console](#). Sur la page de détails du composant, recherchez la liste des dépendances.

Le tableau suivant répertorie les dépendances de toutes les versions de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	>=2,3,0 <3,0,0	Stricte

Pour plus d'informations sur les dépendances des composants, consultez la [référence de la recette des composants](#).

Configuration

Ce composant ne possède aucun paramètre de configuration.

Fichier journal local

Ce composant utilise le fichier journal suivant.

Linux

```
/greengrass/v2/logs/aws.iot.SiteWiseEdgeOpcuaDataSourceSimulator.log
```

Windows

```
C:\greengrass\v2\logs\aws.iot.SiteWiseEdgeOpcuaDataSourceSimulator.log
```

Pour consulter les journaux de ce composant

- Exécutez la commande suivante sur le périphérique principal pour afficher le fichier journal de ce composant en temps réel. Remplacez */greengrass/v2* ou *C:\greengrass\v2* par le chemin d'accès au dossier AWS IoT Greengrass racine.

Linux

```
sudo tail -f /greengrass/v2/logs/  
aws.iot.SiteWiseEdgeOpcuaDataSourceSimulator.log
```

Windows (PowerShell)

```
Get-Content C:\greengrass\v2\logs  
\aws.iot.SiteWiseEdgeOpcuaDataSourceSimulator.log -Tail 10 -Wait
```

Journal des modifications

Le tableau suivant décrit les modifications apportées à chaque version du composant.

Version	Modifications
1.0.0	Première version. Ajoute la prise en charge de Windows Server 2016 ou version ultérieure.

Consultez aussi

- [Qu'est-ce que c'est AWS IoT SiteWise ?](#) dans le guide de AWS IoT SiteWise l'utilisateur.

SiteWise Éditeur IoT

Le composant SiteWise éditeur IoT (`aws.iot.SiteWiseEdgePublisher`) permet aux AWS IoT SiteWise passerelles d'exporter des données de la périphérie vers le AWS Cloud.

Pour plus d'informations sur les AWS IoT SiteWise passerelles, consultez la section [Utilisation AWS IoT SiteWise du périphérique dans le](#) Guide de l'AWS IoT SiteWise utilisateur.

Rubriques

- [Versions](#)
- [Type](#)
- [Système d'exploitation](#)
- [Exigences](#)
- [Dépendances](#)
- [Configuration](#)
- [Données d'entrée](#)
- [Fichier journal local](#)
- [Dépannage et débogage](#)
- [Licences](#)
- [Journal des modifications](#)
- [Consultez aussi](#)

Versions

Les versions de ce composant sont les suivantes :

- 4.1.x
- 4.0. x
- 3.2.x
- 3.1.x
- 3.0.x
- 2.4.x
- 2.3.x
- 2.2.x
- 2.1.x
- 2.0.x

Type

Ce composant est un composant générique (`aws.greengrass.generic`). Le [noyau Greengrass](#) exécute les scripts de cycle de vie du composant.

Pour de plus amples informations, veuillez consulter [Types de composants](#).

Système d'exploitation

Ce composant peut être installé sur les appareils principaux qui exécutent les systèmes d'exploitation suivants :

- Linux
- Windows

Exigences

Ce composant répond aux exigences suivantes :

- L'appareil principal de Greengrass doit fonctionner sur l'une des plateformes suivantes :

- Système d'exploitation : Ubuntu 18.04 ou version ultérieure
architecture : x86_64 (AMD64) ou ARMv8 (Aarch64)
- Système d'exploitation : Red Hat Enterprise Linux (RHEL) 8
architecture : x86_64 (AMD64) ou ARMv8 (Aarch64)
- Système d'exploitation : Amazon Linux 2
architecture : x86_64 (AMD64) ou ARMv8 (Aarch64)
- Système d'exploitation : Debian 11
architecture : x86_64 (AMD64) ou ARMv8 (Aarch64)
- Système d'exploitation : Windows Server 2019 ou version ultérieure
architecture : x86_64 (AMD64)
- L'appareil principal de Greengrass doit se connecter à Internet.
- L'appareil principal de Greengrass doit être autorisé à effectuer l'`iotsitewise:BatchPutAssetPropertyValue` action. Pour plus d'informations, voir [Autoriser les appareils principaux à interagir avec AWS les services](#).

Exemple stratégie d'autorisation

JSON

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

Points de terminaison et ports

Ce composant doit être capable d'effectuer des demandes sortantes vers les points de terminaison et les ports suivants, en plus des points de terminaison et des ports requis pour le fonctionnement

de base. Pour de plus amples informations, veuillez consulter [Autoriser le trafic des appareils via un proxy ou un pare-feu](#).

Endpoint	Port	Obligatoire	Description
<code>data.iots</code> <code>itewise. <i>region</i>.amazonaw</code> <code>s.com</code>	443	Oui	Publiez les données sur AWS IoT SiteWise.

Dépendances

Lorsque vous déployez un composant, il déploie AWS IoT Greengrass également des versions compatibles de ses dépendances. Cela signifie que vous devez satisfaire aux exigences relatives au composant et à toutes ses dépendances pour réussir le déploiement du composant. Cette section répertorie les dépendances des [versions publiées](#) de ce composant et les contraintes de version sémantiques qui définissent les versions des composants pour chaque dépendance. Vous pouvez également consulter les dépendances de chaque version du composant dans la [AWS IoT Greengrass console](#). Sur la page de détails du composant, recherchez la liste des dépendances.

Le tableau suivant répertorie les dépendances pour les versions 2.0.x à 2.2.x de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Noyau de Greengrass	<code>>=2,3,0<3,0.0</code>	Stricte
Gestionnaire de flux	<code>>=2,0.10<3,0.0</code>	Stricte

Pour plus d'informations sur les dépendances des composants, consultez la [référence de la recette des composants](#).

Configuration

Ce composant ne possède aucun paramètre de configuration.

Vous pouvez utiliser la AWS IoT SiteWise console ou l'API pour configurer le composant SiteWise éditeur IoT. Pour plus d'informations, voir [Étape 3 : Configuration de l'éditeur - facultatif](#) dans le guide de AWS IoT SiteWise l'utilisateur.

Données d'entrée

Ce composant lit PutAssetPropertyValueEntry les messages du gestionnaire de AWS IoT Greengrass flux. Pour plus d'informations, consultez [PutAssetPropertyValueEntry](#) dans la Référence d'API AWS IoT SiteWise .

Fichier journal local

Ce composant utilise le fichier journal suivant.

Linux

```
/greengrass/v2/logs/aws.iot.SiteWiseEdgePublisher.log
```

Windows

```
C:\greengrass\v2\logs\aws.iot.SiteWiseEdgePublisher.log
```

Pour consulter les journaux de ce composant

- Exécutez la commande suivante sur le périphérique principal pour afficher le fichier journal de ce composant en temps réel. Remplacez */greengrass/v2* ou *C:\greengrass\v2* par le chemin d'accès au dossier AWS IoT Greengrass racine.

Linux

```
sudo tail -f /greengrass/v2/logs/aws.iot.SiteWiseEdgePublisher.log
```

Windows (PowerShell)

```
Get-Content C:\greengrass\v2\logs\aws.iot.SiteWiseEdgePublisher.log -Tail 10 -Wait
```

Dépannage et débogage

Ce composant inclut un nouveau journal des événements pour aider les clients à identifier et à résoudre les problèmes. Le fichier journal est distinct du fichier journal local et se trouve à l'emplacement suivant. Remplacez `/greengrass/v2` ou `C:\greengrass\v2` par le chemin d'accès au dossier AWS IoT Greengrass racine.

Linux

```
/greengrass/v2/work/aws.iot.SiteWiseEdgePublisher/logs/  
IotSiteWisePublisherEvents.log
```

Windows

```
C:\greengrass\v2\work\aws.iot.SiteWiseEdgePublisher\logs  
\IotSiteWisePublisherEvents.log
```

Ce journal contient des informations détaillées et des instructions de dépannage. Des informations de dépannage sont fournies en même temps que les diagnostics, avec une description de la manière de remédier au problème, et parfois avec des liens vers des informations supplémentaires. Les informations de diagnostic incluent les éléments suivants :

- Niveau de gravité
- Horodatage
- Informations supplémentaires spécifiques à l'événement

Exemple Exemple de journal

```
accountBeingThrottled:  
  Summary: Data upload speed slowed due to quota limits  
  Level: WARN  
  Timestamp: '2023-06-09T21:30:24.654Z'  
  Description: The IoT SiteWise Publisher is limited to the "Rate of data points  
ingested"  
    quota for a customers account. See the associated documentation and associated  
    metric for the number of requests that were limited for more information. Note  
    that this may be temporary and not require any change, although if the issue  
continues  
    you may need to request an increase for the mentioned quota.
```

```
FurtherInformation:
- https://docs.aws.amazon.com/iot-sitewise/latest/userguide/quotas.html
- https://docs.aws.amazon.com/iot-sitewise/latest/userguide/troubleshooting-
gateway.html#gateway-issue-data-streams
AssociatedMetrics:
- Name: TotalErrorCount
  Description: The total number of errors of this type that occurred.
  Value: 327724.0
AssociatedData:
- Name: AggregatePropertyAliases
  Description: The aggregated property aliases of the throttled data.
  FileLocation: /greengrass/v2/work/aws.iot.SiteWiseEdgePublisher/./logs/data/
AggregatePropertyAliases_1686346224654.log
```

Licences

Ce composant 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 composant.

Version	Modifications
4.1.4	Corrections de bugs et améliorations • L' SiteWise éditeur IoT communique désormais les indicateurs de santé du système, tels que l'utilisation de la mémoire et du processeur par l' SiteWise éditeur IoT, à CloudWatch. • Rapport d'erreur mis à jour AWS IoT Greengrass afin que IoT SiteWise Publisher soit redémarré en cas d'échec partiel du flux de travail de démarrage.
4.1.3	Corrections de bugs et améliorations • Corrige une fuite de mémoire lors du processus de pointage des données. • Corrige les failles de sécurité.
4.1.2	Corrections de bugs et améliorations • Résout un problème lié à la synchronisation de grandes configurations.

Version	Modifications
	• Les retards d'abonnement AWS IoT Core n'affecteront plus le processus de démarrage. • Les retards dans l'initialisation des points de contrôle n'affectent plus le processus de démarrage.
4.1.1	Corrections de bugs et améliorations • Résout un problème qui entraînait une utilisation élevée du processeur et de la mémoire sur les passerelles Classic Streams V2 lors de l'envoi de données via une mémoire AWS IoT SiteWise tampon en utilisant la destination Amazon S3 après le redémarrage de l'éditeur. • Permet de remplacer le comportement de journalisation par défaut des composants en créant un fichier facultatif situé à l'adresse <code>:/greengrass/v2/work/aws.iot.SiteWiseEdgePublisher/log4j2.xml</code>.
4.1.0	Nouvelles fonctionnalités • Ajoute un outil de ligne de commande pour gérer les utilisateurs du broker EMQX MQTT intégré. Cette interface permet de gérer les comptes utilisateurs et de configurer les politiques de contrôle d'accès pour la publication de rubriques MQTT et les autorisations d'abonnement. Corrections de bugs et améliorations • Corrige le calcul qui détermine le nombre d'alias uniques pouvant figurer dans un seul fichier Parquet d'ingestion en mémoire tampon.

Version	Modifications
4.0.3	Corrections de bugs et améliorations • Met à jour le nombre maximum d'alias uniques par fichier d'ingestion AWS IoT SiteWise mis en mémoire tampon pour qu'il corresponde à la limite de 10 000 dans le cloud. • Résout un problème de AWS IoT Greengrass journalisation qui signalait de manière incorrecte des états de service non valides. • Ajoute une validation pour empêcher la duplication des noms de flux dans les configurations de destination Amazon S3 dans les passerelles Classic Streams et V2. • Résout un problème de compatibilité lié à l'ingestion en AWS IoT SiteWise mémoire tampon dans des partitions non standard AWS . • Résout un problème à l'origine de l'ingestion de données dupliquées.
4.0.2	Corrections de bugs et améliorations • Résout un problème qui empêchait le téléchargement de données après l'expiration du délai d'attente par lots. • Ajoute une validation pour vérifier que chaque destination dans les configurations de passerelle V3 compatibles MQTT est unique.
4.0.1	Corrections de bugs et améliorations • Résout un problème qui empêchait le téléchargement des données AWS IoT SiteWise lorsque vous désactivez la configuration du temps d'attente maximal par lots. • Résout un problème selon lequel la <code>IoTSiteWisePublisher.NumberOfSubscriptionsToMqttBroker</code> métrique indique de manière inexacte le nombre de sujets uniques souscrits au broker MQTT.

Version	Modifications
4.0.0	Nouvelles fonctionnalités Ajoute la prise en charge des passerelles V3 compatibles MQTT en plus des flux classiques et des passerelles V2 (anciennement connues sous le nom de passerelles Edge auto-hébergées). SiteWise • Permet de s'abonner au broker AWS IoT Greengrass MQTT 5 (EMQX) et d'en recevoir des données. Pour plus d'informations, voir Passerelles V3 compatibles MQTT pour Edge. AWS IoT SiteWise • Ajoute des options de configuration pour les destinations de données à l'aide de filtres de chemin. Pour plus d'informations, voir Comprendre les filtres de chemin pour les destinations AWS IoT SiteWise Edge. Corrections de bugs et améliorations • Résout un problème selon lequel la PublishSuccessCount métrique indique des valeurs négatives. • Résout un problème selon lequel l'éditeur ne démarre pas dans les 100 secondes et entre dans l'BROKENétat. • Corrige les failles de sécurité.
3.2.0	Nouvelles fonctionnalités • Ajout de la prise en charge de l'ingestion des valeurs Null et NaN si l'ingestion est activée dans AWS IoT SiteWise. Pour afficher ou modifier la configuration Null et NaN dans AWS IoT SiteWise, consultez le DescribeStorageConfiguration et PutStorageConfiguration APIs. Corrections de bugs et améliorations • Résout les problèmes à l'origine de fichiers de base de données de points de contrôle corrompus. • Résout les problèmes de génération de métriques dupliquées.

Version	Modifications
3.1.4	 Warning La version 3.1.4 a été abandonnée le 20 février 2025. Les améliorations apportées à cette version sont disponibles dans les versions ultérieures de ce composant. Corrections de bugs et améliorations • Résout les problèmes susceptibles d'entraîner des délais de longer-than-expected démarrage en cas de déconnexion.
3.1.3	Corrections de bugs et améliorations • Résout un problème en raison duquel le fichier journal des événements situé à <code>l'/greengrass/v2/work/aws.iot.SiteWiseEdgePublisher/logs/IoTSiteWisePublisherEvents.log</code> adresse a été créé alors qu'aucun événement n'était enregistré. • Ajoute les CloudWatch métriques suivantes pour surveiller la connexion avec le broker MQTT : • <code>IoTSiteWisePublisher.IsConnectedToMqttBroker</code> • <code>IoTSiteWisePublisher.NumberOfSubscriptionsToMqttBroker</code> • <code>IoTSiteWisePublisher.NumberOfUniqueMqttTopicsReceived</code> • <code>IoTSiteWisePublisher.MqttMessageReceivedSuccessCount</code> • <code>IoTSiteWisePublisher.MqttReceivedSuccessBytes</code> Pour plus d'informations sur ces mesures, consultez la section Mesures de AWS IoT Greengrass Version 2 passerelle. • Résout un problème en raison duquel l'<code>BatchCreateJob</code> API est toujours appelée même en cas d'échec du téléchargement d'un fichier parquet vers S3.

Version	Modifications
3.1.2	Corrections de bugs et améliorations • Résout le problème d'utilisation élevée du processeur introduit dans la version 3.1.1.
3.1.1	 Warning La version 3.1.1 a été abandonnée le 12 mars 2024. Les améliorations apportées à cette version sont disponibles dans les versions ultérieures de ce composant. Corrections de bugs et améliorations • Ajoute une journalisation supplémentaire qui identifie les alias de données concernés en cas d'erreur. • Ajoute l'application locale des limites d' AWS IoT SiteWise API relatives à l'âge des données ingérées. • Résout le problème selon lequel Publisher mélange les points de contrôle des StreamManager flux lorsqu'il existe plusieurs destinations Amazon S3. • Résout le problème de performance lié à la manière dont l'éditeur lit les StreamManager flux.
3.1.0	Nouvelles fonctionnalités • Permet de publier des données sous forme de fichiers parquet sur Amazon S3. • Ajoute un support pour l'ingestion AWS IoT SiteWise en mémoire tampon.
3.0.0	Corrections de bugs et améliorations • Résout les problèmes liés à la prise en charge des proxys. Nouvelles fonctionnalités • Permet de prendre en charge l'ingestion de données à partir d'un courtier MQTT.

Version	Modifications
2.4.1	Corrections de bugs et améliorations • Activez le composant pour qu'il fonctionne avec les versions 11.0.20.8 .1 et ultérieures de Java Corretto 11. Les versions 2.4.0 et 2.3.3 du composant affichent le message "Could not find or load main class" d'erreur lorsqu'elles sont utilisées avec Java Corretto version 11.0.20.8.1.
2.4.0	Nouvelles fonctionnalités • Ajoute un nouveau journal des événements pour faciliter l'identification et la résolution des problèmes. Corrections de bugs et améliorations • Améliore la restauration des points de contrôle de Publisher.
2.3.3	Corrections de bugs et améliorations • Améliore la capacité à prendre en charge un débit élevé.
2.3.2	Corrections de bugs et améliorations • Résout le support du proxy HTTP lors du téléchargement de la configuration de Publisher.
2.3.1	Nouvelles fonctionnalités • Prend en charge l'installation du pack de collecte de données sur ARMv8 l'architecture Linux. • Configuration minimale requise pour Linux ARMv8 : • Mémoire : 4 Go • Processeur : ARM Cortex-A72 ou spécification équivalente
2.2.3	Corrections de bugs et améliorations • Supprime la nouvelle tentative pour une exception générique qui ne figurait pas dans la liste des exceptions récupérables.

Version	Modifications
2.2.2	Corrections de bugs et améliorations • Réintroduit la prise en charge du téléchargement de données AWS IoT SiteWise via un serveur proxy HTTP.
2.2.1	 Note Cette version ne prend pas en charge la configuration du proxy HTTP. Les versions 2.2.2 et supérieures réintroduisent le support de cette fonctionnalité. Nouvelles fonctionnalités • Ajoute la prise en charge de ce composant pour activer la compression lors du téléchargement de données vers. AWS IoT SiteWise

Version	Modifications
2.2.0	 Note Cette version ne prend pas en charge la configuration du proxy HTTP. Les versions 2.2.2 et supérieures réintroduisent le support de cette fonctionnalité. Nouvelles fonctionnalités • Met à jour ce composant pour compresser les données avant de les envoyer au AWS IoT SiteWise service. • Dans la plupart des cas, cette modification réduit l'utilisation de la bande passante de 75 % par rapport aux versions précédentes de ce composant. • Dans la plupart des cas, cette modification augmente l'utilisation du processeur jusqu'à 5 %. Sur les passerelles qui traitent de grandes quantités de données, cette modification peut augmenter l'utilisation du processeur jusqu'à 15 %. • Cette modification n'a aucune incidence sur les frais AWS IoT SiteWise de service ni sur l'utilisation des quotas de service. • Ajoute la prise en charge de Windows Server 2019 ou version ultérieure. Corrections de bugs et améliorations • Résout un problème qui empêchait ce composant de démarrer lorsque le fichier de point de contrôle est endommagé.
2.1.4	Corrections de bugs et améliorations • Corrige la compatibilité avec la version 8 de Java.

Version	Modifications
2.1.3	 Warning Cette version n'est plus disponible, sauf dans les régions de l'est des États-Unis (Ohio), du Canada (centre) et AWS GovCloud (de l'est des États-Unis). Cette version du composant nécessite la version 11 ou supérieure de Java pour fonctionner. Les améliorations apportées à cette version sont disponibles dans les versions ultérieures de ce composant. Corrections de bugs et améliorations • Améliore les messages d'erreur lorsque vous déployez ce composant sur des appareils non pris en charge. • Mises à jour pour enregistrer les erreurs en cas d'échec des téléchargements de données.
2.1.2	Corrections de bugs et améliorations • Mises à jour pour invoquer la fonctionnalité d'exportation des données expirées dès que les données expirent.
2.1.1	Corrections de bugs et améliorations.
2.1.0	Nouvelles fonctionnalités • Permet de publier d'abord les données les plus récentes dans le cloud. • Permet de ne pas publier les données expirées dans le cloud. • Ajoute la prise en charge du stockage local des données expirées. Corrections de bugs et améliorations • Réduit le disque I/O et la latence correspondante.
2.0.2	Corrections de bugs et améliorations.
2.0.1	Première version.

Consultez aussi

- [Qu'est-ce que c'est AWS IoT SiteWise ?](#) dans le guide de AWS IoT SiteWise l'utilisateur.

SiteWise Processeur IoT

Le composant SiteWise processeur IoT (`aws.iot.SiteWiseEdgeProcessor`) permet aux flux AWS IoT SiteWise classiques et aux passerelles V2 de traiter les données à la périphérie.

Avec ce composant, les AWS IoT SiteWise passerelles peuvent utiliser des modèles d'actifs et des actifs pour traiter les données sur les périphériques de passerelle. Pour plus d'informations sur les AWS IoT SiteWise passerelles, consultez la section [Utilisation AWS IoT SiteWise du périphérique dans le](#) Guide de l'AWS IoT SiteWise utilisateur.

Note

La fonctionnalité du pack de traitement des données (DPP) ne sera plus ouverte aux nouveaux clients à compter du 7 novembre 2025. Si vous souhaitez utiliser DPP, inscrivez-vous avant cette date. Les clients existants peuvent continuer à utiliser le service normalement. Pour plus d'informations, voir [Modification de la disponibilité des packs de traitement des données](#) dans le Guide de AWS IoT SiteWise l'utilisateur.

Rubriques

- [Versions](#)
- [Type](#)
- [Système d'exploitation](#)
- [Exigences](#)
- [Dépendances](#)
- [Configuration](#)
- [Fichier journal local](#)
- [Licences](#)
- [Journal des modifications](#)
- [Consultez aussi](#)

Versions

Les versions de ce composant sont les suivantes :

- 3.5.x
- 3.4.x
- 3.3.x
- 3.2.x
- 3.1.x
- 3.0.x
- 2.2.x
- 2.1.x
- 2.0.x

Type

Ce composant est un composant générique (`aws.greengrass.generic`). Le [noyau Greengrass](#) exécute les scripts de cycle de vie du composant.

Pour de plus amples informations, veuillez consulter [Types de composants](#).

Système d'exploitation

Ce composant peut être installé sur les appareils principaux qui exécutent les systèmes d'exploitation suivants :

- Linux
- Windows

Exigences

Ce composant répond aux exigences suivantes :

- L'appareil principal de Greengrass doit fonctionner sur l'une des plateformes suivantes :
 - Système d'exploitation : Ubuntu 20.04 ou version ultérieure

architecture : x86_64 () AMD64

- Système d'exploitation : Red Hat Enterprise Linux (RHEL) 8

architecture : x86_64 () AMD64

- Système d'exploitation : Amazon Linux 2

architecture : x86_64 () AMD64

- Système d'exploitation : Windows Server 2019 ou version ultérieure

architecture : x86_64 () AMD64

- Système d'exploitation : Debian 11 (Bullseye) ou version ultérieure

architecture : x86_64 () AMD64

- Le périphérique principal de Greengrass doit autoriser le trafic entrant sur le port 443.
- Le périphérique principal de Greengrass doit autoriser le trafic sortant sur les ports 443 et 8883.
- Les ports suivants sont réservés pour être utilisés par AWS IoT SiteWise : 80, 443, 3001, 4569, 4572, 8000, 8081, 8082, 8084, 8085, 8086, 8445, 9000, 9500, 11080 et 50010. L'utilisation d'un port réservé pour le trafic peut entraîner l'interruption de la connexion.

Note

Le port 8087 n'est requis que pour les versions 2.0.15 et ultérieures de ce composant.

- Le [rôle d'appareil Greengrass](#) doit disposer d'autorisations vous permettant d'utiliser des AWS IoT SiteWise passerelles sur vos appareils. AWS IoT Greengrass V2 Pour plus d'informations, consultez la section [Exigences](#) du guide de AWS IoT SiteWise l'utilisateur.

Points de terminaison et ports

Ce composant doit être capable d'effectuer des demandes sortantes vers les points de terminaison et les ports suivants, en plus des points de terminaison et des ports requis pour le fonctionnement de base. Pour de plus amples informations, veuillez consulter [Autoriser le trafic des appareils via un proxy ou un pare-feu](#).

Endpoint	Port	Obligatoire	Description
<code>model.iotsitewise. <i>region</i>.amazonaws.com</code>	443	Oui	Obtenez des

Endpoint	Port	Obligatoire	Description
			informati ons sur vos AWS IoT SiteWise actifs et vos modèles d'actifs.
edge.iots itewise. <i>region</i> .amazonaw s.com	443	Oui	Obtenez des informati ons sur la configura tion de la AWS IoT SiteWise passerell e du périphéri que principal.

Endpoint	Port	Obligatoire	Description
<code>ecr.<i>region</i>.amazonaws.com</code>	443	Oui	Téléchargez les images Docker de la passerelle AWS IoT SiteWise Edge depuis Amazon Elastic Container Registry.
<code>iot.<i>region</i>.amazonaws.com</code>	443	Oui	Obtenez des terminaux pour votre Compte AWS.
<code>sts.<i>region</i>.amazonaws.com</code>	443	Oui	Obtenez l'identifiant de votre Compte AWS.
<code>monitor.iotsitewise.<i>region</i>.amazonaws.com</code>	443	Non	Obligatoire si vous accédez aux AWS IoT SiteWise Monitor portails sur l'appareil principal.

Dépendances

Lorsque vous déployez un composant, il déploie AWS IoT Greengrass également des versions compatibles de ses dépendances. Cela signifie que vous devez satisfaire aux exigences relatives au composant et à toutes ses dépendances pour réussir le déploiement du composant. Cette section répertorie les dépendances des [versions publiées](#) de ce composant et les contraintes de version sémantiques qui définissent les versions des composants pour chaque dépendance. Vous pouvez également consulter les dépendances de chaque version du composant dans la [AWS IoT Greengrass console](#). Sur la page de détails du composant, recherchez la liste des dépendances.

Le tableau suivant répertorie les dépendances pour les versions 2.0.x à 2.1.x de ce composant.

Dépendance	Versions compatibles	Type de dépendance
Service d'échange de jetons	>=2,0,3 <3,0,0	Stricte
Gestionnaire de flux	>=2,0.10 <3,0.0	Stricte
Greengrass CLI	>=2,3,0 <3,0,0	Stricte

Pour plus d'informations sur les dépendances des composants, consultez la [référence de la recette des composants](#).

Configuration

Ce composant ne possède aucun paramètre de configuration.

Fichier journal local

Ce composant utilise le fichier journal suivant.

Linux

```
/greengrass/v2/logs/aws.iot.SiteWiseEdgeProcessor.log
```

Windows

```
C:\greengrass\v2\logs\aws.iot.SiteWiseEdgeProcessor.log
```

Pour consulter les journaux de ce composant

- Exécutez la commande suivante sur le périphérique principal pour afficher le fichier journal de ce composant en temps réel. Remplacez `/greengrass/v2` ou `C:\greengrass\v2` par le chemin d'accès au dossier AWS IoT Greengrass racine.

Linux

```
sudo tail -f /greengrass/v2/logs/aws.iot.SiteWiseEdgeProcessor.log
```

Windows (PowerShell)

```
Get-Content C:\greengrass\v2\logs\aws.iot.SiteWiseEdgeProcessor.log -Tail 10 -  
Wait
```

Licences

Ce composant inclut les logiciels/licences tiers suivants :

- Apache-2.0
- MIT
- Clause BSD-2
- Clause BSD-3
- CDDL-1.0
- CDDL-1.1
- DISQUE
- Zlib
- GPL-3.0 avec exception GCC
- Domaine public
- Python-2.0
- Unicode-DFS-2015
- Clause BSD-1
- OpenSSL
- EPL-1,0

- EPL-2.0
- GPL 2.0- with-classpath-exception
- MPL-2.0
- CC0-1,0
- JSON

Ce composant 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 composant.

Version	Modifications
3.5.1	Nouvelles fonctionnalités Ajout de la prise en charge de l'ingestion des valeurs Null et NaN si l'ingestion est activée dans AWS IoT SiteWise. Pour afficher ou modifier la configuration Null et NaN dans AWS IoT SiteWise, consultez le DescribeStorageConfiguration et PutStorageConfiguration APIs. Corrections de bogues et améliorations Dépendances mises à jour pour corriger les vulnérabilités de sécurité potentielles.
3.4.0	Nouvelles fonctionnalités • Ajout de la prise en charge optionnelle des proxys HTTP et HTTPS pour permettre la communication par passerelle via des serveurs proxy qui nécessitent des certificats personnalisés. Pour plus d'informations sur la configuration d'un proxy HTTP pour le AWS IoT Greengrass cœur, consultez Connexion au port 443 ou via un proxy réseau. Pour plus d'informations sur la prise en charge des proxys spécifiques à SiteWise Edge, consultez la section Gérer les magasins de confiance pour la prise en charge des proxys AWS IoT SiteWise Edge dans le guide de AWS IoT SiteWise l'utilisateur. • Ajout de paramètres de délai d'expiration de session configurables pour gérer les périodes d'inactivité pour AWS OpsHub et SiteWise Edge.

Version	Modifications
	APIs Pour plus d'informations, voir Configurer les délais d'expiration des sessions pour AWS IoT SiteWise Edge dans le guide de l'AWS IoT SiteWise utilisateur. Améliorations des performances Réduction du temps nécessaire pour que les données entrantes atteignent le stockage périphérique de 5 secondes à moins d'une seconde. La latence des téléchargements de données AWS IoT SiteWise reste inchangée.
3.3.1	Nouvelle fonctionnalité • Ajout du support CORS optionnel à SiteWise Edge APIs, améliorant les capacités de partage de ressources entre origines. Cette fonctionnalité améliore la flexibilité des applications Web interagissant avec les APIs.
3.3.0	Améliorations des performances • Mécanisme d'actualisation du cache optimisé pour réduire l'I/O l'utilisation entre les AWS IoT SiteWise synchronisations d'actifs en actualisant uniquement les entrées pour les actifs nouveaux ou mis à jour. • Encombrement mémoire réduit pour maintenir un cache contenant un grand nombre de propriétés de ressources synchronisées. Corrections de bogues et améliorations • Suppression des journaux pour l'ingestion des valeurs de propriétés individuelles en l'absence d'erreur d'ingestion, ce qui réduit le bruit des journaux en cas de taux d'ingestion élevés. • Meilleure lisibilité du journal grâce à l'utilisation d'un format lisible par l'homme pour certaines entrées du journal. • Ajout du support pour Java 17 et versions ultérieures.

Version	Modifications
3.2.1	Corrections de bogues et améliorations • Correction d'un problème en raison duquel les appels AWS IoT SiteWise d'API ne paginaient pas de manière synchrone avec SiteWise Edge. • Correction du problème consistant à ne plus publier la <code>MessageRemaining.SiteWise_Edge_Stream</code> métrique. • Les CloudWatch métriques suivantes ont été ajoutées pour surveiller la connexion avec le broker MQTT. • <code>IoTSiteWiseProcessor.IsConnectedToMqttBroker</code> • <code>IoTSiteWiseProcessor.NumberOfSubscriptionsToMqttBroker</code> • <code>IoTSiteWiseProcessor.NumberOfUniqueMqttTopicsReceived</code> • <code>IoTSiteWiseProcessor.MqttMessageReceivedSuccessCount</code> • <code>IoTSiteWiseProcessor.MqttReceivedSuccessBytes</code> Pour plus d'informations sur ces mesures, consultez la section Mesures de AWS IoT Greengrass Version 2 passerelle.

Version	Modifications
3.2.0	Améliorations des performances - Optimisez les services d'API afin de réduire l'encombrement mémoire et de nécessiter moins d'espace disque pour l'installation - Cela permet de réduire de 2 Go l'utilisation initiale de la mémoire (utilise désormais 7,5 Go de mémoire au démarrage, mais 16 Go sont toujours recommandés) et de 500 Mo de réduction de la taille du téléchargement (nécessite désormais un téléchargement de 1,4 Go) pour l'ensemble du composant. Nouvelles fonctionnalités <code>GetAssetPropertyValueAggregates</code> L'API prend désormais en charge les fenêtres d'agrégation de 15 minutes en périphérie. - Les ports 8081 et 8082 n'ont plus besoin d'être disponibles pour que ce composant fonctionne correctement.  Note Le point de terminaison local pour APIs le plan de AWS IoT SiteWise données <code>get-asset-property-value</code>, tel que, passe de <code>http://localhost:8081</code> à <code>http://localhost:11080/data</code>. Le point de terminaison local du plan de AWS IoT SiteWise contrôle APIs <code>list-asset-models</code>, tel que, passe de <code>http://localhost:11080</code> à <code>http://localhost:11080/control</code>. AWS recommande toujours d'utiliser les points de terminaison HTTPS de la passerelle SiteWise Edge. Ces points de terminaison n'ont pas changé. Corrections de bogues et améliorations - La synchronisation à partir de AWS IoT SiteWise permet désormais de faire passer les ressources à un état valide si la synchronisation précédente a été interrompue. Cela permettra de résoudre les problèmes liés à la corruption de certaines ressources après un redémarrage forcé.

Version	Modifications
	• Corrige une situation rare dans laquelle une ressource peut être corrompue en périphérie si elle est modifiée pendant la synchronisation. La synchronisation échouera désormais si cette condition est détectée, et la ressource sera réessayée lors de la prochaine synchronisation. • Résout un problème qui aurait pu permettre au point de terminaison HTTP APIs d'être appelé de l'extérieur. Seul le protocole HTTPS peut désormais être utilisé pour appeler APIs en dehors de l'adresse de boucle locale. • <code>ListAssets</code> L'API affiche désormais la hiérarchie des actifs pour les actifs stockés en périphérie. • Résout un problème d'échec du redémarrage, de mise à niveau ou de rétrogradation du pack de traitement des données sous Windows. • Corrige un bogue dans le pack de traitement des données pour système d'exploitation Windows qui empêchait les clients d'utiliser des informations d'identification pour se connecter à un courtier MQTT.

3.1.3

Corrections de bogues et améliorations

- Correction d'un problème selon lequel le pack de traitement des données signalait à tort une synchronisation réussie alors que certaines ressources échouaient réellement.
- Autorisez plusieurs actifs à porter le même nom tant qu'ils n'ont pas le même parent.

3.1.1

Corrections de bogues et améliorations

- Correction d'un problème d'échec de la demande SigV4 en raison d'une incompatibilité de fuseau horaire.
- Correction d'un problème selon lequel les propriétés de transformation et de métrique cessent d'être calculées lorsqu'elles reposent sur des attributs après le redémarrage.
- Activez la prise en charge de la configuration personnalisée du port Stream Manager.
- Corrigez un problème en raison duquel les propriétés synchronisées avec le périphérique pouvaient ne plus être mises à jour.

Version	Modifications
3.1.0	Corrections de bogues et améliorations • Correction d'un problème en raison duquel <code>ListAssetModels</code> l'API ne parvient pas à générer le jeton suivant.
3.0.0	Nouvelles fonctionnalités • Permet de prendre en charge l'ingestion de données à partir d'un courtier MQTT.
2.2.1	Corrections de bogues et améliorations • Ajustez le processus de synchronisation afin de rendre le stockage des données du plan de contrôle plus cohérent avec le fonctionnement du cloud. Cela a un léger impact sur la mise à niveau.  Note Les données du plan de contrôle synchronisées sur la version 2.2.1 ou supérieure ne seront pas compatibles avec les versions précédentes. Pour revenir aux versions précédentes, vous devez effectuer une nouvelle installation. Cela n'a aucun impact sur les mises à niveau, les données synchronisées sur les versions précédentes fonctionneront avec la version 2.2.1. • Modifications supplémentaires apportées à la chaîne AWS d'informations d'identification pour hiérarchiser les AWS IoT Greengrass V2 informations d'identification.

Version	Modifications
2,137	Corrections de bogues et améliorations • Dépréciez le dependency-routing-service processus et intégrez ses fonctionnalités au property-state-service processus afin de réduire l'utilisation des ressources liées à la communication des processus. • Augmentez la limite maximale de résultats pour l'get-asset-property-value-history API à 20 000 afin qu'elle corresponde à la limite utilisée par AWS IoT SiteWise. • Correction d'un problème en raison duquel le jeton suivant n'était pas fourni dans les résultats paginés de l'get-asset-property-value-history API alors qu'aucune limite de résultats maximale n'était spécifiée.
2,135	Corrections de bogues et améliorations • Modifie la chaîne AWS d'informations d'identification pour hiérarchiser les AWS IoT Greengrass informations d'identification. • Résout un problème lié à la détection des comptes lors du déploiement dans le cadre d'un AWS IoT groupe d'objets.
2.1.34	Corrections de bogues et améliorations • Ajuste les metric/transform calculs pour utiliser le multithreading sous Linux. Windows continue d'exécuter des calculs monothread pour des raisons de compatibilité. • Résout un problème en raison duquel les calculs métriques étaient absents pour certaines fenêtres de calcul.
2.1.33	Corrections de bogues et améliorations • Résout un problème lié au signalement de l'état d'erreur sur la console Greengrass.
2.1.32	Corrections de bogues et améliorations • Ajoute la prise en charge des noms d'utilisateur et des groupes personnalisés.

Version	Modifications
2.1.31	Corrections de bogues et améliorations • Prend en charge le calcul de la moyenne pondérée dans le temps et de l'écart type pondéré dans le temps pour les données modélisées. AWS IoT SiteWise
2.1.29	Corrections de bogues et améliorations • Ajoute la prise en charge de la fonctionnalité de filtrage des actifs en périphérie.
2.1.28	Corrections de bogues et améliorations • Optimise la synchronisation des ressources pour permettre à un grand nombre de ressources de se synchroniser de la périphérie AWS Cloud à la périphérie.
2.1.24	Corrections de bogues et améliorations • Résout un problème qui entraînait la disparition du tableau de bord lors de la deuxième synchronisation d'une ressource.
2.1.23	Corrections de bogues et améliorations • Ajout d'un délai d'attente pour le <code>aws.iot.SiteWiseEdgeProcess</code> ou processus d'installation afin d'éviter l'échec de l'installation si la connexion Internet est lente. • Synchronisation des ressources optimisée pour améliorer l'efficacité de la synchronisation entre le cloud et la périphérie.
2.1.21	 Warning La mise à niveau de la version 2.0.x vers la version 2.1.x entraînera une perte de données locales. Nouvelles fonctionnalités • Ajoute la prise en charge de Windows Server 2019 ou version ultérieure. • Supprime le docker pour les systèmes d'exploitation basés sur Linux.

Version	Modifications
2,0,16	Cette version contient des corrections de bogues et des améliorations.
2,0,15	Corrections de bugs et améliorations • Modifie le port que ce composant utilise pour les opérations d'API de synchronisation des ressources de 8085 à 8087. Par conséquent, ce composant nécessite désormais le port 8087 pour être disponible. Ce composant nécessite toujours que le port 8085 soit disponible. • Met à jour AWS OpsHub l'authentification pour refuser les utilisateurs non autorisés lors de la connexion plutôt que lorsqu'un utilisateur tente d'appeler des opérations d'API.
2,0,14	Cette version contient des corrections de bogues et des améliorations.
2.0.13	Corrections de bugs et améliorations • Résout un problème : lorsque ce composant transmet des données à Amazon CloudWatch Metrics, il indique désormais correctement quelles données ne sont pas modélisées.
2.0.9	Corrections de bugs et améliorations • Améliore la fiabilité pour créer et mettre à jour les AWS IoT SiteWise ressources sur le périphérique principal. • Ajoute des opérations d'API locales supplémentaires que vous pouvez utiliser pour surveiller les composants installés sur le périphérique principal, la version de chaque composant et l'état de chaque composant. Vous pouvez consulter ces informations dans l'onglet Paramètres de l' AWS IoT SiteWise application AWS OpsHub for sur l'appareil principal. • Ajoute un état de santé pour les conteneurs Docker exécutés par ce composant. Vous pouvez exécuter la <code>docker ps</code> commande pour voir l'état de santé des conteneurs.
2.0.7	Corrections de bugs et améliorations • Corrige la prise en charge de l'affichage des AWS IoT SiteWise Monitor portails sur le périphérique principal.

Version	Modifications
2.0.6	Corrections de bugs et améliorations • Corrige les <code>latest()</code> fonctions AWS IoT SiteWise <code>statetime()</code> <code>earliest()</code> , et que ce composant calcule sur le périphérique principal.
2.0.5	Corrections de bugs et améliorations • Ajoute la prise en charge de la AWS IoT SiteWise <code>pretrigger()</code> fonction dans les transformations que ce composant calcule sur le périphérique principal. • Modifie le chemin où ce composant stocke la configuration LDAP (Lightweight Directory Access Protocol) pour l'authentification.
2.0.2	Première version.

Consultez aussi

- [Qu'est-ce que c'est AWS IoT SiteWise ?](#) dans le guide de AWS IoT SiteWise l'utilisateur.

Composants pris en charge par l'éditeur

Les composants pris en charge par l'éditeur sont disponibles dans une version préliminaire AWS IoT Greengrass et sont susceptibles d'être modifiés. Ces composants ne sont pas pris en charge par AWS. Vous devez contacter l'éditeur pour tout problème lié à chacun des composants.

Les composants pris en charge par Greengrass Publisher sont développés, proposés et entretenus par des fournisseurs de composants tiers. Les fournisseurs de composants tiers proviennent soit du AWS Partner Device Catalog, soit de AWS Heroes, soit de la communauté. Vous pouvez acheter les composants de ce catalogue en contactant directement le fournisseur de composants tiers.

Les composants pris en charge par Greengrass Publisher sont les suivants :

Rubriques

- [AIShield.Bord](#)

- [EdgeLabs Capteur AI](#)
- [Greengrass S3 Inventor](#)

AIShield.Bord

Ce composant a été développé et est pris AIShield en charge par Bosch. Renforcez la sécurité de votre IA avec AIShield .Edge. Ce composant est conçu pour déployer de manière fluide des défenses adaptées aux menaces sur les appareils périphériques, afin de protéger vos appareils contre les attaques basées sur l'IA.

Ce composant offre les avantages suivants :

- Passez en douceur de l'analyse des vulnérabilités grâce à la sécurité de l' AIShield IA à des défenses de pointe renforcées au sein de AWS
- Déployez facilement des défenses personnalisées sur plusieurs appareils de pointe
- Protection étendue adaptée à diverses configurations d'IA qui prend en charge différents types de modèles et de cadres
- Restez à jour grâce à une intégration parfaite dans les flux Amazon SageMaker AI de travail Greengrass
- Obtenez des informations immédiates sur les menaces potentielles, grâce aux données transmises directement à AWS IoT Core
- Une voie de sécurité cohérente basée sur l'IA pour le déploiement de la défense à la périphérie, selon AIShield AI Security on the AWS Marketplace

Ce composant doit être exécuté sur la plate-forme suivante :

- système d'exploitation : Linux

Si vous souhaitez acheter ce composant, contactez Bosch Software and Digital Solutions : [<AIShield.Contact@bosch.com>](mailto:AIShield.Contact@bosch.com).

EdgeLabs Capteur AI

Ce composant a été développé et est soutenu par l'IA EdgeLabs. AI EdgeLabs Sensor est une application basée sur des conteneurs qui contient des fonctionnalités de détection et de prévention

des menaces basées sur l'IA. Le capteur AI est intégré à un composant Greengrass et déployé en tant que conteneur autonome sur le périphérique principal aux côtés d'autres composants Greengrass.

Ce composant actuel est un agent basé sur un conteneur qui vérifie en permanence les communications réseau et recherche les modèles de menace dans les logiciels exécutés sur l'hôte Edge ou la passerelle IoT. Ce composant utilise l'eBPF, la vérification comportementale de la bande passante des processus et la configuration basée sur l'hôte. Les principales fonctionnalités de ce composant sont basées sur les fonctions NDR/IPS et EDR.

Ce composant offre les avantages suivants :

- Détection des menaces basée sur l'IA contre les attaques réseau et les logiciels malveillants (EDR/NDR)
- Réponse aux incidents (IPS) automatisée basée sur l'IA
- Renseignements sur les menaces au niveau local de l'hôte avec un transfert de données minimal vers l'extérieur
- Déploiement léger avec Docker et Greengrass

Ce composant doit être exécuté sur l'une des plateformes suivantes :

- système d'exploitation : Linux

Si vous souhaitez acheter ce composant, contactez AI EdgeLabs : <contact@edgelabs.ai>.

Greengrass S3 Inventor

Ce composant a été développé et est soutenu par Nathan Glover. [Le composant Greengrass S3 Ingestor est conçu pour être utilisé avec le composant de gestion de flux.](#) Ce composant prend un flux de messages JSON délimité par des lignes depuis le gestionnaire de flux et les regroupe dans un fichier GZIP. Ce composant permet une ingestion efficace des données dans Amazon S3 pour un traitement ultérieur ou pour le stockage. Ce composant ne prend pas en charge l'envoi de données vers le AWS Cloud en temps réel.

Ce composant doit être exécuté sur l'une des plateformes suivantes :

- système d'exploitation : Linux

- Système d'exploitation : Windows

Si vous souhaitez acheter ce composant, contactez Nathan Glover : <nathan@glovers.id.au.>

Composantes communautaires

Le catalogue des logiciels Greengrass est un index des composants de Greengrass développés par la communauté Greengrass. À partir de ce catalogue, vous pouvez télécharger, modifier et déployer des composants pour créer vos applications Greengrass. Vous pouvez consulter le catalogue sur le lien suivant : <https://github.com/aws-greengrass/aws-greengrass-software-catalog>.

Chaque composant possède un GitHub référentiel public que vous pouvez explorer. Consultez le catalogue des logiciels Greengrass GitHub pour trouver la liste complète des composants de la communauté. Par exemple, ce catalogue inclut les composants suivants :

- [Amazon Kinesis Video Streams](#)

Ce composant ingère les flux audio et vidéo des caméras locales qui utilisent le protocole [RTSP \(Real Time Streaming Protocol\)](#). Le composant télécharge ensuite les flux audio et vidéo [sur Amazon Kinesis Video Streams](#).

- [Passerelle Bluetooth IoT](#)

Ce composant utilise la [BluePy](#) bibliothèque qui permet la communication avec les appareils Bluetooth Low Energy (LE) pour créer des interfaces client Bluetooth LE.

- [Rotateur de certificats](#)

Ce composant fournit un moyen de faire alterner le certificat de l'appareil AWS IoT Greengrass principal et la clé privée à grande échelle au sein de votre flotte.

- [Tunneling sécurisé conteneurisé](#)

Ce composant fournit un conteneur Docker pour un tunneling sécurisé avec toutes les dépendances et les bibliothèques correspondantes dans une recette réutilisable qui ne repose pas sur un système d'exploitation hôte spécifique.

- [Grafana](#)

Ce composant vous permet d'héberger un serveur [Grafana](#) sur un appareil principal de Greengrass. Vous pouvez utiliser les tableaux de bord Grafana pour visualiser et gérer les données sur l'appareil principal.

- [GStreamer pour Amazon Lookout for Vision](#)

Ce composant fournit un GStreamer plugin qui vous permet de détecter les anomalies de Lookout for Vision dans vos GStreamer pipelines personnalisés.

- [Assistant à domicile](#)

Ce composant permet au client d'utiliser [Home Assistant](#) pour contrôler localement les appareils domotiques. Il permet l'intégration avec les AWS services à la périphérie et dans le cloud pour fournir des solutions domotiques qui étendent Home Assistant.

- [DBGrafana Tableau de bord Influx](#)

Ce composant fournit une expérience en un clic pour configurer les composants InfluxDB et Grafana. Il connecte InfluxDB à Grafana et automatise la configuration d'un tableau de bord Grafana local qui affiche la télémétrie en temps réel. AWS IoT Greengrass

- [InfluxDB](#)

Ce composant fournit une base de données de séries chronologiques [InfluxDB](#) sur un périphérique principal Greengrass. Vous pouvez utiliser ce composant pour traiter les données des capteurs IoT, analyser les données en temps réel et surveiller les opérations à la périphérie.

- [éditeur InfluxDB](#)

Ce composant transmet la télémétrie de l'état AWS IoT Greengrass du système depuis le plugin [Nucleus Emitter](#) vers InfluxDB. Ce composant peut également transmettre la télémétrie personnalisée à InfluxDB.

- [Framework PubSub pour l'IoT](#)

Ce framework fournit une architecture d'application, un code modèle et des exemples déployables qui aident à améliorer la qualité du code pour les applications pubsub IoT distribuées pilotées par des événements à l'AWS IoT Greengrass aide de composants personnalisés v2. Pour de plus amples informations, veuillez consulter [Création de AWS IoT Greengrass composants](#).

- [Jupyter Labs](#)

Ce composant est déployé JupyterLab sur un périphérique AWS IoT Greengrass principal. L'environnement Jupyter a accès aux ressources variables de processus et d'environnement définies par AWS IoT Greengrass, ce qui simplifie le processus de test et de développement de composants écrits en Python.

- [Serveur Web local](#)

Ce composant vous permet de créer une interface utilisateur Web locale sur un appareil principal de Greengrass. Vous pouvez créer une interface utilisateur Web locale qui vous permet de configurer les paramètres de l'appareil et de l'application ou de surveiller le périphérique, par exemple.

- [LoRaWaAdaptateur de protocole N](#)

Ce composant ingère les données des périphériques sans fil locaux qui utilisent le protocole LoRaWa N, qui est un protocole LPWAN (Low Power Wide Area Network). Le composant vous permet d'analyser les données et d'agir sur celles-ci localement sans communiquer avec le cloud.

- [Modbus TCP](#)

Ce composant collecte les données des appareils locaux à l'aide du protocole ModbusTCP et les publie dans des flux de données sélectionnés.

- [Nœud rouge](#)

Ce composant installe Node-RED sur un périphérique AWS IoT Greengrass principal à l'aide de NPM. Le composant dépend du composant [Node-RED Auth](#) qui doit être explicitement déployé et configuré. Vous pouvez utiliser la [CLI Node-RED pour Greengrass pour déployer des flux](#) Node-RED sur des appareils. AWS IoT Greengrass

- [Docker Node-Red](#)

Ce composant installe Node-RED sur le périphérique AWS IoT Greengrass principal à l'aide du conteneur Docker officiel de Node-RED. Le composant dépend du composant [Node-RED Auth](#) qui doit être explicitement déployé et configuré. Vous pouvez utiliser la [CLI Node-RED pour Greengrass pour déployer des flux](#) Node-RED sur des appareils. AWS IoT Greengrass

- [Authentification Node-Red](#)

Ce composant configure un nom d'utilisateur et un mot de passe pour sécuriser l'instance Node-RED exécutée sur un AWS IoT Greengrass périphérique principal.

- [OpenThreadRouteur frontalier](#)

Ce composant déploie le conteneur Docker OpenThread Border Router. Le composant permet de composer un périphérique Matter qui inclut un routeur Thread border.

- [Connecteur de données de streaming OSI Pi](#)

Ce composant permet de diffuser en temps réel l'ingestion de données depuis OSI Pi Data Archive vers une architecture de données moderne. AWS II s'intègre à OSI Pi Asset Framework qui est géré de manière centralisée via la AWS IoT PubSub messagerie.

- [Fournisseur de Parsec](#)

Ce composant permet aux AWS IoT Greengrass appareils d'intégrer des solutions de sécurité matérielles à l'aide du projet open source [Parsec](#) de la [Cloud Native Computing Foundation \(CNCF\)](#).

- [Base de données PostgreSQL](#)

Ce composant prend en charge la base de données relationnelle [PostgreSQL](#) en périphérie. Les clients peuvent utiliser ce composant pour approvisionner et gérer une instance PostgreSQL locale dans un conteneur docker.

- [Téléchargeur de fichiers S3](#)

Ce composant surveille un répertoire à la recherche de nouveaux fichiers, les télécharge sur Amazon Simple Storage Service (Amazon S3), puis les supprime après un chargement réussi.

- [Client Secrets Manager](#)

Ce composant fournit un outil CLI qui peut être utilisé par d'autres composants ayant besoin de récupérer les secrets du composant Secrets Manager dans un script de cycle de vie de recette.

- [Acheminement TES vers le conteneur](#)

Ce composant configure nftables ou iptables sur un AWS IoT Greengrass appareil afin qu'il puisse utiliser le [Service d'échange de jetons](#) composant avec des conteneurs.

- [WebRTC](#)

Ce composant ingère les flux audio et vidéo des caméras RTSP connectées au périphérique AWS IoT Greengrass principal. Ensuite, le composant transforme les flux audio et vidéo en peer-to-peer communication ou en relais via Amazon Kinesis Video Streams.

Pour demander une fonctionnalité ou signaler un bogue, ouvrez un GitHub problème dans le référentiel correspondant à ce composant. AWS ne fournit pas de support pour les composants communautaires. Pour de plus amples informations, veuillez consulter le `.CONTRIBUTING.md` fichier dans le référentiel de chaque composant.

Plusieurs composants AWS fournis sont également open source. Pour de plus amples informations, veuillez consulter [Logiciel de AWS IoT Greengrass base open source](#).

AWS IoT Greengrass outils de développement

Utilisez les outils de AWS IoT Greengrass développement pour créer, tester, créer, publier et déployer des composants Greengrass personnalisés.

- [Kit de développement Greengrass CLI](#)

Utilisez l'interface de ligne de commande du kit de AWS IoT Greengrass développement (CLI GDK) dans votre environnement de développement local pour créer des composants à partir de modèles et de composants communautaires du catalogue de logiciels [Greengrass](#). Vous pouvez utiliser la CLI GDK pour créer le composant et le publier sur le AWS IoT Greengrass service en tant que composant privé dans votre Compte AWS.

- [Interface de ligne de commande Greengrass](#)

Utilisez l'interface de ligne de commande Greengrass (Greengrass CLI) sur les appareils principaux de Greengrass pour déployer et déboguer les composants de Greengrass. La CLI Greengrass est un composant que vous pouvez déployer sur vos appareils principaux pour créer des déploiements locaux, afficher les détails des composants installés et explorer les fichiers journaux.

- [Console de débogage locale](#)

Utilisez la console de débogage locale sur les appareils principaux de Greengrass pour déployer et déboguer les composants de Greengrass à l'aide d'une interface Web de tableau de bord local. La console de débogage locale est un composant que vous pouvez déployer sur vos appareils principaux pour créer des déploiements locaux et afficher les détails des composants installés.

AWS IoT Greengrass fournit également les éléments suivants SDKs que vous pouvez utiliser dans les composants Greengrass personnalisés :

- Le Kit SDK des appareils AWS IoT, qui contient la bibliothèque de communication interprocessus (IPC). Pour de plus amples informations, veuillez consulter [Utilisez le Kit SDK des appareils AWS IoT pour communiquer avec le noyau de Greengrass, les autres composants et AWS IoT Core](#).

- Le SDK Stream Manager, que vous pouvez utiliser pour transférer des flux de données vers le AWS Cloud. Pour de plus amples informations, veuillez consulter [Gérez les flux de données sur les appareils principaux de Greengrass](#).

Rubriques

- [AWS IoT Greengrass Interface de ligne de commande du kit de développement](#)
- [Interface de ligne de commande Greengrass](#)
- [Utiliser le framework AWS IoT Greengrass de test](#)

AWS IoT Greengrass Interface de ligne de commande du kit de développement

L'interface de ligne de commande du kit de AWS IoT Greengrass développement (GDK CLI) fournit des fonctionnalités qui vous aident à développer des composants [Greengrass personnalisés](#). Vous pouvez utiliser la CLI GDK pour créer, créer et publier des composants personnalisés. Lorsque vous créez un référentiel de composants avec la CLI GDK, vous pouvez partir d'un modèle ou d'un composant communautaire du catalogue de logiciels [Greengrass](#). Ensuite, vous pouvez choisir un système de compilation qui empaquète les fichiers sous forme d'archives ZIP, utilise un script de génération Maven ou Gradle ou exécute une commande de construction personnalisée. Après avoir créé un composant, vous pouvez utiliser la CLI GDK pour le publier sur le AWS IoT Greengrass service. Vous pouvez ainsi utiliser la AWS IoT Greengrass console ou l'API pour déployer le composant sur vos appareils principaux Greengrass.

Lorsque vous développez des composants Greengrass sans la CLI GDK, vous devez mettre à jour la version et l'URLs artefact dans [le fichier de recette du composant](#) chaque fois que vous créez une nouvelle version du composant. Lorsque vous utilisez la CLI GDK, elle peut automatiquement mettre à jour la version et l'artefact URLs pour vous chaque fois que vous publiez une nouvelle version du composant.

La CLI GDK est open source et disponible sur GitHub. Vous pouvez personnaliser et étendre la CLI GDK pour répondre à vos besoins de développement de composants. Nous vous invitons à ouvrir des problèmes et à extraire des requêtes sur le GitHub référentiel. Vous pouvez trouver la source de la CLI GDK sur le lien suivant : <https://github.com/aws-greengrass/aws-greengrass-gdk-cli>

Prérequis

Pour installer et utiliser la CLI du Greengrass Development Kit, vous avez besoin des éléments suivants :

- Un Compte AWS. Si vous n'en avez pas, veuillez consulter [Configurez un Compte AWS](#).
- Un ordinateur de développement de type Windows, macOS ou Unix doté d'une connexion Internet.
- Pour la version 1.1.0 ou ultérieure de GDK CLI, [Python](#) 3.6 ou version ultérieure est installé sur votre ordinateur de développement.

Pour la version 1.0.0 de la CLI GDK, [Python](#) 3.8 ou version ultérieure installé sur votre ordinateur de développement.

- [Git](#) est installé sur votre ordinateur de développement.
- AWS Command Line Interface (AWS CLI) installé et configuré avec des informations d'identification sur votre ordinateur de développement. Pour plus d'informations, consultez les [sections Installation, mise à jour et désinstallation du AWS CLI](#) et [Configuration du AWS CLI dans le](#) guide de l'AWS Command Line Interface utilisateur.

Note

Si vous utilisez un Raspberry Pi ou un autre appareil ARM 32 bits, installez la AWS CLI V1. AWS CLI La V2 n'est pas disponible pour les appareils ARM 32 bits. Pour plus d'informations, voir [Installation, mise à jour et désinstallation de la AWS CLI version 1](#).

- Pour utiliser la CLI GDK afin de publier des composants sur le AWS IoT Greengrass service, vous devez disposer des autorisations suivantes :
 - `s3:CreateBucket`
 - `s3:GetBucketLocation`
 - `s3:PutObject`
 - `greengrass:CreateComponentVersion`
 - `greengrass:ListComponentVersions`
- Pour utiliser la CLI GDK afin de créer un composant dont les artefacts existent dans un compartiment S3 et non dans le système de fichiers local, vous devez disposer des autorisations suivantes :
 - `s3:ListBucket`

Cette fonctionnalité est disponible pour GDK CLI v1.1.0 et versions ultérieures.

Journal des modifications

Le tableau suivant décrit les modifications apportées à chaque version de la CLI GDK. Pour plus d'informations, consultez la [page des versions de la CLI GDK](#) sur GitHub.

Version	Modifications
1.6.2	Corrections de bogues et améliorations • Résout un problème selon lequel Windows gradlew.bat ne fonctionne pas en raison du chemin relatif. • Améliorations mineures apportées à la journalisation, aux tests et à l'empaquetage.
1.6.1	Corrections de bogues et améliorations • Ajoute un correctif de sécurité pour l'analyse des arguments de la CLI. • Permet au GDK d'obtenir le nom de la dernière version de Greengrass Testing Framework (GTF) comme version GTF par défaut. • Permet à GDK de recommander aux clients utilisant une ancienne version de GTF de passer à la dernière version.
1.6.0	Nouvelles fonctionnalités • Ajoute un contrôle de validation de recette par rapport au schéma de recette Greengrass pendant les commandes <code>component build</code> et <code>component publish</code>. Cette mise à jour aide les développeurs à identifier les problèmes réalisables dans leurs recettes de composants plus tôt dans le processus de création des composants. • Ajoute au modèle une suite de tests de confiance qui peut être extraite vers le bas par la <code>test-e2e init</code> commande. Cette suite de tests de confiance comprend huit tests génériques qui peuvent être utilisés et étendus pour répondre aux besoins de test des composants de base.

Version	Modifications
	Corrections de bogues et améliorations • Met à jour la version par défaut de Greengrass Testing Framework (GTF) utilisée par la <code>test-e2e</code> commande vers la version 1.2.0.
1.5.0	Corrections de bogues et améliorations Met à jour les modèles reconnus par l'option de <code>excludes</code> construction quand <code>build_system</code> c'est le <code>caszip</code>. Cette version reconnaîtra désormais les modèles globulaires qui correspondent aux noms de chemin en fonction de leurs caractères génériques. Cela permet de spécifier de manière personnalisée les répertoires à exclure.
1.4.0	Nouvelles fonctionnalités • Ajoute une nouvelle <code>config</code> commande qui lance une invite interactive pour modifier les champs d'un fichier de configuration GDK existant. • Modifie les <code>gdk component publish</code> commandes <code>gdk component build</code> et pour vérifier que la taille de la recette est conforme aux exigences de Greengrass ($\leq 16\,000$ octets) avant de continuer. Corrections de bogues et améliorations • Ajoute une journalisation supplémentaire à la sortie de la <code>gdk component build</code> commande lorsqu'une erreur de syntaxe de recette empêche la fin de la compilation pour prise de conscience. • Renomme <code>gtf-version</code> respectivement le <code>otf-options</code> et <code>otf-version</code> le, en raison du changement du nom d'Open Test Framework en Greengrass Testing Framework. <code>gtf-options</code>

Version	Modifications
1.3.0	Nouvelles fonctionnalités • Ajoute une nouvelle <code>test-e2e</code> commande pour prendre en charge le end-to-end test des composants à l'aide d'Open Test Framework. • Ajoute une nouvelle option de configuration <code>zip_name</code>, pour prendre en charge les noms de fichiers zip configurables avec le système de compilation zip. • Rend la <code>region</code> propriété dans le fichier de configuration GDK facultative. Corrections de bogues et améliorations • Résout un problème de création d'un nouveau répertoire même lorsque le modèle ou le référentiel spécifié n'existe pas lors de l'initialisation d'un projet GDK avec l'<code>--name</code> argument.
1.2.3	Corrections de bogues et améliorations • Résout un problème d'échec de la création d'un compartiment en raison d'une mauvaise gestion des erreurs. • Résout un problème de suppression des structures de liste dans la recette du composant.
1.2.2	Corrections de bogues et améliorations • Les touches de recette ne distinguent plus les majuscules et minuscules. • Ajoute une vérification pour déterminer si un compartiment existe dans un Région AWS et est accessible par l'utilisateur avant de créer un nouveau compartiment. Nécessite que l'utilisateur dispose de <code>GetBucketLocation</code> cette autorisation. • Résout un problème lié au <code>excludes</code> mot clé dans le fichier de configuration de la CLI GDK.

Version	Modifications
1.2.1	Corrections de bogues et améliorations • Accepte l'entrée de configuration Canada (Centralca-central-1) () Région AWS dans la région du gdk-config.json fichier. • Résout les problèmes liés à l'argument --region GDK CLI de la publish commande.
1.2.0	Nouvelles fonctionnalités • Ajoute l'optionentrée à la build configuration dans le fichier de configuration de la CLI GDK. Supporte excludes under options pour exclure certains fichiers de l'artefact zip lors de l'utilisation du système de zip compilation. • Ajoute le système de gradlew génération pour utiliser Gradle Wrapper pour créer des composants. • Ajoute la prise en charge des fichiers de construction Kotlin DSL pour l'option de gradle construction. • Ajoute une options entrée à la publish configuration dans le fichier de configuration de la CLI GDK. Supporte le file_upload_args under options pour fournir des arguments supplémentaires lors du téléchargement de fichiers sur Amazon S3. Corrections de bogues et améliorations • Résout un problème selon lequel les versions de Gradle n'étaient pas nettoyées avant d'exécuter une commande de génération. • Résout un problème en raison duquel la compilation ne se terminait pas lorsque la commande de construction échouait. • Améliore le format de sortie de la gdk component list commande.

Version	Modifications
1.1.0	Nouvelles fonctionnalités • Ajoute le support pour le système de construction Gradle. • Ajoute la prise en charge du système de compilation Maven sur les appareils Windows. • Ajoute l'--bucket argument à la commande de publication du composant. Vous pouvez utiliser cet argument pour spécifier le compartiment exact dans lequel la CLI GDK télécharge les artefacts des composants. • Ajoute l'--name argument à la commande d'initialisation du composant. Vous pouvez utiliser cette option pour spécifier le dossier dans lequel la CLI GDK initialise le composant. • Ajoute la prise en charge des artefacts de composants qui existent dans un compartiment S3 mais pas dans le dossier de construction du composant local. Vous pouvez utiliser cette fonctionnalité pour réduire les coûts de bande passante pour les artefacts de composants volumineux, tels que les modèles d'apprentissage automatique. Corrections de bogues et améliorations • Met à jour la commande de publication du composant pour vérifier si le composant est créé avant de le publier. Si le composant n'est pas créé, cette commande le crée désormais pour vous. • Résout un problème en raison duquel le système de génération zip ne parvient pas à générer sur les appareils Windows lorsque le nom du fichier ZIP contient des majuscules. • Améliore le format des messages de journal et remplace le niveau de journal par défaut INFO sur les appareils qui exécutent des versions de Python antérieures à 3.8. • Modifie la version minimale requise pour Python en Python 3.6.
1.0.0	Première version.

Installation ou mise à jour de l'interface de ligne de commande AWS IoT Greengrass du kit de développement

L'interface de ligne de commande du kit de AWS IoT Greengrass développement (GDK CLI) est basée sur Python, vous pouvez donc pip l'installer sur votre ordinateur de développement.

Tip

Vous pouvez également installer la CLI GDK dans un environnement virtuel Python tel que [venv](#). Pour plus d'informations, consultez [Environnements virtuels et packages](#) dans la documentation de Python 3.

Pour installer ou mettre à jour la CLI GDK

1. Exécutez la commande suivante pour installer la dernière version de la CLI GDK à partir de son [GitHub référentiel](#).

```
python3 -m pip install -U git+https://github.com/aws-greengrass/aws-greengrass-gdk-cli.git@v1.6.2
```

Note

Pour installer une version spécifique de la CLI GDK, remplacez-la *versionTag* par la balise de version à installer. Vous pouvez consulter les balises de version de la CLI GDK dans son [GitHub référentiel](#).

```
python3 -m pip install -U git+https://github.com/aws-greengrass/aws-greengrass-gdk-cli.git@versionTag
```

2. Exécutez la commande suivante pour vérifier que la CLI GDK a été correctement installée.

```
gdk --help
```

Si la gdk commande n'est pas trouvée, ajoutez son dossier dans PATH.

- Sur les appareils Linux, ajoutez `/home/MyUser/.local/bin` à PATH et remplacez-le par *MyUser* le nom de votre utilisateur.

- Sur les appareils Windows, ajoutez *PythonPath*\\Scripts à PATH et remplacez-le par *PythonPath* le chemin d'accès au dossier Python de votre appareil.

Vous pouvez désormais utiliser la CLI GDK pour créer, créer et publier des composants Greengrass. Pour plus d'informations sur l'utilisation de la CLI GDK, consultez [AWS IoT Greengrass Commandes de l'interface de ligne de commande du kit de développement](#).

AWS IoT Greengrass Commandes de l'interface de ligne de commande du kit de développement

L'interface de ligne de commande du kit de AWS IoT Greengrass développement (GDK CLI) fournit une interface de ligne de commande que vous pouvez utiliser pour créer, créer et publier des composants Greengrass sur votre ordinateur de développement. Les commandes GDK CLI utilisent le format suivant.

```
gdk <command> <subcommand> [arguments]
```

Lorsque vous [installez la CLI GDK](#), le programme d'installation ajoute des éléments gdk au PATH afin que vous puissiez exécuter la CLI GDK depuis la ligne de commande.

Vous pouvez utiliser les arguments suivants avec n'importe quelle commande :

- Utilisez `-h` ou `--help` pour obtenir des informations sur une commande GDK CLI.
- Utilisez `-v` ou `--version` pour voir quelle version de GDK CLI est installée.
- Utilisez `-d` ou `--debug` pour générer des journaux détaillés que vous pouvez utiliser pour déboguer la CLI GDK.

Cette section décrit les commandes de la CLI GDK et fournit des exemples pour chaque commande. Le synopsis de chaque commande montre ses arguments et leur utilisation. Les arguments facultatifs sont indiqués entre crochets.

Commandes disponibles

- [composant](#)
- [config](#)
- [test-e2e](#)

composant

Utilisez la `component` commande dans l'interface de ligne de commande du kit de AWS IoT Greengrass développement (CLI GDK) pour créer, créer et publier des composants Greengrass personnalisés.

Sous-commandes

- [init](#)
- [build](#)
- [publication](#)
- [liste](#)

init

Initialisez un dossier de composants Greengrass à partir d'un modèle de composant ou d'un composant communautaire.

La CLI GDK récupère les composants communautaires du [catalogue des logiciels Greengrass](#) et les modèles de composants du référentiel de modèles de [AWS IoT Greengrass composants](#) sur. GitHub

Note

Si vous utilisez GDK CLI v1.0.0, vous devez exécuter cette commande dans un dossier vide. La CLI GDK télécharge le modèle ou le composant communautaire dans le dossier actuel. Si vous utilisez la CLI GDK v1.1.0 ou une version ultérieure, vous pouvez spécifier l' - -nameargument pour spécifier le dossier dans lequel la CLI GDK télécharge le modèle ou le composant communautaire. Si vous utilisez cet argument, spécifiez un dossier qui n'existe pas. La CLI GDK crée le dossier pour vous. Si vous ne spécifiez pas cet argument, la CLI GDK utilise le dossier actuel, qui doit être vide.

Si le composant utilise le [système de génération zip](#), la CLI GDK compresse certains fichiers du dossier du composant dans un fichier zip portant le même nom que le dossier du composant. Par exemple, si le nom du dossier du composant estHelloWorld, la CLI GDK crée un fichier zip nomméHelloWorld.zip. Dans la recette du composant, le nom de l'artefact zip doit correspondre au nom du dossier du composant. Si vous utilisez la version 1.0.0 de la CLI GDK sur un appareil Windows, les noms des dossiers de composants et des fichiers zip ne doivent contenir que des lettres minuscules.

Si vous initialisez un modèle ou un composant communautaire qui utilise le système de génération zip dans un dossier portant un nom différent de celui du modèle ou du composant,

vous devez modifier le nom de l'artefact zip dans la recette du composant. Mettez à jour les Lifecycle définitions Artifacts et de manière à ce que le nom du fichier zip corresponde au nom du dossier du composant. L'exemple suivant met en évidence le nom du fichier zip dans les Lifecycle définitions Artifacts et.

JSON

```
{
  ...
  "Manifests": [
    {
      "Platform": {
        "os": "all"
      },
      "Artifacts": [
        {
          "URI": "s3://BUCKET_NAME/COMPONENT_NAME/
COMPONENT_VERSION/HelloWorld.zip",
          "Unarchive": "ZIP"
        }
      ],
      "Lifecycle": {
        "Run": "python3 -u {artifacts:decompressedPath}/HelloWorld/main.py
{configuration:/Message}"
      }
    }
  ]
}
```

YAML

```
---
...
Manifests:
  - Platform:
      os: all
    Artifacts:
      - URI: "s3://BUCKET_NAME/COMPONENT_NAME/
COMPONENT_VERSION/HelloWorld.zip"
        Unarchive: ZIP
    Lifecycle:
```

```
Run: "python3 -u {artifacts:decompressedPath}/HelloWorld/main.py  
{configuration:/Message}"
```

Résumé

```
$ gdk component init  
  --language  
  --template  
  --repository  
  --name
```

Arguments (initialisation à partir du modèle de composant)

- `-l, --language` — Langage de programmation à utiliser pour le modèle que vous spécifiez.

Vous devez spécifier soit, `--repository` soit `--language` et `--template`.

- `-t, --template` — Modèle de composant à utiliser pour un projet de composant local. Pour afficher les modèles disponibles, utilisez la commande [list](#).

Vous devez spécifier soit, `--repository` soit `--language` et `--template`.

- `-n, --name` — (Facultatif) Le nom du dossier local dans lequel la CLI GDK initialise le composant. Spécifiez un dossier qui n'existe pas. La CLI GDK crée le dossier pour vous.

Cette fonctionnalité est disponible pour GDK CLI v1.1.0 et versions ultérieures.

Arguments (initialisation à partir du composant communautaire)

- `-r, --repository` — Le composant communautaire à récupérer dans le dossier local. Pour afficher les composants communautaires disponibles, utilisez la commande [list](#).

Vous devez spécifier soit, `--repository` soit `--language` et `--template`.

- `-n, --name` — (Facultatif) Le nom du dossier local dans lequel la CLI GDK initialise le composant. Spécifiez un dossier qui n'existe pas. La CLI GDK crée le dossier pour vous.

Cette fonctionnalité est disponible pour GDK CLI v1.1.0 et versions ultérieures.

Sortie

L'exemple suivant montre le résultat produit lorsque vous exécutez cette commande pour initialiser un dossier de composants à partir du modèle Python Hello World.

```
$ gdk component init -l python -t HelloWorld
[2021-11-29 12:51:40] INFO - Initializing the project directory with a python
component template - 'HelloWorld'.
[2021-11-29 12:51:40] INFO - Fetching the component template 'HelloWorld-python'
from Greengrass Software Catalog.
```

L'exemple suivant montre le résultat produit lorsque vous exécutez cette commande pour initialiser un dossier de composants à partir d'un composant communautaire.

```
$ gdk component init -r aws-greengrass-labs-database-influxdb
[2022-01-24 15:44:33] INFO - Initializing the project directory with a component
from repository catalog - 'aws-greengrass-labs-database-influxdb'.
[2022-01-24 15:44:33] INFO - Fetching the component repository 'aws-greengrass-labs-
database-influxdb' from Greengrass Software Catalog.
```

build

Intégrez la source d'un composant dans une recette et des artefacts que vous pouvez publier sur le AWS IoT Greengrass service. La CLI GDK exécute le système de génération que vous spécifiez dans le [fichier de configuration de la CLI GDK](#), `gdk-config.json`. Vous devez exécuter cette commande dans le dossier où se trouve le `gdk-config.json` fichier.

Lorsque vous exécutez cette commande, la CLI GDK crée une recette et des artefacts dans le `greengrass-build` dossier du dossier des composants. La CLI GDK enregistre la recette dans le `greengrass-build/recipes` dossier et enregistre les artefacts dans le `greengrass-build/artifacts/componentName/componentVersion` dossier.

Si vous utilisez GDK CLI v1.1.0 ou version ultérieure, la recette du composant peut spécifier des artefacts qui existent dans un compartiment S3 mais pas dans le dossier de construction du composant local. Vous pouvez utiliser cette fonctionnalité pour réduire l'utilisation de la bande passante lorsque vous développez des composants comportant de gros artefacts, tels que des modèles d'apprentissage automatique.

Après avoir créé un composant, vous pouvez effectuer l'une des opérations suivantes pour le tester sur un appareil Greengrass principal :

- Si vous développez sur un appareil différent de celui sur lequel vous exécutez le logiciel AWS IoT Greengrass Core, vous devez publier le composant pour le déployer sur un appareil principal Greengrass. Publiez le composant sur le AWS IoT Greengrass service et déployez-le sur le

périphérique principal de Greengrass. Pour plus d'informations, consultez la commande de [publication](#) et [Créer des déploiements](#).

- Si vous développez sur le même appareil que celui sur lequel vous exécutez le logiciel AWS IoT Greengrass Core, vous pouvez publier le composant sur le AWS IoT Greengrass service à déployer, ou vous pouvez créer un déploiement local pour installer et exécuter le composant. Pour créer un déploiement local, utilisez la CLI Greengrass. Pour plus d'informations, consultez [Interface de ligne de commande Greengrass](#) et [Testez AWS IoT Greengrass les composants avec des déploiements locaux](#). Lorsque vous créez le déploiement local, greengrass-build/recipes spécifiez-le comme dossier de recettes et greengrass-build/artifacts comme dossier d'artefacts.

Résumé

```
$ gdk component build
```

Arguments

Aucun

Sortie

L'exemple suivant montre le résultat produit lorsque vous exécutez cette commande.

```
$ gdk component build
[2021-11-29 13:18:49] INFO - Getting project configuration from gdk-config.json
[2021-11-29 13:18:49] INFO - Found component recipe file 'recipe.yaml' in the
project directory.
[2021-11-29 13:18:49] INFO - Building the component 'com.example.PythonHelloWorld'
with the given project configuration.
[2021-11-29 13:18:49] INFO - Using 'zip' build system to build the component.
[2021-11-29 13:18:49] WARNING - This component is identified as using 'zip' build
system. If this is incorrect, please exit and specify custom build command in the
'gdk-config.json'.
[2021-11-29 13:18:49] INFO - Zipping source code files of the component.
[2021-11-29 13:18:49] INFO - Copying over the build artifacts to the greengrass
component artifacts build folder.
[2021-11-29 13:18:49] INFO - Updating artifact URIs in the recipe.
[2021-11-29 13:18:49] INFO - Creating component recipe in 'C:\Users\MyUser\Documents
\greengrass-components\python\HelloWorld\greengrass-build\recipes'.
```

publication

Publiez ce composant sur le AWS IoT Greengrass service. Cette commande télécharge les artefacts de construction dans un compartiment S3, met à jour l'URI de l'artefact dans la recette et crée une nouvelle version du composant à partir de la recette. La CLI GDK utilise le compartiment S3 et la AWS région que vous spécifiez dans le [fichier de configuration de la CLI GDK](#), `gdk-config.json`. Vous devez exécuter cette commande dans le dossier où se trouve le `gdk-config.json` fichier.

Si vous utilisez la CLI GDK v1.1.0 ou une version ultérieure, vous pouvez spécifier l'--bucket argument pour spécifier le compartiment S3 dans lequel la CLI GDK télécharge les artefacts du composant. Si vous ne spécifiez pas cet argument, la CLI GDK télécharge vers le compartiment S3 dont le nom est `bucket-region-accountId`, où `bucket` et `region` sont les valeurs que vous spécifiez dans `gdk-config.json`, ainsi que votre `accountId` Compte AWS identifiant. La CLI GDK crée le bucket s'il n'existe pas.

Si vous utilisez la CLI GDK v1.2.0 ou une version ultérieure, vous pouvez remplacer ce qui est Région AWS spécifié dans le fichier de configuration de la CLI GDK à l'aide du paramètre. --region Vous pouvez également définir des options supplémentaires à l'aide du --options paramètre. Pour obtenir la liste des options disponibles, consultez [Fichier de configuration CLI du Greengrass Development Kit](#).

Lorsque vous exécutez cette commande, la CLI GDK publie le composant avec la version que vous spécifiez dans la recette. Si vous le spécifiez NEXT_PATCH, la CLI GDK utilise la prochaine version du correctif qui n'existe pas encore. Les versions sémantiques utilisent une majeure. mineur. système de numérotation des patches. Pour plus d'informations, consultez la [spécification de version sémantique](#).

Note

Si vous utilisez la CLI GDK v1.1.0 ou une version ultérieure, lorsque vous exécutez cette commande, la CLI GDK vérifie si le composant est créé. Si le composant n'est pas créé, la CLI GDK [le construit](#) avant de le publier.

Résumé

```
$ gdk component publish  
  [--bucket] [--region] [--options]
```

Arguments

- `-b, --bucket` — (Facultatif) Spécifiez le nom du compartiment S3 dans lequel la CLI GDK publie les artefacts des composants.

Si vous ne spécifiez pas cet argument, la CLI GDK télécharge vers le compartiment S3 dont le nom est `bucket-region-accountId`, où `bucket` et `region` sont les valeurs que vous spécifiez dans `gdk-config.json`, ainsi que votre `accountId` Compte AWS identifiant. La CLI GDK crée le bucket s'il n'existe pas.

La CLI GDK crée le bucket s'il n'existe pas.

Cette fonctionnalité est disponible pour GDK CLI v1.1.0 et versions ultérieures.

- `-r, --region` — (Facultatif) Spécifiez le nom du Région AWS lorsque le composant est créé. Cet argument remplace le nom de la région dans la configuration de la CLI GDK.

Cette fonctionnalité est disponible pour GDK CLI v1.2.0 et versions ultérieures.

- `-o, --options` (Facultatif) Spécifiez une liste d'options pour publier un composant. L'argument doit être une chaîne JSON valide ou un chemin de fichier vers un fichier JSON contenant les options de publication. Cet argument remplace les options de la configuration de la CLI GDK.

Cette fonctionnalité est disponible pour GDK CLI v1.2.0 et versions ultérieures.

Sortie

L'exemple suivant montre le résultat produit lorsque vous exécutez cette commande.

```
$ gdk component publish
[2021-11-29 13:45:29] INFO - Getting project configuration from gdk-config.json
[2021-11-29 13:45:29] INFO - Found component recipe file 'recipe.yaml' in the
project directory.
[2021-11-29 13:45:29] INFO - Found credentials in shared credentials file: ~/.aws/
credentials
[2021-11-29 13:45:30] INFO - Publishing the component 'com.example.PythonHelloWorld'
with the given project configuration.
[2021-11-29 13:45:30] INFO - No private version of the component
'com.example.PythonHelloWorld' exist in the account. Using '1.0.0' as the next
version to create.
[2021-11-29 13:45:30] INFO - Uploading the component built artifacts to s3 bucket.
[2021-11-29 13:45:30] INFO - Uploading component artifacts to S3 bucket: {bucket}.
If this is your first time using this bucket, add the 's3:GetObject' permission
to each core device's token exchange role to allow it to download the component
```

```
artifacts. For more information, see https://docs.aws.amazon.com/greengrass/v2/
developer/guide/device-service-role.html.
[2021-11-29 13:45:30] INFO - Not creating an artifacts bucket as it already exists.
[2021-11-29 13:45:30] INFO - Updating the component recipe
com.example.PythonHelloWorld-1.0.0.
[2021-11-29 13:45:30] INFO - Creating a new greengrass component
com.example.PythonHelloWorld-1.0.0
[2021-11-29 13:45:30] INFO - Created private version '1.0.0' of the component in the
account. 'com.example.PythonHelloWorld'.
```

liste

Récupérez la liste des modèles de composants et des composants communautaires disponibles.

La CLI GDK récupère les composants communautaires du [catalogue des logiciels Greengrass](#) et les modèles de composants du référentiel de modèles de [AWS IoT Greengrass composants](#) sur GitHub.

Vous pouvez transmettre le résultat de cette commande à la commande [init](#) pour initialiser les référentiels de composants à partir de modèles et de composants communautaires.

Résumé

```
$ gdk component list
  [--template]
  [--repository]
```

Arguments

- **-t, --template** — (Facultatif) Spécifiez cet argument pour répertorier les modèles de composants disponibles. Cette commande affiche le nom et la langue de chaque modèle au format *name - language*. Par exemple, dans HelloWorld-python, le nom du modèle est HelloWorld et la langue est python.
- **-r, --repository** — (Facultatif) Spécifiez cet argument pour répertorier les référentiels de composants communautaires disponibles.

Sortie

L'exemple suivant montre le résultat produit lorsque vous exécutez cette commande.

```
$ gdk component list --template
[2021-11-29 12:29:04] INFO - Listing all the available component templates from
Greengrass Software Catalog.
```



```
[2021-11-29 12:29:04] INFO - Found '2' component templates to display.  
1. HelloWorld-python  
2. HelloWorld-java
```

config

Utilisez la `config` commande de l'interface de ligne de commande du kit de AWS IoT Greengrass développement (CLI GDK) pour modifier la configuration du GDK dans le fichier de configuration, `gdk-config.json`

Sous-commandes

- [mise à jour](#)

mise à jour

Lancez une invite interactive pour modifier les champs d'un fichier de configuration GDK existant.

Résumé

```
$ gdk config update  
[--component]
```

Arguments

- `-c, --component` — Pour mettre à jour les champs relatifs aux composants dans le `gdk-config.json` fichier. Cet argument est obligatoire car il s'agit de la seule option.

Sortie

L'exemple suivant montre le résultat produit lorsque vous exécutez cette commande pour configurer un composant.

```
$ gdk config update --component  
Current value of the REQUIRED component_name is (default:  
  com.example.PythonHelloWorld):  
Current value of the REQUIRED author is (default: author):  
Current value of the REQUIRED version is (default: NEXT_PATCH):  
Do you want to change the build configurations? (y/n)  
Do you want to change the publish configurations? (y/n)  
[2023-09-26 10:19:48] INFO - Config file has been updated. Exiting...
```

test-e2e

Utilisez la `test-e2e` commande de l'interface de ligne de commande du kit de AWS IoT Greengrass développement (CLI GDK) pour initialiser, créer et end-to-end exécuter des modules de test dans le projet GDK.

Sous-commandes

- [init](#)
- [build](#)
- [run](#)

init

Initialisez un projet GDK CLI existant avec un module de test qui utilise Greengrass Testing Framework (GTF).

Par défaut, la CLI GDK récupère le modèle de module maven depuis le référentiel de [modèles de AWS IoT Greengrass composants](#) sur. GitHub Ce module maven est livré avec une dépendance au fichier `aws-greengrass-testing-standalone` JAR.

Cette commande crée un nouveau répertoire appelé `gg-e2e-tests` inside of the GDK project. Si le répertoire du module de test existe déjà et qu'il n'est pas vide, la commande s'arrête sans rien faire. Ce `gg-e2e-tests` dossier contient la fonctionnalité Cucumber et les définitions d'étapes structurées dans un projet Maven.

Par défaut, cette commande essaie d'utiliser la dernière version de GTF.

Résumé

```
$ gdk test-e2e init  
  [--gtf-version]
```

Arguments

- `-ov, --gtf-version` — (Facultatif) Version du GTF à utiliser avec le module de end-to-end test dans le projet GDK. Cette valeur doit être l'une des versions GTF des [versions](#). Cet argument remplace celui de la configuration `gtf_version` de la CLI GDK.

Sortie

L'exemple suivant montre le résultat produit lorsque vous exécutez cette commande pour initialiser le projet GDK avec le module de test.

```
$ gdk test-e2e init
[2023-12-06 12:20:28] INFO - Using the GTF version provided in the GDK test config
1.2.0
[2023-12-06 12:20:28] INFO - Downloading the E2E testing template from GitHub into
gg-e2e-tests directory...
```

build

Note

Vous devez créer le composant en l'exécutant `gdk component build` avant de créer le module de end-to-end test.

Créez le module end-to-end de test. La CLI GDK crée le module de test à l'aide du système de génération que vous spécifiez dans le [fichier de configuration de la CLI GDK](#) `gdk-config.json`, sous la `test-e2e` propriété. Vous devez exécuter cette commande dans le dossier où se trouve le `gdk-config.json` fichier.

Par défaut, la CLI GDK utilise le système de construction maven pour créer le module de test. [Maven](#) est requis pour exécuter la `gdk test-e2e build` commande.

Vous devez créer le composant en l'exécutant `gdk-component-build` avant de créer le module de test, si les fichiers de fonctionnalités de test contiennent des variables telles que `GDK_COMPONENT_NAME` et `GDK_COMPONENT_RECIPE_FILE` à interpoler.

Lorsque vous exécutez cette commande, la CLI GDK interpole toutes les variables de la configuration du projet GDK et crée le `gg-e2e-tests` module pour générer le fichier JAR de test final.

Résumé

```
$ gdk test-e2e build
```

Arguments

Aucun

Sortie

L'exemple suivant montre le résultat produit lorsque vous exécutez cette commande.

```
$ gdk test-e2e build
[2023-07-20 15:36:48] INFO - Updating feature file: file:///path/to//
HelloWorld/greengrass-build/gg-e2e-tests/src/main/resources/greengrass/features/
component.feature
[2023-07-20 15:36:48] INFO - Creating the E2E testing recipe file:///path/to/
HelloWorld/greengrass-build/recipes/e2e_test_recipe.yaml
[2023-07-20 15:36:48] INFO - Building the E2E testing module
[2023-07-20 15:36:48] INFO - Running the build command 'mvn package'
.....
```

run

Exécutez le module de test avec les options de test du fichier de configuration GDK.

Note

Vous devez créer le module de test en l'exécutant `gdk test-e2e build` avant d'exécuter les end-to-end tests.

Résumé

```
$ gdk test-e2e run
  [--gtf-options]
```

Arguments

- `-oo, --gtf-options` — (Facultatif) Spécifiez une liste d'options pour exécuter les end-to-end tests. L'argument doit être une chaîne JSON valide ou un chemin de fichier vers un fichier JSON contenant les options GTF. Les options fournies dans le fichier de configuration sont fusionnées avec celles fournies dans les arguments de commande. Si une option est présente aux deux endroits, celle en argument a priorité sur celle du fichier de configuration.

Si l'option `tags` n'est pas spécifiée dans cette commande, GDK l'utilise `Sample` pour les balises. Si `ggc-archive` ce n'est pas spécifié, GDK télécharge la dernière version de l'archive Greengrass nucleus.

Sortie

L'exemple suivant montre le résultat produit lorsque vous exécutez cette commande.

```
$ gdk test-e2e run
[2023-07-20 16:35:53] INFO - Downloading latest nucleus archive from url https://
d2s8p88vqu9w66.cloudfront.net/releases/greengrass-latest.zip
[2023-07-20 16:35:57] INFO - Running test jar with command java -jar /path/to/
greengrass-build/gg-e2e-tests/target/uat-features-1.0.0.jar --ggc-archive=/path/to/
aws-greengrass-gdk-cli/HelloWorld/greengrass-build/greengrass-nucleus-latest.zip --
tags=Sample

16:35:59.693 [] [] [] [INFO]
  com.aws.greengrass.testing.modules.GreengrassContextModule - Extracting /path/
to/workplace/aws-greengrass-gdk-cli/HelloWorld/greengrass-build/greengrass-
nucleus-latest.zip into /var/folders/7g/ltzcb_3s77nbtmkzfb6brwv40000gr/T/gg-
testing-7718418114158172636/greengrass
16:36:00.534 [gtf-1.1.0-SNAPSHOT] [] [] [INFO]
  com.aws.greengrass.testing.features.LoggerSteps - GTF Version is gtf-1.1.0-SNAPSHOT
.....
```

Fichier de configuration CLI du Greengrass Development Kit

L'interface de ligne de commande du kit de AWS IoT Greengrass développement (CLI GDK) lit un fichier de configuration `gdk-config.json` nommé pour créer et publier des composants. Ce fichier de configuration doit exister à la racine du référentiel de composants. Vous pouvez utiliser la [commande d'initialisation](#) de la CLI GDK pour initialiser les référentiels de composants avec ce fichier de configuration.

Rubriques

- [Format de fichier de configuration GDK CLI](#)
- [Exemples de fichiers de configuration GDK CLI](#)

Format de fichier de configuration GDK CLI

Lorsque vous définissez un fichier de configuration GDK CLI pour un composant, vous spécifiez les informations suivantes au format JSON.

`gdk_version`

Version minimale de la CLI GDK compatible avec ce composant. Cette valeur doit être l'une des versions de la CLI GDK des [versions](#).

`component`

Configuration de ce composant.

componentName

`author`

L'auteur ou l'éditeur du composant.

`version`

Version du composant. Spécifiez l'un des éléments suivants :

- **NEXT_PATCH**— Lorsque vous choisissez cette option, la CLI GDK définit la version lorsque vous publiez le composant. La CLI GDK interroge le AWS IoT Greengrass service pour identifier la dernière version publiée du composant. Ensuite, il définit la version sur la version de correctif suivante après cette version. Si vous n'avez jamais publié le composant auparavant, la CLI GDK utilise la version `1.0.0`.

Si vous choisissez cette option, vous ne pouvez pas utiliser la [CLI Greengrass](#) pour déployer et tester localement le composant sur votre ordinateur de développement local qui exécute le logiciel AWS IoT Greengrass Core. Pour activer les déploiements locaux, vous devez plutôt spécifier une version sémantique.

- Une version sémantique, telle que `1.0.0`. Les versions sémantiques utilisent une majeure. mineur. système de numérotation des patchs. Pour plus d'informations, consultez la [spécification de version sémantique](#).

Si vous développez des composants sur un appareil principal Greengrass sur lequel vous souhaitez les déployer et les tester, choisissez cette option. Vous devez créer le composant avec une version spécifique pour créer des déploiements locaux avec la CLI [Greengrass](#).

build

Configuration à utiliser pour intégrer la source de ce composant en artefacts. Cet objet contient les informations suivantes :

`build_system`

Le système de compilation à utiliser. Sélectionnez parmi les options suivantes :

- `zip`— Regroupe le dossier du composant dans un fichier ZIP à définir comme seul artefact du composant. Choisissez cette option pour les types de composants suivants :
 - Composants utilisant des langages de programmation interprétés, tels que Python ou JavaScript.
 - Composants qui empaquetent des fichiers autres que du code, tels que des modèles d'apprentissage automatique ou d'autres ressources.

La CLI GDK compresse le dossier du composant dans un fichier zip portant le même nom que le dossier du composant. Par exemple, si le nom du dossier du composant est `HelloWorld`, la CLI GDK crée un fichier zip nommé `HelloWorld.zip`.

Note

Si vous utilisez la version 1.0.0 de la CLI GDK sur un appareil Windows, les noms des dossiers de composants et des fichiers zip ne doivent contenir que des lettres minuscules.

Lorsque la CLI GDK comprime le dossier du composant dans un fichier zip, elle ignore les fichiers suivants :

- le fichier `gdk-config.json` ;
- Le fichier de recette (`recipe.json` ou `recipe.yaml`)
- Créez des dossiers, tels que `greengrass-build`
- `maven`— Exécute la `mvn clean package` commande pour intégrer la source du composant en artefacts. Choisissez cette option pour les composants qui utilisent [Maven](#), tels que les composants Java.

Sur les appareils Windows, cette fonctionnalité est disponible pour GDK CLI v1.1.0 et versions ultérieures.

- **gradle**— Exécute la `gradle build` commande pour intégrer la source du composant en artefacts. Choisissez cette option pour les composants qui utilisent [Gradle](#). Cette fonctionnalité est disponible pour GDK CLI v1.1.0 et versions ultérieures.

Le système de `gradle` construction prend en charge Kotlin DSL en tant que fichier de construction. Cette fonctionnalité est disponible pour GDK CLI v1.2.0 et versions ultérieures.

- **gradlew**— Exécute la `gradlew` commande pour intégrer la source du composant en artefacts. Choisissez cette option pour les composants qui utilisent le [Gradle Wrapper](#).

Cette fonctionnalité est disponible pour GDK CLI v1.2.0 et versions ultérieures.

- **custom**— Exécute une commande personnalisée pour intégrer la source du composant dans une recette et des artefacts. Spécifiez la commande personnalisée dans le `custom_build_command` paramètre.

`custom_build_command`

(Facultatif) La commande de construction personnalisée à exécuter pour un système de génération personnalisé. Vous devez spécifier ce paramètre si vous spécifiez `custom` pour `build_system`.

 Important

Cette commande doit créer une recette et des artefacts dans les dossiers suivants du dossier des composants. La CLI GDK crée ces dossiers pour vous lorsque vous exécutez la [commande component build](#).

- Dossier de recettes : `greengrass-build/recipes`
- Dossier Artefacts : `greengrass-build/artifacts/componentName/componentVersion`

Remplacez *componentName* par le nom du composant et remplacez *componentVersion* par la version du composant ou `NEXT_PATCH`.

Vous pouvez spécifier une chaîne unique ou une liste de chaînes, chaque chaîne étant un mot dans la commande. Par exemple, pour exécuter une commande de

génération personnalisée pour un composant C++, vous pouvez spécifier **cmake --build build --config Release** ou **["cmake", "--build", "build", "--config", "Release"]**.

Pour voir un exemple de système de construction personnalisé, consultez le [aws.greengrass.labs.LocalWebServer community component sur GitHub](https://aws.github.io/greengrass-labs-local-web-server-community-component/).

options

(Facultatif) Options de configuration supplémentaires utilisées pendant le processus de création du composant.

Cette fonctionnalité est disponible pour GDK CLI v1.2.0 et versions ultérieures.

excludes

Liste de modèles globaux qui définissent les fichiers à exclure du répertoire des composants lors de la création du fichier zip. Valable uniquement lorsque `build_system` c'est le caszip.

Note

Dans les versions 1.4.0 et antérieures de la CLI GDK, tout fichier correspondant à une entrée de la liste des exclusions est exclu de tous les sous-répertoires du composant. Pour obtenir le même comportement dans les versions 1.5.0 et ultérieures de la CLI GDK, ajoutez les `**/` entrées existantes dans la liste des exclusions. Par exemple, `*.txt` exclura les fichiers texte du répertoire uniquement ; `**/*.txt` exclura les fichiers texte de tous les répertoires et sous-répertoires.

Dans les versions 1.5.0 et ultérieures de la CLI GDK, vous pouvez voir un avertissement lors de la création du composant lorsque `excludes` est défini dans le fichier de configuration GDK. Pour désactiver cet avertissement, définissez la variable `GDK_EXCLUDES_WARN_IGNORE` d'environnement sur `true`.

La CLI GDK exclut toujours les fichiers suivants du fichier zip :

- le fichier `gdk-config.json` ;
- Le fichier de recette (`recipe.json` ou `recipe.yaml`)

- Créez des dossiers, tels que `greengrass-build`

Les fichiers suivants sont exclus par défaut. Cependant, vous pouvez contrôler lesquels de ces fichiers sont exclus à l'aide de `excludes` cette option.

- Tout dossier commençant par le préfixe « test » (`() test*`)
- Tous les fichiers cachés
- Le dossier `node_modules`

Si vous spécifiez l'`excludes` option, la CLI GDK exclut uniquement les fichiers que vous avez définis avec l'`excludes` option. Si vous ne spécifiez pas `excludes` cette option, la CLI GDK exclut les fichiers et dossiers par défaut indiqués précédemment.

`zip_name`

Le nom du fichier zip à utiliser lorsque vous créez un artefact zip pendant le processus de génération. Valable uniquement lorsque `build_system` c'est le `caszip`. Si le `build_system` champ est vide, le nom du composant est utilisé pour le nom du fichier zip.

`publish`

Configuration à utiliser pour publier ce composant sur le AWS IoT Greengrass service.

Si vous utilisez la CLI GDK v1.1.0 ou une version ultérieure, vous pouvez spécifier l' `--bucket` argument pour spécifier le compartiment S3 dans lequel la CLI GDK télécharge les artefacts du composant. Si vous ne spécifiez pas cet argument, la CLI GDK télécharge vers le compartiment S3 dont le nom est `bucket-region-accountId`, où `bucket` et `region` sont les valeurs que vous spécifiez `gdk-config.json`, ainsi que votre `accountId` Compte AWS identifiant. La CLI GDK crée le bucket s'il n'existe pas.

Cet objet contient les informations suivantes :

`bucket`

Le nom du compartiment S3 à utiliser pour héberger les artefacts du composant.

`region`

L' Région AWS endroit où la CLI GDK publie ce composant.

Cette propriété est facultative si vous utilisez GDK CLI v1.3.0 ou version ultérieure.

options

(Facultatif) Options de configuration supplémentaires utilisées lors de la création de la version du composant.

Cette fonctionnalité est disponible pour GDK CLI v1.2.0 et versions ultérieures.

file_upload_args

Structure JSON contenant des arguments envoyés à Amazon S3 lors du chargement de fichiers dans un compartiment, tels que des métadonnées et des mécanismes de chiffrement. Pour une liste des arguments autorisés, consultez la [S3Transfer](#) classe dans la documentation de Boto3. .

test-e2e

(Facultatif) Configuration à utiliser lors du end-to-end test du composant. Cette fonctionnalité est disponible pour GDK CLI v1.3.0 et versions ultérieures.

build

build_system— Le système de construction à utiliser. L'option par défaut est `maven`. Sélectionnez parmi les options suivantes :

- **maven**— Exécute la `mvn package` commande pour créer le module de test. Choisissez cette option pour créer le module de test qui utilise [Maven](#).
- **gradle**— Exécute la `gradle build` commande pour créer le module de test. Choisissez cette option pour le module de test qui utilise [Gradle](#).

gtf_version

(Facultatif) Version du Greengrass Testing Framework (GTF) à utiliser comme dépendance du module de end-to-end test lorsque vous initialisez le projet GDK avec GTF. Cette valeur doit être l'une des versions GTF des [versions](#). La valeur par défaut est GTF version 1.1.0.

gtf_options

(Facultatif) Options de configuration supplémentaires utilisées lors du end-to-end test du composant.

La liste suivante inclut les options que vous pouvez utiliser avec la version 1.1.0 de GTF.

- **additional-plugins**— (Facultatif) Plugins Cucumber supplémentaires

- `aws-region`— Cible des points de terminaison régionaux spécifiques pour les AWS services. La valeur par défaut correspond à ce que le AWS SDK découvre.
- `credentials-path`— Chemin d'accès facultatif aux informations d'identification du AWS profil. Par défaut, ce sont les informations d'identification découvertes sur l'environnement hôte.
- `credentials-path-rotation`— Durée de rotation facultative pour les AWS informations d'identification. La valeur par défaut est de 15 minutes ou `PT15M`.
- `csr-path`— Le chemin du CSR à l'aide duquel le certificat de l'appareil sera généré.
- `device-mode`— L'appareil cible en cours de test. Par défaut, il s'agit d'un appareil local.
- `env-stage`— Cible l'environnement de déploiement de Greengrass. Par défaut, c'est la production.
- `existing-device-cert-arn`— L'ARN d'un certificat existant que vous souhaitez utiliser comme certificat d'appareil pour Greengrass.
- `feature-path`— Fichier ou répertoire contenant des fichiers de fonctionnalités supplémentaires. Par défaut, aucun fichier de fonctionnalités supplémentaire n'est utilisé.
- `gg-cli-version`— Remplace la version de la CLI Greengrass. La valeur par défaut est celle trouvée dans `ggc.version`
- `gg-component-bucket`— Le nom d'un compartiment Amazon S3 existant qui héberge les composants Greengrass.
- `gg-component-overrides`— Liste des remplacements de composants Greengrass.
- `gg-persist`— Liste des éléments de test à conserver après un essai. Le comportement par défaut est de ne rien conserver. Les valeurs acceptées sont : `aws.resourcesinstalled.software`, et `generated.files`.
- `gg-runtime`— Une liste de valeurs destinées à influencer la manière dont le test interagit avec les ressources de test. Ces valeurs remplacent le paramètre `gg.persist` Si la valeur par défaut est vide, cela suppose que toutes les ressources de test sont gérées par scénario de test, y compris le moteur d'exécution Greengrass installé. Les valeurs acceptées sont : `aws.resourcesinstalled.software`, et `generated.files`.
- `ggc-archive`— Le chemin d'accès au composant du noyau Greengrass archivé.
- `ggc-install-root`— Répertoire pour installer le composant Greengrass nucleus. La valeur par défaut est `test.temp.path` et le dossier `test run`.
- `ggc-log-level`— Définissez le niveau logarithmique du noyau Greengrass pour le test. La valeur par défaut est « INFO ».

- `ggc-tes-rolename`— Le rôle IAM que AWS IoT Greengrass Core assumera pour accéder aux AWS services. Si aucun rôle portant un nom donné n'existe, un rôle sera créé avec une politique d'accès par défaut.
- `ggc-trusted-plugins`— La liste séparée par des virgules des chemins (sur l'hôte) des plugins fiables qui doivent être ajoutés à Greengrass. Pour fournir le chemin sur le DUT lui-même, préfixez-le par « `dut :` »
- `ggc-user-name`— La valeur POSIXUser `user:group` pour le noyau Greengrass. La valeur par défaut est le nom d'utilisateur actuel connecté.
- `ggc-version`— Remplace la version du composant Greengrass nucleus en cours d'exécution. La valeur par défaut est celle trouvée dans `ggc.archive`.
- `log-level`— Niveau de journalisation du test. La valeur par défaut est « `INFO` ».
- `parallel-config`— Ensemble d'index de lots et de nombre de lots sous forme de chaîne JSON. La valeur par défaut de l'index des lots est 0 et le nombre de lots est 1.
- `proxy-url`— Configurez tous les tests pour acheminer le trafic via cette URL.
- `tags`— Exécutez uniquement des balises de fonctionnalité. Peut être intersecté avec « `&` »
- `test-id-prefix`— Un préfixe commun appliqué à toutes les ressources spécifiques au test, y compris les noms de AWS ressources et les balises. Le préfixe par défaut est « `gg` ».
- `test-log-path`— Répertoire qui contiendra les résultats de l'ensemble du test. La valeur par défaut est « `TestResults` ».
- `test-results-json`— Indicateur permettant de déterminer si le rapport Cucumber JSON résultant est généré écrit sur le disque. La valeur par défaut est `true` (vrai).
- `test-results-log`— Indicateur permettant de déterminer si la sortie de console est générée écrite sur le disque. La valeur par défaut est `false`.
- `test-results-xml`— Indicateur permettant de déterminer si le rapport JUnit XML obtenu est généré et écrit sur le disque. La valeur par défaut est `true` (vrai).
- `test-temp-path`— Répertoire pour générer des artefacts de test locaux. La valeur par défaut est un répertoire temporaire aléatoire préfixé par `gg-testing`.
- `timeout-multiplier`— Multiplicateur fourni à tous les délais d'expiration des tests. La valeur par défaut est 1,0.

Exemples de fichiers de configuration GDK CLI

Vous pouvez vous référer aux exemples de fichiers de configuration de la CLI GDK suivants pour vous aider à configurer les environnements de composants Greengrass.

Bonjour tout le monde (Python)

Le fichier de configuration de la CLI GDK suivant prend en charge un composant Hello World qui exécute un script Python. Ce fichier de configuration utilise le système de zip compilation pour emballer le script Python du composant dans un fichier ZIP que la CLI GDK télécharge sous forme d'artefact.

```
{
  "component": {
    "com.example.PythonHelloWorld": {
      "author": "Amazon",
      "version": "NEXT_PATCH",
      "build": {
        "build_system" : "zip",
        "options": {
          "excludes": [".*"]
        }
      },
      "publish": {
        "bucket": "greengrass-component-artifacts",
        "region": "us-west-2",
        "options": {
          "file_upload_args": {
            "Metadata": {
              "some-key": "some-value"
            }
          }
        }
      }
    }
  },
  "test-e2e":{
    "build":{
      "build_system": "maven"
    },
    "gtf_version": "1.1.0",
    "gtf_options": {
      "tags": "Sample"
    }
  },
  "gdk_version": "1.6.1"
}
```

Bonjour tout le monde (Java)

Le fichier de configuration GDK CLI suivant prend en charge un composant Hello World qui exécute une application Java. Ce fichier de configuration utilise le système de maven compilation pour emballer le code source Java du composant dans un fichier JAR que la CLI GDK télécharge sous forme d'artefact.

```
{
  "component": {
    "com.example.JavaHelloWorld": {
      "author": "Amazon",
      "version": "NEXT_PATCH",
      "build": {
        "build_system" : "maven"
      },
      "publish": {
        "bucket": "greengrass-component-artifacts",
        "region": "us-west-2",
        "options": {
          "file_upload_args": {
            "Metadata": {
              "some-key": "some-value"
            }
          }
        }
      }
    }
  },
  "test-e2e":{
    "build":{
      "build_system": "maven"
    },
    "gtf_version": "1.1.0",
    "gtf_options": {
      "tags": "Sample"
    }
  },
  "gdk_version": "1.6.1"
}
```

Composantes communautaires

Plusieurs composants communautaires du [catalogue de logiciels Greengrass](#) utilisent la CLI GDK. Vous pouvez explorer les fichiers de configuration de la CLI GDK dans les référentiels de ces composants.

Pour afficher les fichiers de configuration de la CLI GDK des composants de la communauté

1. Exécutez la commande suivante pour répertorier les composants de la communauté qui utilisent la CLI GDK.

```
gdk component list --repository
```

La réponse indique le nom du GitHub référentiel pour chaque composant de communauté qui utilise la CLI GDK. Chaque référentiel existe au sein de l'awslabsorganisation.

```
[2022-02-22 17:27:31] INFO - Listing all the available component repositories from
Greengrass Software Catalog.
[2022-02-22 17:27:31] INFO - Found '6' component repositories to display.
1. aws-greengrass-labs-database-influxdb
2. aws-greengrass-labs-telemetry-influxdbpublisher
3. aws-greengrass-labs-dashboard-grafana
4. aws-greengrass-labs-dashboard-influxdb-grafana
5. aws-greengrass-labs-local-web-server
6. aws-greengrass-labs-lookoutvision-gstreamer
```

2. Ouvrez le GitHub référentiel d'un composant communautaire à l'adresse URL suivante. Remplacez *community-component-name* par le nom d'un composant communautaire de l'étape précédente.

```
https://github.com/awslabs/community-component-name
```

Interface de ligne de commande Greengrass

L'interface de ligne de commande (CLI) Greengrass vous permet d'interagir avec AWS IoT Greengrass Core sur votre appareil pour développer des composants et résoudre des problèmes localement. Par exemple, vous pouvez utiliser la CLI Greengrass pour créer un déploiement local et redémarrer un composant sur le périphérique principal.

Déployez le [composant Greengrass CLI](#) (`aws.greengrass.Cli`) pour installer la Greengrass CLI sur votre appareil principal.

Important

Nous vous recommandons d'utiliser ce composant uniquement dans les environnements de développement, et non dans les environnements de production. Ce composant permet d'accéder aux informations et aux opérations dont vous n'avez généralement pas besoin dans un environnement de production. Suivez le principe du moindre privilège en déployant ce composant uniquement sur les appareils principaux là où vous en avez besoin.

Rubriques

- [Installation de la CLI Greengrass](#)
- [Commandes Greengrass CLI](#)

Installation de la CLI Greengrass

Vous pouvez installer la CLI Greengrass de l'une des manières suivantes :

- Utilisez `--deploy-dev-tools` cet argument lorsque vous configurez le logiciel AWS IoT Greengrass Core pour la première fois sur votre appareil. Vous devez également spécifier `--provision true` d'appliquer cet argument.
- Déployez le composant Greengrass CLI (`aws.greengrass.Cli`) sur votre appareil.

Cette section décrit les étapes de déploiement du composant Greengrass CLI. Pour plus d'informations sur l'installation de la CLI Greengrass lors de la configuration initiale, consultez.

[Tutoriel : Débuter avec AWS IoT Greengrass V2](#)

Conditions préalables

Pour déployer le composant Greengrass CLI, vous devez répondre aux exigences suivantes :

- AWS IoT Greengrass Logiciel de base installé et configuré sur votre appareil principal. Pour de plus amples informations, veuillez consulter [Tutoriel : Débuter avec AWS IoT Greengrass V2](#).

- Pour utiliser le AWS CLI pour déployer la Greengrass CLI, vous devez avoir installé et configuré le. AWS CLI Pour plus d'informations, veuillez consulter [configuration de l'outil AWS CLI](#) dans le guide de l'utilisateur de l'outil AWS Command Line Interface .
- Vous devez être autorisé à utiliser la CLI Greengrass pour interagir avec le logiciel AWS IoT Greengrass principal. Pour utiliser la CLI Greengrass, effectuez l'une des opérations suivantes :
 - Utilisez l'utilisateur du système qui exécute le logiciel AWS IoT Greengrass Core.
 - Utilisez un utilisateur doté d'autorisations root ou administratives. Sur les appareils principaux de Linux, vous pouvez l'utiliser sudo pour obtenir des autorisations root.
 - Utilisez un utilisateur système appartenant à un groupe que vous spécifiez dans les paramètres de AuthorizedWindowsGroups configuration AuthorizedPosixGroups ou lorsque vous déployez le composant. Pour plus d'informations, consultez la section Configuration des [composants de la CLI Greengrass](#).

Déployer le composant Greengrass CLI

Procédez comme suit pour déployer le composant Greengrass CLI sur votre appareil principal :

Pour déployer le composant Greengrass CLI (console)

1. Connectez-vous à la [console AWS IoT Greengrass](#).
2. Dans le menu de navigation, sélectionnez Composants.
3. Sur la page Components (Composants), sous l'onglet Public components (Composants publics), choisissez `aws.greengrass.Cli`.
4. Sur la page `aws.greengrass.Cli`, choisissez Deploy (Déployer).
5. Dans Ajouter au déploiement, choisissez Créer un nouveau déploiement.
6. Sur la page Spécifier la cible, sous Cibles de déploiement, dans la liste Nom de la cible, choisissez le groupe Greengrass vers lequel vous souhaitez effectuer le déploiement, puis cliquez sur Next.
7. Sur la page Sélectionner les composants, vérifiez que le `aws.greengrass.Cli` composant est sélectionné, puis choisissez Next.
8. Sur la page Configurer les composants, conservez les paramètres de configuration par défaut et choisissez Next.
9. Sur la page Configurer les paramètres avancés, conservez les paramètres de configuration par défaut et choisissez Next.
10. Sur la page de révision, cliquez sur Déployer

Pour déployer le composant Greengrass CLI ()AWS CLI

1. Sur votre appareil, créez un `deployment.json` fichier pour définir la configuration de déploiement du composant Greengrass CLI. Ce fichier doit ressembler à ce qui suit :

```
{
  "targetArn": "targetArn",
  "components": {
    "aws.greengrass.Cli": {
      "componentVersion": "2.16.1",
      "configurationUpdate": {
        "merge": "{\"AuthorizedPosixGroups\": \"<i>group1</i>, <i>group2</i>, ..., <i>groupN</i>\", \"AuthorizedWindowsGroups\": \"<i>group1</i>, <i>group2</i>, ..., <i>groupN</i>\"}"
      }
    }
  }
}
```

- Dans le champ `target`, remplacez *targetArn* par l'Amazon Resource Name (ARN) de l'objet ou du groupe d'objets à cibler pour le déploiement, au format suivant :
 - Objet : `arn:aws:iot:region:account-id:thing/thingName`
 - Groupe d'objets : `arn:aws:iot:region:account-id:thinggroup/thingGroupName`
- Dans l'objet `aws.greengrass.Cli` composant, spécifiez les valeurs comme suit :

`version`

Version du composant Greengrass CLI.

`configurationUpdate.AuthorizedPosixGroups`

(Facultatif) Chaîne contenant une liste de groupes de systèmes séparés par des virgules. Vous autorisez ces groupes de systèmes à utiliser la CLI Greengrass pour interagir avec le logiciel AWS IoT Greengrass principal. Vous pouvez spécifier des noms de groupes ou des groupes IDs. Par exemple, `group1,1002,group3` autorise trois groupes de systèmes (`group11002`, `etgroup3`) à utiliser la CLI Greengrass.

Si vous ne spécifiez aucun groupe à autoriser, vous pouvez utiliser la CLI Greengrass en tant qu'utilisateur `root` (`sudo`) ou en tant qu'utilisateur système qui exécute le logiciel AWS IoT Greengrass Core.

configurationUpdate.AuthorizedWindowsGroups

(Facultatif) Chaîne contenant une liste de groupes de systèmes séparés par des virgules. Vous autorisez ces groupes de systèmes à utiliser la CLI Greengrass pour interagir avec le logiciel AWS IoT Greengrass principal. Vous pouvez spécifier des noms de groupes ou des groupes IDs. Par exemple, `group1,1002,group3` autorise trois groupes de systèmes (`group11002`, `etgroup3`) à utiliser la CLI Greengrass.

Si vous ne spécifiez aucun groupe à autoriser, vous pouvez utiliser la CLI Greengrass en tant qu'administrateur ou en tant qu'utilisateur du système qui exécute le logiciel AWS IoT Greengrass principal.

2. Exécutez la commande suivante pour déployer le composant Greengrass CLI sur le périphérique :

```
$ aws greengrassv2 create-deployment --cli-input-json file://path/to/deployment.json
```

Pendant l'installation, le composant ajoute un lien symbolique `greengrass-cli` dans le `/greengrass/v2/bin` dossier de votre appareil, et vous exécutez la CLI Greengrass à partir de ce chemin. Pour exécuter la CLI Greengrass sans son chemin absolu, ajoutez votre `/greengrass/v2/bin` dossier à votre variable PATH. Pour vérifier l'installation de Greengrass CLI, exécutez la commande suivante :

Linux or Unix

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

Windows

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

Vous devriez voir la sortie suivante :

```
Usage: greengrass-cli [-hV] [--ggcRootPath=<ggcRootPath>] [COMMAND]
Greengrass command line interface

--ggcRootPath=<ggcRootPath>
```

```

The AWS IoT Greengrass V2 root directory.
-h, --help      Show this help message and exit.
-V, --version   Print version information and exit.

Commands:
  help          Show help information for a command.
  component     Retrieve component information and stop or restart
                components.
  deployment    Create local deployments and retrieve deployment status.
  logs          Analyze Greengrass logs.
  get-debug-password Generate a password for use with the HTTP debug view
                component.

```

S'il `greengrass-cli` n'est pas trouvé, le déploiement n'a peut-être pas réussi à installer la CLI Greengrass. Pour de plus amples informations, veuillez consulter [Résolution des problèmes AWS IoT Greengrass V2](#).

Commandes Greengrass CLI

La CLI Greengrass fournit une interface de ligne de commande pour interagir localement avec votre appareil AWS IoT Greengrass principal. Les commandes Greengrass CLI utilisent le format suivant.

```
$ greengrass-cli <command> <subcommand> [arguments]
```

Par défaut, le fichier `greengrass-cli` exécutable du `/greengrass/v2/bin/` dossier interagit avec la version du logiciel AWS IoT Greengrass Core exécutée dans le `/greengrass/v2` dossier. Si vous appelez un exécutable qui n'est pas placé à cet emplacement, ou si vous souhaitez interagir avec le logiciel AWS IoT Greengrass Core à un autre emplacement, vous devez utiliser l'une des méthodes suivantes pour spécifier explicitement le chemin racine du logiciel AWS IoT Greengrass Core avec lequel vous souhaitez interagir :

- Définissez la variable d'environnement `GGC_ROOT_PATH` sur `/greengrass/v2`.
- Ajoutez l'option `--ggcRootPath /greengrass/v2` argument à votre commande comme indiqué dans l'exemple suivant.

```
greengrass-cli --ggcRootPath /greengrass/v2 <command> <subcommand> [arguments]
```

Vous pouvez utiliser les arguments suivants avec n'importe quelle commande :

- À utiliser `--help` pour obtenir des informations sur une commande Greengrass CLI spécifique.

- À utiliser `--version` pour obtenir des informations sur la version de Greengrass CLI.

Cette section décrit les commandes de la CLI Greengrass et fournit des exemples de ces commandes. Le synopsis de chaque commande montre ses arguments et leur utilisation. Les arguments facultatifs sont indiqués entre crochets.

Commandes disponibles

- [composant](#)
- [déploiement](#)
- [journaux](#)
- [get-debug-password](#)

composant

Utilisez la `component` commande pour interagir avec les composants locaux de votre appareil principal.

Sous-commandes

- [détails](#)
- [liste](#)
- [redémarrer](#)
- [stop](#)

détails

Récupérez la version, le statut et la configuration d'un composant.

Résumé

```
greengrass-cli component details --name <component-name>
```

Arguments

`--name, -n`. Le nom du composant.

Sortie

L'exemple suivant montre le résultat produit lorsque vous exécutez cette commande.

```
$ sudo greengrass-cli component details --name MyComponent
```

```
Component Name: MyComponent  
Version: 1.0.0  
State: RUNNING  
Configuration: null
```

liste

Récupérez le nom, la version, l'état et la configuration de chaque composant installé sur le périphérique.

Résumé

```
greengrass-cli component list
```

Arguments

Aucun

Sortie

L'exemple suivant montre le résultat produit lorsque vous exécutez cette commande.

```
$ sudo greengrass-cli component list  
  
Components currently running in Greengrass:  
Component Name: FleetStatusService  
Version: 0.0.0  
State: RUNNING  
Configuration: {"periodicUpdateIntervalSec":86400.0}  
Component Name: UpdateSystemPolicyService  
Version: 0.0.0  
State: RUNNING  
Configuration: null  
Component Name: aws.greengrass.Nucleus  
Version: 2.0.0  
State: FINISHED  
Configuration: {"awsRegion":"region","runWithDefault":  
{"posixUser":"ggc_user:ggc_group"},"telemetry":{}}  
Component Name: DeploymentService  
Version: 0.0.0
```

```
State: RUNNING
Configuration: null
Component Name: TelemetryAgent
Version: 0.0.0
State: RUNNING
Configuration: null
Component Name: aws.greengrass.Cli
Version: 2.0.0
State: RUNNING
Configuration: {"AuthorizedPosixGroups":"ggc_user"}
```

redémarrer

Redémarrez les composants.

Résumé

```
greengrass-cli component restart --names <component-name>,...
```

Arguments

--names, -n. Le nom du composant. Au moins un nom de composant est requis. Vous pouvez spécifier des noms de composants supplémentaires, en séparant chaque nom par une virgule.

Sortie

Aucun

stop

Arrêtez d'exécuter les composants.

Résumé

```
greengrass-cli component stop --names <component-name>,...
```

Arguments

--names, -n. Le nom du composant. Au moins un nom de composant est requis. Vous pouvez spécifier des noms de composants supplémentaires si nécessaire, en séparant chaque nom par une virgule.

Sortie

Aucun

déploiement

Utilisez la `deployment` commande pour interagir avec les composants locaux de votre appareil principal.

Pour suivre la progression d'un déploiement local, utilisez la `status` sous-commande. Vous ne pouvez pas suivre la progression d'un déploiement local à l'aide de la console.

Sous-commandes

- [créer](#)
- [annuler](#)
- [liste](#)
- [status](#)

créer

Créez ou mettez à jour un déploiement local à l'aide de recettes de composants, d'artefacts et d'arguments d'exécution spécifiés.

Résumé

```
greengrass-cli deployment create
  --recipeDir path/to/component/recipe
  [--artifactDir path/to/artifact/folder ]
  [--update-config {component-configuration}]
  [--groupId <thing-group>]
  [--merge "<component-name>=<component-version>"]...
  [--runWith "<component-name>:posixUser=<user-name>[:<group-name>]"...]
  [--systemLimits "{component-system-resource-limits}"]...
  [--remove <component-name>,...]
  [--failure-handling-policy <policy name>[ROLLBACK, DO_NOTHING]>]
```

Arguments

- `--recipeDir, -r`. Le chemin complet du dossier contenant les fichiers de recette des composants.

- `--artifactDir,-a`. Le chemin complet du dossier contenant les fichiers d'artefacts que vous souhaitez inclure dans votre déploiement. Le dossier des artefacts doit contenir la structure de répertoire suivante :

```
/path/to/artifact/folder/<component-name>/<component-version>/<artifacts>
```

- `--update-config,-c`. Les arguments de configuration pour le déploiement, fournis sous forme de chaîne JSON ou de fichier JSON. La chaîne JSON doit être au format suivant :

```
{ \
  "componentName": { \
    "MERGE": {"config-key": "config-value"}, \
    "RESET": ["path/to/reset/"] \
  } \
}
```

MERGE et RESET distinguent les majuscules des minuscules et doivent être en majuscules.

- `--groupId,-g`. Le groupe d'objets cible pour le déploiement.
- `--merge,-m`. Le nom et la version du composant cible que vous souhaitez ajouter ou mettre à jour. Vous devez fournir les informations du composant dans le format `<component>=<version>`. Utilisez un argument distinct pour chaque composant supplémentaire à spécifier. Si nécessaire, utilisez l'`--runWith` argument pour fournir les `windowsUser` informations `posixUser` `posixGroup`, et pour exécuter le composant.
- `--runWith`. Les `posixUser` `windowsUser` informations et relatives à l'exécution d'un composant générique ou Lambda. `posixGroup` Vous devez fournir ces informations dans le format `<component>:{posixUser|windowsUser}=<user>[:<=posixGroup>]`. Par exemple, vous pouvez spécifier **HelloWorld:posixUser=ggc_user:ggc_group** ou **HelloWorld:windowsUser=ggc_user**. Utilisez un argument distinct pour chaque option supplémentaire à spécifier.

Pour de plus amples informations, veuillez consulter [Configurer l'utilisateur qui exécute les composants](#).

- `--systemLimits`. Les limites de ressources du système à appliquer aux processus des composants Lambda génériques et non conteneurisés sur le périphérique principal. Vous pouvez configurer la quantité maximale d'utilisation du processeur et de la RAM que les processus de chaque composant peuvent utiliser. Spécifiez un objet JSON sérialisé ou un chemin d'accès à un fichier JSON. L'objet JSON doit avoir le format suivant.

```
{ \
  "componentName": { \
    "cpus": cpuTimeLimit, \
    "memory": memoryLimitInKb \
  } \
}
```

Vous pouvez configurer les limites de ressources système suivantes pour chaque composant :

- **cpus**— Le temps processeur maximal que les processus de ce composant peuvent utiliser sur le périphérique principal. Le temps processeur total d'un appareil principal est équivalent au nombre de cœurs processeurs de l'appareil. Par exemple, sur un périphérique principal doté de 4 cœurs de processeur, vous pouvez définir cette valeur 2 pour limiter les processus de ce composant à 50 % d'utilisation de chaque cœur de processeur. Sur un appareil doté d'un cœur de processeur, vous pouvez définir cette valeur 0.25 pour limiter les processus de ce composant à 25 % d'utilisation du processeur. Si vous définissez cette valeur sur un nombre supérieur au nombre de cœurs de processeur, le logiciel AWS IoT Greengrass Core ne limite pas l'utilisation du processeur par le composant.
- **memory**— La quantité maximale de RAM (en kilo-octets) que les processus de ce composant peuvent utiliser sur le périphérique principal.

Pour de plus amples informations, veuillez consulter [Configuration des limites de ressources système pour les composants](#).

Cette fonctionnalité est disponible pour les versions 2.4.0 et ultérieures du [composant Greengrass nucleus et de la CLI Greengrass](#) sur les appareils principaux Linux. AWS IoT Greengrass ne prend actuellement pas en charge cette fonctionnalité sur les appareils Windows principaux.

- **--remove**. Nom du composant cible que vous souhaitez supprimer d'un déploiement local. Pour supprimer un composant qui a été fusionné d'un déploiement dans le cloud, vous devez fournir l'ID de groupe du groupe d'objets cible au format suivant :

Greengrass nucleus v2.4.0 and later

```
--remove <component-name> --groupId <group-name>
```

Earlier than v2.4.0

```
--remove <component-name> --groupId thinggroup/<group-name>
```

- `--failure-handling-policy`. Définit l'action entreprise en cas d'échec d'un déploiement. Vous pouvez définir deux actions :
 - `ROLLBACK` –
 - `DO_NOTHING` –

Cette fonctionnalité est disponible pour les versions 2.11.0 et ultérieures du [Noyau de Greengrass](#)

Sortie

L'exemple suivant montre le résultat produit lorsque vous exécutez cette commande.

```
$ sudo greengrass-cli deployment create \  
  --merge MyApp1=1.0.0 \  
  --merge MyApp2=1.0.0 --runWith MyApp2:posixUser=ggc_user \  
  --remove MyApp3 \  
  --recipeDir recipes/ \  
  --artifactDir artifacts/  
  
Local deployment has been submitted! Deployment Id: 44d89f46-1a29-4044-  
ad89-5151213dfcbc
```

annuler

Annule le déploiement spécifié.

Résumé

```
greengrass-cli deployment cancel  
-i <deployment-id>
```

Arguments

- i. Identifiant unique du déploiement à annuler. L'ID de déploiement est renvoyé dans la sortie de la `create` commande.

Sortie

- Aucun

liste

Récupérez l'état des 10 derniers déploiements locaux.

Résumé

```
greengrass-cli deployment list
```

Arguments

Aucun

Sortie

L'exemple suivant montre le résultat produit lorsque vous exécutez cette commande. En fonction de l'état de votre déploiement, la sortie affiche l'une des valeurs d'état suivantes : IN_PROGRESS, SUCCEEDED, ou FAILED.

```
$ sudo greengrass-cli deployment list

44d89f46-1a29-4044-ad89-5151213dfcbc: SUCCEEDED
Created on: 6/27/23 11:05 AM
```

status

Récupérez l'état d'un déploiement spécifique.

Résumé

```
greengrass-cli deployment status -i <deployment-id>
```

Arguments

-i. ID du déploiement.

Sortie

L'exemple suivant montre le résultat produit lorsque vous exécutez cette commande. En fonction de l'état de votre déploiement, la sortie affiche l'une des valeurs d'état suivantes : IN_PROGRESS, SUCCEEDED, ou FAILED.

```
$ sudo greengrass-cli deployment status -i 44d89f46-1a29-4044-ad89-5151213dfcbc

44d89f46-1a29-4044-ad89-5151213dfcbc: FAILED
Created on: 6/27/23 11:05 AM
Detailed Status: <Detailed deployment status>
Deployment Error Stack: List of error codes
Deployment Error Types: List of error types
Failure Cause: Cause
```

journaux

Utilisez la `logs` commande pour analyser les journaux Greengrass sur votre appareil principal.

Sous-commandes

- [get](#)
- [listez les mots clés](#)
- [list-log-files](#)

get

Collectez, filtrez et visualisez les fichiers journaux de Greengrass. Cette commande prend uniquement en charge les fichiers journaux au format JSON. Vous pouvez spécifier le [format de journalisation](#) dans la configuration du noyau.

Résumé

```
greengrass-cli logs get
  [--log-dir path/to/a/log/folder]
  [--log-file path/to/a/log/file]
  [--follow true | false ]
  [--filter <filter> ]
  [--time-window <start-time>,<end-time> ]
```

```
[--verbose ]  
[--no-color ]  
[--before <value> ]  
[--after <value> ]  
[--syslog ]  
[--max-long-queue-size <value> ]
```

Arguments

- `--log-dir,-ld`. Le chemin d'accès au répertoire dans lequel vérifier la présence de fichiers journaux, tels que **/greengrass/v2/logs**. Ne pas utiliser avec `--syslog`. Utilisez un argument distinct pour chaque répertoire supplémentaire à spécifier. Vous devez utiliser au moins l'un des `--log-dir` ou `--log-file`. Vous pouvez également utiliser les deux arguments dans une seule commande.
- `--log-file,-lf`. Les chemins d'accès aux répertoires de journaux que vous souhaitez utiliser. Utilisez un argument distinct pour chaque répertoire supplémentaire à spécifier. Vous devez utiliser au moins l'un des `--log-dir` ou `--log-file`. Vous pouvez également utiliser les deux arguments dans une seule commande.
- `--follow,-fol`. Afficher les mises à jour du journal dès qu'elles se produisent. Greengrass CLI continue de fonctionner et lit les journaux spécifiés. Si vous spécifiez une fenêtre temporelle, Greengrass CLI arrête de surveiller les journaux une fois toutes les fenêtres temporelles terminées.
- `--filter,-f`. Le mot clé, les expressions régulières ou la paire clé-valeur à utiliser comme filtre. Fournissez cette valeur sous forme de chaîne, d'expression régulière ou de paire clé-valeur. Utilisez un argument distinct pour chaque filtre supplémentaire à spécifier.

Lors de l'évaluation, plusieurs filtres spécifiés dans un seul argument sont séparés par des opérateurs OR, et les filtres spécifiés dans des arguments supplémentaires sont combinés avec des opérateurs ET. Par exemple, si votre commande inclut `--filter "installed" --filter "name=alpha,name=beta"`, la CLI Greengrass filtrera et affichera les messages de journal contenant à la fois le mot clé `installed` et une name clé contenant les valeurs `alpha` ou `beta`.

- `--time-window,-t`. La fenêtre temporelle pour laquelle les informations du journal doivent être affichées. Vous pouvez utiliser à la fois des horodatages exacts et des décalages relatifs. Vous devez fournir ces informations dans le format **<begin-time>**, **<end-time>**. Si vous ne spécifiez ni l'heure de début ni l'heure de fin, la valeur de cette option est par défaut la date et l'heure actuelles du système. Utilisez un argument distinct pour chaque fenêtre temporelle supplémentaire à spécifier.

Greengrass CLI prend en charge les formats d'horodatage suivants :

- `yyyy-MM-DD`, par exemple, `2020-06-30`. L'heure par défaut est 00:00:00 lorsque vous utilisez ce format.

`yyyyMMdd`, par exemple, `20200630`. L'heure par défaut est 00:00:00 lorsque vous utilisez ce format.

`HH:mm:ss`, par exemple, `15:30:45`. La date par défaut est la date système actuelle lorsque vous utilisez ce format.

`HH:mm:ssSSS`, par exemple, `15:30:45`. La date correspond par défaut à la date système actuelle lorsque vous utilisez ce format.

`YYYY-MM-DD 'T' HH:mm:ss 'Z'`, par exemple, `2020-06-30T15:30:45Z`.

`YYYY-MM-DD 'T' HH:mm:ss`, par exemple, `2020-06-30T15:30:45`.

`yyyy-MM-dd 'T' HH:mm:ss .SSS`, par exemple, `2020-06-30T15:30:45.250`.

Les décalages relatifs spécifient un décalage de période par rapport à l'heure actuelle du système. Greengrass CLI prend en charge le format suivant pour les décalages relatifs : `+. | -` [`<value>`h|h|r|hours] [`value`m|min|minutes] [`value`s|sec|seconds]

Par exemple, l'argument suivant pour spécifier une fenêtre horaire comprise entre 1 heure et 2 heures 15 minutes avant l'heure actuelle `--time-window -2h15min, -1hr`.

- `--verbose`. Afficher tous les champs des messages du journal. Ne pas utiliser avec `--syslog`.
- `--no-color, -nc`. Supprimez le code couleur. Le code couleur par défaut pour les messages du journal utilise du texte rouge en gras. Supporte uniquement les terminaux de type Unix car il utilise des séquences d'échappement ANSI.
- `--before, -b`. Le nombre de lignes à afficher avant une entrée de journal correspondante. La valeur par défaut est 0.
- `--after, -a`. Le nombre de lignes à afficher après une entrée de journal correspondante. La valeur par défaut est 0.
- `--syslog`. Traitez tous les fichiers journaux à l'aide du protocole syslog défini par RFC3164. Ne pas utiliser avec `--log-dir` et `--verbose`. Le protocole Syslog utilise le format suivant : `"<$Priority>$Timestamp $Host $Logger ($Class): $Message"` Si vous ne

spécifiez pas de fichier journal, Greengrass CLI lit les messages de journal depuis les emplacements suivants : `/var/log/messages`/`/var/log/syslog`, ou le `/var/log/system.log`

AWS IoT Greengrass ne prend actuellement pas en charge cette fonctionnalité sur les appareils Windows principaux.

- `--max-log-queue-size,-m`. Nombre maximal d'entrées de journal à allouer à la mémoire. Utilisez cette option pour optimiser l'utilisation de la mémoire. La valeur par défaut est 100.

Sortie

L'exemple suivant montre le résultat produit lorsque vous exécutez cette commande.

```
$ sudo greengrass-cli logs get --verbose \  
  --log-file /greengrass/v2/logs/greengrass.log \  
  --filter deployment,serviceName=DeploymentService \  
  --filter level=INFO \  
  --time-window 2020-12-08T01:11:17,2020-12-08T01:11:22  
  
2020-12-08T01:11:17.615Z [INFO] (pool-2-thread-14)  
com.aws.greengrass.deployment.DeploymentService: Current deployment finished.  
{DeploymentId=44d89f46-1a29-4044-ad89-5151213dfcbc, serviceName=DeploymentService,  
currentState=RUNNING}  
2020-12-08T01:11:17.675Z [INFO] (pool-2-thread-14)  
com.aws.greengrass.deployment.IotJobsHelper: Updating status of persisted  
deployment. {Status=SUCCEEDED, StatusDetails={detailed-deployment-  
status=SUCCESSFUL}, ThingName=MyThing, JobId=22d89f46-1a29-4044-ad89-5151213dfcbc
```

listez les mots clés

Afficher les mots-clés suggérés que vous pouvez utiliser pour filtrer les fichiers journaux.

Résumé

```
greengrass-cli logs list-keywords [arguments]
```

Arguments

Aucun

Sortie

Les exemples suivants montrent le résultat produit lorsque vous exécutez cette commande.

```
$ sudo greengrass-cli logs list-keywords

Here is a list of suggested keywords for Greengrass log:
level=$str
thread=$str
loggerName=$str
eventType=$str
serviceName=$str
error=$str
```

```
$ sudo greengrass-cli logs list-keywords --syslog

Here is a list of suggested keywords for syslog:
priority=$int
host=$str
logger=$str
class=$str
```

list-log-files

Afficher les fichiers journaux situés dans un répertoire spécifié.

Résumé

```
greengrass-cli logs list-log-files [arguments]
```

Arguments

`--log-dir, -ld`. Le chemin d'accès au répertoire dans lequel vérifier la présence de fichiers journaux.

Sortie

L'exemple suivant montre le résultat produit lorsque vous exécutez cette commande.

```
$ sudo greengrass-cli logs list-log-files -ld /greengrass/v2/logs/

/greengrass/v2/logs/aws.greengrass.Nucleus.log
```

```
/greengrass/v2/logs/main.log
/greengrass/v2/logs/greengrass.log
Total 3 files found.
```

get-debug-password

Utilisez la `get-debug-password` commande pour imprimer un mot de passe généré aléatoirement pour le [composant de console de débogage local](#) (`aws.greengrass.LocalDebugConsole`). Le mot de passe expire 8 heures après sa création.

Résumé

```
greengrass-cli get-debug-password
```

Arguments

Aucun

Sortie

L'exemple suivant montre le résultat produit lorsque vous exécutez cette commande.

```
$ sudo greengrass-cli get-debug-password

Username: debug
Password: bEDp3M0Hdj8ou2w5de_sCBI2XAaguy3a8XxREXAMPLE
Password expires at: 2021-04-01T17:01:43.921999931-07:00
The local debug console is configured to use TLS security. The certificate is self-
signed so you will need to bypass your web browser's security warnings to open the
console.
Before you bypass the security warning, verify that the certificate fingerprint
matches the following fingerprints.
SHA-256: 15 0B 2C E2 54 8B 22 DE 08 46 54 8A B1 2B 25 DE FB 02 7D 01 4E 4A 56 67 96
DA A6 CC B1 D2 C4 1B
SHA-1: BC 3E 16 04 D3 80 70 DA E0 47 25 F9 90 FA D6 02 80 3E B5 C1
```

Utiliser le framework AWS IoT Greengrass de test

Greengrass Testing Framework (GTF) est un ensemble de composants qui soutiennent l'end-to-endautomatisation du point de vue du client. GTF utilise [Cucumber](#) comme pilote de fonctionnalités.

AWS IoT Greengrass utilise les mêmes éléments de base pour qualifier les modifications logicielles sur différents appareils. Pour plus d'informations, consultez [Greengrass Testing Framework sur Github](#).

Le GTF est implémenté à l'aide de Cucumber, un outil utilisé pour exécuter des tests automatisés, afin d'encourager un développement des composants piloté par le comportement (BDD). Dans Cucumber, les fonctionnalités de ce système sont décrites dans un type de fichier spécial appelé `feature`. Chaque fonctionnalité est décrite dans un format lisible par l'homme appelé scénarios, qui sont des spécifications pouvant être converties en tests automatisés. Chaque scénario est décrit comme une série d'étapes qui définissent les interactions et les résultats de ce système testé à l'aide d'un langage spécifique au domaine appelé Gherkin. Une [étape Gherkin](#) est liée au code de programmation à l'aide d'une méthode appelée définition d'étape qui relie la spécification au flux de test. Les définitions d'étapes dans GTF sont implémentées avec Java.

Rubriques

- [Comment ça marche](#)
- [Journal des modifications](#)
- [Options de configuration du Greengrass Testing Framework](#)
- [Tutoriel : Exécuter end-to-end des tests à l'aide du framework de test Greengrass et du kit de développement Greengrass](#)
- [Tutoriel : Utiliser un test de confiance issu de la suite de tests de confiance](#)

Comment ça marche

AWS IoT Greengrass distribue le GTF sous la forme d'un JAR autonome composé de plusieurs modules Java. Pour utiliser GTF pour end-to-end tester des composants, vous devez implémenter les tests dans un projet Java. L'ajout du JAR autonome de test en tant que dépendance dans votre projet Java vous permet d'utiliser les fonctionnalités existantes du GTF et de les étendre en écrivant vos propres cas de test personnalisés. Pour exécuter les scénarios de test personnalisés, vous pouvez créer votre projet Java et exécuter le fichier JAR cible avec les options de configuration décrites dans [Options de configuration du Greengrass Testing Framework](#).

JAR autonome GTF

Greengrass utilise Cloudfront comme référentiel [Maven](#) pour héberger différentes versions du JAR autonome GTF. Pour une liste complète des versions de GTF, voir les versions de [GTF](#).

Le JAR autonome GTF inclut les modules suivants. Il n'est pas limité à ces seuls modules. Vous pouvez sélectionner chacune de ces dépendances séparément dans votre projet ou les inclure toutes en même temps dans le [fichier JAR autonome de test](#).

- `aws-greengrass-testing-resources`: Ce module fournit une abstraction permettant de gérer le cycle de vie d'une AWS ressource au cours d'un test. Vous pouvez l'utiliser pour définir vos AWS ressources personnalisées à l'aide de l'`ResourceSpec` abstraction afin que GTF puisse s'occuper de la création et de la suppression de ces ressources pour vous.
- `aws-greengrass-testing-platform`: Ce module fournit une abstraction au niveau de la plate-forme pour le périphérique testé pendant le cycle de vie du test. Il permet APIs d'interagir avec le système d'exploitation indépendamment de la plate-forme et peut être utilisé pour simuler les commandes exécutées dans le shell de l'appareil.
- `aws-greengrass-testing-components`: Ce module comprend des exemples de composants utilisés pour tester les fonctionnalités principales de Greengrass, telles que les déploiements, l'IPC et d'autres fonctionnalités.
- `aws-greengrass-testing-features`: Ce module comprend des étapes communes réutilisables et leurs définitions qui sont utilisées pour les tests dans l'environnement Greengrass.

Rubriques

- [Journal des modifications](#)
- [Options de configuration du Greengrass Testing Framework](#)
- [Tutoriel : Exécuter end-to-end des tests à l'aide du framework de test Greengrass et du kit de développement Greengrass](#)
- [Tutoriel : Utiliser un test de confiance issu de la suite de tests de confiance](#)

Journal des modifications

Le tableau suivant décrit les modifications apportées à chaque version du GTF. Pour plus d'informations, consultez la [page des versions du GTF](#) sur GitHub.

Version	Modifications
1.2.0	Nouvelles fonctionnalités • Ajoute des étapes liées au réseau pour configurer le MQTT et la connectivité au réseau Internet pendant les tests.

Version	Modifications
	• Ajoute des étapes métriques au système pour surveiller l'utilisation de la RAM et du processeur de l'appareil. Corrections de bogues et améliorations • L'étape de déploiement local de Greengrass CLI est réessayée jusqu'à ce qu'elle réussisse. • Les tests arrêtent gracieusement le noyau de Greengrass au lieu de le tuer. • Ajoute une amélioration grâce à laquelle GTF interroge le point de terminaison des AWS IoT informations d'identification jusqu'à ce que les informations d'identification soient récupérables pour l'objet et l'alias de rôle. • Corrige les artefacts et les répertoires de recettes manquants. Cette version corrige également les versions de composants manquantes. • Résout un problème d'échec du GTF lors du nettoyage de l'image docker si l'image docker n'existe pas. • Ajoute le mot clé CURRENT en tant que version du composant.
1.1.0	Nouvelles fonctionnalités • Permet d'installer un composant personnalisé avec configuration. Cela nécessite une recette pour le composant personnalisé. • Permet de mettre à jour un déploiement local avec une configuration personnalisée. Corrections de bogues et améliorations • Résout le problème d'incohérence entre les versions GTF du contexte du journal.
1.0.0	Première version.

Options de configuration du Greengrass Testing Framework

Options de configuration GTF

Greengrass Testing Framework (GTF) vous permet de configurer certains paramètres lors du lancement du processus de end-to-end test afin d'orchestrer le flux de test. Vous pouvez spécifier ces options de configuration en tant qu'arguments CLI pour le JAR autonome GTF.

Les versions 1.1.0 et ultérieures de GTF fournissent les options de configuration suivantes.

- `additional-plugins`— (Facultatif) Plugins Cucumber supplémentaires
- `aws-region`— Cible des points de terminaison régionaux spécifiques pour les AWS services. La valeur par défaut correspond à ce que le AWS SDK découvre.
- `credentials-path`— Chemin d'accès facultatif aux informations d'identification du AWS profil. Par défaut, ce sont les informations d'identification découvertes sur l'environnement hôte.
- `credentials-path-rotation`— Durée de rotation facultative pour les AWS informations d'identification. La valeur par défaut est 15 minutes ou `PT15M`.
- `csr-path`— Le chemin du CSR à l'aide duquel le certificat de l'appareil sera généré.
- `device-mode`— L'appareil cible testé. Par défaut, c'est le périphérique local.
- `env-stage`— Cible l'environnement de déploiement de Greengrass. Par défaut, c'est la production.
- `existing-device-cert-arn`— L'ARN d'un certificat existant que vous souhaitez utiliser comme certificat d'appareil pour Greengrass.
- `feature-path`— Fichier ou répertoire contenant des fichiers de fonctionnalités supplémentaires. Par défaut, aucun fichier de fonctionnalités supplémentaire n'est utilisé.
- `gg-cli-version`— Remplace la version de la CLI Greengrass. La valeur par défaut est celle trouvée dans `ggc.version`
- `gg-component-bucket`— Le nom d'un compartiment Amazon S3 existant qui héberge les composants Greengrass.
- `gg-component-overrides`— Liste des remplacements de composants Greengrass.
- `gg-persist`— Liste des éléments de test à conserver après un essai. Le comportement par défaut est de ne rien conserver. Les valeurs acceptées sont : `aws.resourcesinstalled.software`, et `generated.files`.
- `gg-runtime`— Une liste de valeurs destinées à influencer la manière dont le test interagit avec les ressources de test. Ces valeurs remplacent le paramètre `gg.persist` Si la valeur

par défaut est vide, cela suppose que toutes les ressources de test sont gérées par scénario de test, y compris le moteur d'exécution Greengrass installé. Les valeurs acceptées sont : `aws.resourcesinstalled.software`, `etgenerated.files`.

- `ggc-archive`— Le chemin d'accès au composant du noyau Greengrass archivé.
- `ggc-install-root`— Répertoire pour installer le composant Greengrass nucleus. La valeur par défaut est `test.temp.path` et le dossier `test run`.
- `ggc-log-level`— Définissez le niveau logarithmique du noyau Greengrass pour le test. La valeur par défaut est « INFO ».
- `ggc-tes-rolename`— Le rôle IAM que AWS IoT Greengrass Core assumera pour accéder aux AWS services. Si aucun rôle portant un nom donné n'existe, un rôle sera créé avec une politique d'accès par défaut.
- `ggc-trusted-plugins`— La liste séparée par des virgules des chemins (sur l'hôte) des plugins fiables qui doivent être ajoutés à Greengrass. Pour fournir le chemin sur le DUT lui-même, préfixez-le par « `dut :` »
- `ggc-user-name`— La valeur POSIXUser de `user:group` pour le noyau Greengrass. La valeur par défaut est le nom d'utilisateur actuel connecté.
- `ggc-version`— Remplace la version du composant Greengrass nucleus en cours d'exécution. La valeur par défaut est celle trouvée dans `ggc.archive`.
- `log-level`— Niveau de journalisation du test. La valeur par défaut est « INFO ».
- `parallel-config`— Ensemble d'index de lots et de nombre de lots sous forme de chaîne JSON. La valeur par défaut de l'index des lots est 0 et le nombre de lots est 1.
- `proxy-url`— Configurez tous les tests pour acheminer le trafic via cette URL.
- `tags`— Exécutez uniquement des balises de fonctionnalité. Peut être intersecté avec « & »
- `test-id-prefix`— Un préfixe commun appliqué à toutes les ressources spécifiques au test, y compris les noms de AWS ressources et les balises. Le préfixe par défaut est « `gg` ».
- `test-log-path`— Répertoire qui contiendra les résultats de l'ensemble du test. La valeur par défaut est « `TestResults` ».
- `test-results-json`— Indicateur permettant de déterminer si le rapport Cucumber JSON résultant est généré écrit sur le disque. La valeur par défaut est `true` (vrai).
- `test-results-log`— Indicateur permettant de déterminer si la sortie de console est générée écrite sur le disque. La valeur par défaut est `false`.
- `test-results-xml`— Indicateur permettant de déterminer si le rapport JUnit XML obtenu est généré et écrit sur le disque. La valeur par défaut est `true` (vrai).

- `test-temp-path`— Répertoire pour générer des artefacts de test locaux. La valeur par défaut est un répertoire temporaire aléatoire préfixé par `gg-testing`.
- `timeout-multiplier`— Multiplicateur fourni à tous les délais d'expiration des tests. La valeur par défaut est 1,0.

Tutoriel : Exécuter end-to-end des tests à l'aide du framework de test Greengrass et du kit de développement Greengrass

AWS IoT Greengrass Le Testing Framework (GTF) et le Greengrass Development Kit (GDK) offrent aux développeurs des moyens d'exécuter des tests. end-to-end Vous pouvez suivre ce didacticiel pour initialiser un projet GDK avec un composant, initialiser un projet GDK avec un module de end-to-end test et créer un cas de test personnalisé. Après avoir créé votre scénario de test personnalisé, vous pouvez exécuter le test.

Dans ce didacticiel, vous allez effectuer les opérations suivantes :

1. Initialisez un projet GDK avec un composant.
2. Initialisez un projet GDK avec un module de end-to-end test.
3. Créez un cas de test personnalisé.
4. Ajoutez une étiquette au nouveau scénario de test.
5. Créez le fichier JAR de test.
6. Exécutez le test .

Rubriques

- [Prérequis](#)
- [Étape 1 : Initialisation d'un projet GDK avec un composant](#)
- [Étape 2 : Initialisation d'un projet GDK avec un module de test end-to-end](#)
- [Étape 3 : créer un cas de test personnalisé](#)
- [Étape 4 : ajouter un tag au nouveau scénario de test](#)
- [Étape 5 : créer le fichier JAR de test](#)
- [Étape 6 : Exécuter le test](#)
- [Exemple : création d'un scénario de test personnalisé](#)

Prérequis

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

- GDK version 1.3.0 ou ultérieure
- Java
- Maven
- Git

Étape 1 : Initialisation d'un projet GDK avec un composant

- Initialisez un dossier vide avec un projet GDK. Téléchargez le HelloWorld composant implémenté en Python en exécutant la commande suivante.

```
gdk component init -t HelloWorld -l python -n HelloWorld
```

Cette commande crée un nouveau répertoire nommé HelloWorld dans le répertoire en cours.

Étape 2 : Initialisation d'un projet GDK avec un module de test end-to-end

- GDK vous permet de télécharger le modèle du module de test composé d'une fonctionnalité et d'une implémentation par étapes. Exécutez la commande suivante pour ouvrir le HelloWorld répertoire et initialiser le projet GDK existant à l'aide d'un module de test.

```
cd HelloWorld  
gdk test-e2e init
```

Cette commande crée un nouveau répertoire nommé gg-e2e-tests dans le HelloWorld répertoire. Ce répertoire de test est un projet [Maven](#) qui dépend du JAR autonome de test Greengrass.

Étape 3 : créer un cas de test personnalisé

La rédaction d'un scénario de test personnalisé comprend généralement deux étapes : créer un fichier de fonctionnalités avec un scénario de test et implémenter les définitions d'étapes. Pour un exemple de création d'un scénario de test personnalisé, voir [Exemple : création d'un scénario de test personnalisé](#). Suivez les étapes suivantes pour créer votre scénario de test personnalisé :

1. Création d'un fichier de fonctionnalités avec un scénario de test

Une fonctionnalité décrit généralement une fonctionnalité spécifique du logiciel testé. Dans Cucumber, chaque fonctionnalité est spécifiée sous la forme d'un fichier de fonctionnalités individuel avec un titre, une description détaillée et un ou plusieurs exemples de cas spécifiques appelés scénarios. Chaque scénario comprend un titre, une description détaillée et une série d'étapes qui définissent les interactions et les résultats attendus. Les scénarios sont rédigés dans un format structuré à l'aide des mots clés « donné », « quand » et « alors ».

2. Mettre en œuvre les définitions des étapes

Une définition d'étape lie l'[étape Gherkin](#) en langage clair au code programmatique. Lorsque Cucumber identifie une étape Gherkin dans un scénario, il recherche une définition d'étape correspondante à exécuter.

Étape 4 : ajouter un tag au nouveau scénario de test

- Vous pouvez attribuer des balises aux fonctionnalités et aux scénarios afin d'organiser le processus de test. Vous pouvez utiliser des balises pour classer les sous-ensembles de scénarios et également sélectionner les hooks à exécuter de manière conditionnelle. Les fonctionnalités et les scénarios peuvent comporter plusieurs balises séparées par un espace.

Dans cet exemple, nous utilisons le HelloWorld composant.

Dans le fichier de fonctionnalités, ajoutez une nouvelle balise nommée @HelloWorld à côté de la @Sample balise.

```
@Sample @HelloWorld
Scenario: As a developer, I can create a component and deploy it on my device
....
```

Étape 5 : créer le fichier JAR de test

- Construisez le composant. Vous devez créer le composant avant de créer le module de test.

```
gdk component build
```

- Créez le module de test à l'aide de la commande suivante. Cette commande créera le fichier JAR de test dans le greengrass-build dossier.

```
gdk test-e2e build
```

Étape 6 : Exécuter le test

Lorsque vous exécutez un scénario de test personnalisé, le GTF automatise le cycle de vie du test ainsi que la gestion des ressources créées pendant le test. Il approvisionne d'abord un appareil en cours de test (DUT) en tant qu' AWS IoT objet et y installe le logiciel de base Greengrass. Il créera ensuite un nouveau composant nommé HelloWorld en utilisant la recette spécifiée dans ce chemin. Le HelloWorld composant est ensuite déployé sur le périphérique principal par le biais d'un déploiement d'objets Greengrass. Il sera ensuite vérifié si le déploiement est réussi. L'état du déploiement passera à 3 COMPLETED minutes si le déploiement est réussi.

1. Accédez au `gdk-config.json` fichier dans le répertoire du projet pour cibler les tests avec la HelloWorld balise. Mettez à jour la `test-e2e` clé à l'aide de la commande suivante.

```
"test-e2e":{
  "gtf_options" : {
    "tags":"HelloWorld"
  }
}
```

2. Avant d'exécuter les tests, vous devez fournir des AWS informations d'identification au périphérique hôte. GTF utilise ces informations d'identification pour gérer les AWS ressources pendant le processus de test. Assurez-vous que le rôle que vous fournissez dispose des autorisations nécessaires pour automatiser les opérations nécessaires incluses dans le test.

Exécutez les commandes suivantes pour fournir les AWS informations d'identification.

- Linux or Unix

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

Windows Command Prompt (CMD)

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

PowerShell

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

3. Exécutez le test à l'aide de la commande suivante.

```
gdk test-e2e run
```

Cette commande télécharge la dernière version du noyau Greengrass dans le greengrass-build dossier et exécute des tests en l'utilisant. Cette commande cible également uniquement les scénarios dotés de la HelloWorld balise et génère un rapport pour ces scénarios. Vous verrez que les AWS ressources créées lors de ce test sont supprimées à la fin du test.

Exemple : création d'un scénario de test personnalisé

Exemple

Le module de test téléchargé dans le projet GDK se compose d'un exemple de fonctionnalité et d'un fichier d'implémentation des étapes.

Dans l'exemple suivant, nous créons un fichier de fonctionnalités pour tester la fonctionnalité de déploiement d'objets du logiciel Greengrass. Nous testons partiellement la fonctionnalité de cette fonctionnalité avec un scénario qui effectue le déploiement d'un composant via Greengrass AWS Cloud. Il s'agit d'une série d'étapes qui nous aident à comprendre les interactions et les résultats attendus de ce cas d'utilisation.

1. Création d'un fichier de fonctionnalités

Accédez au `gg-e2e-tests/src/main/resources/greengrass/features` dossier dans le répertoire actuel. Vous pouvez trouver l'exemple `component.feature` qui ressemble à l'exemple suivant.

Dans ce fichier de fonctionnalités, vous pouvez tester la fonctionnalité de déploiement d'objets du logiciel Greengrass. Vous pouvez tester partiellement la fonctionnalité de cette fonctionnalité avec un scénario qui effectue le déploiement d'un composant via le cloud Greengrass. Le scénario est une série d'étapes qui aident à comprendre les interactions et les résultats attendus de ce cas d'utilisation.

Feature: Testing features of Greengrassv2 component

Background:

Given my device is registered as a Thing
And my device is running Greengrass

@Sample

Scenario: As a developer, I can create a component and deploy it on my device

When I create a Greengrass deployment with components

HelloWorld | /path/to/recipe/file

And I deploy the Greengrass deployment configuration

Then the Greengrass deployment is COMPLETED on the device after 180 seconds

And I call my custom step

GTF contient les définitions des étapes de toutes les étapes suivantes, à l'exception de l'étape nommée :And I call my custom step.

2. Mettre en œuvre les définitions des étapes

Le fichier JAR autonome GTF contient les définitions d'étapes de toutes les étapes à l'exception d'une étape :. And I call my custom step Vous pouvez implémenter cette étape dans le module de test.

Accédez au code source du fichier de test. Vous pouvez lier votre étape personnalisée à l'aide d'une définition d'étape à l'aide de la commande suivante.

```
@And("I call my custom step")
public void customStep() {
    System.out.println("My custom step was called ");
}
```

Tutoriel : Utiliser un test de confiance issu de la suite de tests de confiance

AWS IoT Greengrass Le Testing Framework (GTF) et le Greengrass Development Kit (GDK) offrent aux développeurs des moyens d'exécuter des tests. end-to-end Vous pouvez suivre ce didacticiel pour initialiser un projet GDK avec un composant, initialiser un projet GDK avec un module de test et utiliser un end-to-end test de confiance issu de la suite de tests de confiance. Après avoir créé votre scénario de test personnalisé, vous pouvez exécuter le test.

Un test de confiance est un test générique fourni par Greengrass qui valide les comportements fondamentaux des composants. Ces tests peuvent être modifiés ou étendus pour répondre à des besoins de composants plus spécifiques.

Pour ce didacticiel, nous utiliserons un HelloWorld composant. Si vous utilisez un autre composant, remplacez-le par le HelloWorld vôtre.

Dans ce didacticiel, vous allez effectuer les opérations suivantes :

1. Initialisez un projet GDK avec un composant.
2. Initialisez un projet GDK avec un module de end-to-end test.
3. Utilisez un test issu de la suite de tests de confiance.
4. Ajoutez une étiquette au nouveau scénario de test.
5. Créez le fichier JAR de test.
6. Exécutez le test .

Rubriques

- [Prérequis](#)
- [Étape 1 : Initialisation d'un projet GDK avec un composant](#)
- [Étape 2 : Initialisation d'un projet GDK avec un module de test end-to-end](#)
- [Étape 3 : Utiliser un test issu de la suite de tests de confiance](#)
- [Étape 4 : ajouter un tag au nouveau scénario de test](#)
- [Étape 5 : créer le fichier JAR de test](#)
- [Étape 6 : Exécuter le test](#)
- [Exemple : utiliser un test de confiance](#)

Prérequis

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

- GDK version 1.6.0 ou ultérieure
- Java
- Maven
- Git

Étape 1 : Initialisation d'un projet GDK avec un composant

- Initialisez un dossier vide avec un projet GDK. Téléchargez le HelloWorld composant implémenté en Python en exécutant la commande suivante.

```
gdk component init -t HelloWorld -l python -n HelloWorld
```

Cette commande crée un nouveau répertoire nommé HelloWorld dans le répertoire en cours.

Étape 2 : Initialisation d'un projet GDK avec un module de test end-to-end

- GDK vous permet de télécharger le modèle du module de test composé d'une fonctionnalité et d'une implémentation par étapes. Exécutez la commande suivante pour ouvrir le HelloWorld répertoire et initialiser le projet GDK existant à l'aide d'un module de test.

```
cd HelloWorld
gdk test-e2e init
```

Cette commande crée un nouveau répertoire nommé gg-e2e-tests dans le HelloWorld répertoire. Ce répertoire de test est un projet [Maven](#) qui dépend du JAR autonome de test Greengrass.

Étape 3 : Utiliser un test issu de la suite de tests de confiance

La rédaction d'un scénario de test de confiance consiste à utiliser le fichier de fonctionnalités fourni et, si nécessaire, à modifier les scénarios. Pour un exemple d'utilisation d'un test de confiance, voir [Exemple : création d'un scénario de test personnalisé](#). Pour utiliser un test de confiance, procédez comme suit :

- Utilisez le fichier de fonctionnalités fourni.

Accédez au gg-e2e-tests/src/main/resources/greengrass/features dossier dans le répertoire actuel. Ouvrez le confidenceTest.feature fichier d'exemple pour utiliser le test de confiance.

Étape 4 : ajouter un tag au nouveau scénario de test

- Vous pouvez attribuer des balises aux fonctionnalités et aux scénarios afin d'organiser le processus de test. Vous pouvez utiliser des balises pour classer les sous-ensembles de scénarios et également sélectionner les hooks à exécuter de manière conditionnelle. Les fonctionnalités et les scénarios peuvent comporter plusieurs balises séparées par un espace.

Dans cet exemple, nous utilisons le HelloWorld composant.

Chaque scénario est étiqueté avec `@ConfidenceTest`. Modifiez ou ajoutez des balises si vous souhaitez exécuter uniquement un sous-ensemble de la suite de tests. Chaque scénario de test est décrit en haut de chaque test de confiance. Le scénario est une série d'étapes qui aident à comprendre les interactions et les résultats attendus de chaque cas de test. Vous pouvez étendre ces tests en ajoutant vos propres étapes ou en modifiant les étapes existantes.

```
@ConfidenceTest
Scenario: As a Developer, I can deploy GDK_COMPONENT_NAME to my device and see it
         is working as expected
....
```

Étape 5 : créer le fichier JAR de test

- Construisez le composant. Vous devez créer le composant avant de créer le module de test.

```
gdk component build
```

- Créez le module de test à l'aide de la commande suivante. Cette commande créera le fichier JAR de test dans le greengrass-build dossier.

```
gdk test-e2e build
```

Étape 6 : Exécuter le test

Lorsque vous exécutez un test de confiance, le GTF automatise le cycle de vie du test ainsi que la gestion des ressources créées pendant le test. Il approvisionne d'abord un appareil en cours de test (DUT) en tant qu' AWS IoT objet et y installe le logiciel de base Greengrass. Il créera ensuite un nouveau composant nommé HelloWorld en utilisant la recette spécifiée dans ce chemin. Le HelloWorld composant est ensuite déployé sur le périphérique principal par le biais d'un

déploiement d'objets Greengrass. Il sera ensuite vérifié si le déploiement est réussi. L'état du déploiement passera à 3 COMPLETED minutes si le déploiement est réussi.

1. Accédez au `gdk-config.json` fichier dans le répertoire du projet pour cibler les tests avec le `ConfidenceTest` tag ou le tag `yo8u` spécifié à l'étape 4. Mettez à jour la `test-e2e` clé à l'aide de la commande suivante.

```
"test-e2e":{
  "gtf_options" : {
    "tags":"ConfidenceTest"
  }
}
```

2. Avant d'exécuter les tests, vous devez fournir des AWS informations d'identification au périphérique hôte. GTF utilise ces informations d'identification pour gérer les AWS ressources pendant le processus de test. Assurez-vous que le rôle que vous fournissez dispose des autorisations nécessaires pour automatiser les opérations nécessaires incluses dans le test.

Exécutez les commandes suivantes pour fournir les AWS informations d'identification.

- Linux or Unix

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

Windows Command Prompt (CMD)

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

PowerShell

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

3. Exécutez le test à l'aide de la commande suivante.

```
gdk test-e2e run
```

Cette commande télécharge la dernière version du noyau Greengrass dans le greengrass-build dossier et exécute des tests en l'utilisant. Cette commande cible également uniquement les scénarios dotés de la ConfidenceTest balise et génère un rapport pour ces scénarios. Vous verrez que les AWS ressources créées lors de ce test sont supprimées à la fin du test.

Exemple : utiliser un test de confiance

Exemple

Le module de test téléchargé dans le projet GDK consiste en un fichier de fonctionnalités fourni.

Dans l'exemple suivant, nous utilisons un fichier de fonctionnalités pour tester la fonctionnalité de déploiement d'objets du logiciel Greengrass. Nous testons partiellement la fonctionnalité de cette fonctionnalité avec un scénario qui effectue le déploiement d'un composant via Greengrass AWS Cloud. Il s'agit d'une série d'étapes qui nous aident à comprendre les interactions et les résultats attendus de ce cas d'utilisation.

- Utilisez le fichier de fonctionnalités fourni.

Accédez au `gg-e2e-tests/src/main/resources/greengrass/features` dossier dans le répertoire actuel. Vous pouvez trouver l'exemple `confidenceTest.feature` qui ressemble à l'exemple suivant.

```
Feature: Confidence Test Suite
```

```
Background:
```

```
    Given my device is registered as a Thing
    And my device is running Greengrass
```

```
@ConfidenceTest
```

```
Scenario: As a Developer, I can deploy GDK_COMPONENT_NAME to my device and see it
is working as expected
```

```
    When I create a Greengrass deployment with components
```

```
        | GDK_COMPONENT_NAME | GDK_COMPONENT_RECIPE_FILE |
        | aws.greengrass.Cli | LATEST |
```

```
    And I deploy the Greengrass deployment configuration
```

```
    Then the Greengrass deployment is COMPLETED on the device after 180 seconds
```

```
    # Update component state accordingly. Possible states: {RUNNING, FINISHED,
    BROKEN, STOPPING}
```

```
And I verify the GDK_COMPONENT_NAME component is RUNNING using the greengrass-  
cli
```

Chaque scénario de test est décrit en haut de chaque test de confiance. Le scénario est une série d'étapes qui aident à comprendre les interactions et les résultats attendus de chaque cas de test. Vous pouvez étendre ces tests en ajoutant vos propres étapes ou en modifiant les étapes existantes. Chacun des scénarios inclut des commentaires qui vous aideront à effectuer ces ajustements.

Développer des AWS IoT Greengrass composants

Vous pouvez développer et tester des composants sur votre appareil principal Greengrass. Par conséquent, vous pouvez créer et itérer votre AWS IoT Greengrass logiciel sans interagir avec le AWS Cloud. Lorsque vous avez terminé une version de votre composant, vous pouvez la télécharger AWS IoT Greengrass dans le cloud, afin que vous et votre équipe puissiez déployer le composant sur d'autres appareils de votre flotte. Pour plus d'informations sur le déploiement des composants, consultez [Déployer AWS IoT Greengrass des composants sur des appareils](#).

Chaque composant est composé d'une recette et d'artefacts.

- Recettes

Chaque composant contient un fichier de recette qui définit ses métadonnées. La recette spécifie également les paramètres de configuration, les dépendances des composants, le cycle de vie et la compatibilité de la plate-forme du composant. Le cycle de vie du composant définit les commandes qui installent, exécutent et arrêtent le composant. Pour de plus amples informations, veuillez consulter [AWS IoT Greengrass référence de recette de composant](#).

Vous pouvez définir des recettes au format [JSON](#) ou [YAML](#).

- Artefacts

Les composants peuvent avoir un nombre illimité d'artefacts, qui sont des binaires de composants. Les artefacts peuvent inclure des scripts, du code compilé, des ressources statiques et tout autre fichier consommé par un composant. Les composants peuvent également consommer des artefacts issus de leurs dépendances.

AWS IoT Greengrass fournit des composants prédéfinis que vous pouvez utiliser dans vos applications et déployer sur vos appareils. Par exemple, vous pouvez utiliser le composant Stream

Manager pour télécharger des données vers différents AWS services, ou vous pouvez utiliser le composant CloudWatch Metrics pour publier des métriques personnalisées sur Amazon CloudWatch. Pour de plus amples informations, veuillez consulter [AWS-composants fournis](#).

AWS IoT Greengrass organise un index des composants de Greengrass, appelé Greengrass Software Catalog. Ce catalogue suit les composants de Greengrass développés par la communauté Greengrass. À partir de ce catalogue, vous pouvez télécharger, modifier et déployer des composants pour créer vos applications Greengrass. Pour de plus amples informations, veuillez consulter [Composantes communautaires](#).

Le logiciel AWS IoT Greengrass Core exécute les composants en tant qu'utilisateur et groupe du système, tels que `ggc_user` et `ggc_group`, que vous configurez sur le périphérique principal. Cela signifie que les composants disposent des autorisations de cet utilisateur du système. Si vous utilisez un utilisateur système sans répertoire de base, les composants ne peuvent pas utiliser de commandes d'exécution ou de code utilisant un répertoire de base. Cela signifie que vous ne pouvez pas utiliser la `pip install some-library --user` commande pour installer des packages Python par exemple. Si vous avez suivi le [didacticiel de démarrage](#) pour configurer votre appareil principal, cela signifie que l'utilisateur de votre système n'a pas de répertoire personnel. Pour plus d'informations sur la configuration de l'utilisateur et du groupe qui exécutent les composants, consultez [Configurer l'utilisateur qui exécute les composants](#).

Note

AWS IoT Greengrass utilise des versions sémantiques pour les composants. Les versions sémantiques suivent une majeure. mineur. système de numéro de patch. Par exemple, la version `1.0.0` représente la première version majeure d'un composant. Pour plus d'informations, consultez la [spécification de version sémantique](#).

Rubriques

- [Cycle de vie des composants](#)
- [Types de composants](#)
- [Création de AWS IoT Greengrass composants](#)
- [Testez AWS IoT Greengrass les composants avec des déploiements locaux](#)
- [Publiez des composants à déployer sur vos appareils principaux](#)
- [Interagissez avec les AWS services](#)

- [Exécuter un conteneur Docker](#)
- [AWS IoT Greengrass référence de recette de composant](#)
- [Référence à la variable d'environnement du composant](#)

Cycle de vie des composants

Le cycle de vie des composants définit les étapes que le logiciel AWS IoT Greengrass Core utilise pour installer et exécuter les composants. Chaque étape définit un script et d'autres informations qui indiquent le comportement du composant. Par exemple, lorsque vous installez un composant, le logiciel AWS IoT Greengrass Core exécute le script de `install` cycle de vie de ce composant. Les composants des équipements principaux présentent les états de cycle de vie suivants :

- NEW— La recette et les artefacts du composant sont chargés sur le périphérique principal, mais le composant n'est pas installé. Une fois qu'un composant a atteint cet état, il exécute son [script d'installation](#).
- INSTALLED— Le composant est installé sur le périphérique principal. Le composant passe dans cet état après avoir exécuté son [script d'installation](#).
- STARTING— Le composant démarre sur le périphérique principal. Le composant entre dans cet état lorsqu'il exécute son [script de démarrage](#). Si le démarrage réussit, le composant passe à l'RUNNING état.
- RUNNING— Le composant est en cours d'exécution sur le périphérique principal. Le composant entre dans cet état lorsqu'il exécute son [script d'exécution](#) ou lorsque des processus en arrière-plan sont actifs depuis son script de démarrage.
- FINISHED— Le composant s'est exécuté correctement et a terminé son exécution.
- STOPPING— Le composant est en train de s'arrêter. Le composant entre dans cet état lorsqu'il exécute son [script d'arrêt](#).
- ERRORED— Le composant a rencontré une erreur. Lorsque le composant entre dans cet état, il exécute son [script de restauration](#). Ensuite, le composant redémarre pour essayer de revenir à une utilisation normale. Si le composant entre trois fois dans ERRORED cet état sans succès, le composant devient BROKEN.
- BROKEN— Le composant a rencontré des erreurs à plusieurs reprises et ne peut pas être rétabli. Vous devez déployer à nouveau le composant pour le réparer.

Types de composants

Le type de composant indique comment le logiciel AWS IoT Greengrass Core exécute le composant. Les types de composants peuvent être les suivants :

- Noyau (`aws.greengrass.nucleus`)

Le noyau Greengrass est le composant qui fournit les fonctionnalités minimales du logiciel AWS IoT Greengrass Core. Pour de plus amples informations, veuillez consulter [Noyau de Greengrass](#).

- Plug-in (`aws.greengrass.plugin`)

Le noyau Greengrass exécute un composant de plug-in dans la même machine virtuelle Java (JVM) que le noyau. Le noyau redémarre lorsque vous modifiez la version d'un composant du plugin sur un périphérique principal. Pour installer et exécuter les composants du plugin, vous devez configurer le noyau Greengrass pour qu'il s'exécute en tant que service système. Pour de plus amples informations, veuillez consulter [Configurer le noyau Greengrass en tant que service système](#).

Plusieurs composants fournis par AWS sont des composants de plug-in, ce qui leur permet de s'interfacer directement avec le noyau de Greengrass. Les composants du plugin utilisent le même fichier journal que le noyau Greengrass. Pour de plus amples informations, veuillez consulter [AWS IoT Greengrass Journaux de surveillance](#).

- Générique (`aws.greengrass.generic`)

Le noyau Greengrass exécute les scripts de cycle de vie d'un composant générique, si le composant définit un cycle de vie.

Ce type est le type par défaut pour les composants personnalisés.

- Lambda (`aws.greengrass.lambda`)

[Le noyau Greengrass exécute un composant de fonction Lambda à l'aide du composant de lancement Lambda.](#)

Lorsque vous créez un composant à partir d'une fonction Lambda, le composant est de ce type. Pour de plus amples informations, veuillez consulter [Exécuter AWS Lambda des fonctions](#).

 Note

Il est déconseillé de spécifier le type de composant dans une recette. AWS IoT Greengrass définit le type pour vous lorsque vous créez un composant.

Création de AWS IoT Greengrass composants

Vous pouvez développer des AWS IoT Greengrass composants personnalisés sur un ordinateur de développement local ou un appareil principal de Greengrass. AWS IoT Greengrass fournit [l'interface de ligne de commande du kit de AWS IoT Greengrass développement \(CLI GDK\)](#) pour vous aider à créer, créer et publier des composants à partir de modèles de composants prédéfinis [et](#) de composants communautaires. Vous pouvez également exécuter des commandes shell intégrées pour créer, créer et publier des composants. Choisissez l'une des options suivantes pour créer des composants Greengrass personnalisés :

- Utiliser la CLI du kit de développement Greengrass

Utilisez la CLI GDK pour développer des composants sur un ordinateur de développement local. La CLI GDK crée et empaquète le code source des composants dans une recette et des artefacts que vous pouvez publier en tant que composant privé du AWS IoT Greengrass service. Vous pouvez configurer la CLI GDK pour mettre à jour automatiquement la version et l'artefact du composant URIs lorsque vous publiez le composant, de sorte que vous n'avez pas besoin de mettre à jour la recette à chaque fois. Pour développer un composant à l'aide de la CLI GDK, vous pouvez partir d'un modèle ou d'un composant communautaire du catalogue de logiciels [Greengrass](#). Pour de plus amples informations, veuillez consulter [AWS IoT Greengrass Interface de ligne de commande du kit de développement](#).

- Exécuter des commandes shell intégrées

Vous pouvez exécuter des commandes shell intégrées pour développer des composants sur un ordinateur de développement local ou sur un périphérique principal Greengrass. Vous utilisez des commandes shell pour copier ou intégrer le code source des composants dans des artefacts. Chaque fois que vous créez une nouvelle version d'un composant, vous devez créer ou mettre à jour la recette avec la nouvelle version du composant. Lorsque vous publiez le composant sur le AWS IoT Greengrass service, vous devez mettre à jour l'URI de chaque artefact de composant de la recette.

Rubriques

- [Création d'un composant \(CLI GDK\)](#)
- [Création d'un composant \(commandes shell\)](#)

Création d'un composant (CLI GDK)

Suivez les instructions de cette section pour créer et créer un composant à l'aide de la CLI GDK.

Pour développer un composant Greengrass (GDK CLI)

1. Si ce n'est pas déjà fait, installez la CLI GDK sur votre ordinateur de développement. Pour de plus amples informations, veuillez consulter [Installation ou mise à jour de l'interface de ligne de commande AWS IoT Greengrass du kit de développement](#).
2. Accédez au dossier dans lequel vous souhaitez créer des dossiers de composants.

Linux or Unix

```
mkdir ~/greengrassv2  
cd ~/greengrassv2
```

Windows Command Prompt (CMD)

```
mkdir %USERPROFILE%\greengrassv2  
cd %USERPROFILE%\greengrassv2
```

PowerShell

```
mkdir ~/greengrassv2  
cd ~/greengrassv2
```

3. Choisissez un modèle de composant ou un composant de communauté à télécharger. La CLI GDK télécharge le modèle ou le composant communautaire, afin que vous puissiez partir d'un exemple fonctionnel. Utilisez la commande [component list](#) pour récupérer la liste des modèles ou des composants communautaires disponibles.
 - Pour répertorier les modèles de composants, exécutez la commande suivante. Chaque ligne de la réponse inclut le nom du modèle et le langage de programmation.

```
gdk component list --template
```

- Pour répertorier les composants de la communauté, exécutez la commande suivante.

```
gdk component list --repository
```

4. Créez et modifiez un dossier de composants dans lequel la CLI GDK télécharge le modèle ou le composant communautaire. *HelloWorld* Remplacez-le par le nom du composant ou par un autre nom permettant d'identifier ce dossier de composants.

Linux or Unix

```
mkdir HelloWorld  
cd HelloWorld
```

Windows Command Prompt (CMD)

```
mkdir HelloWorld  
cd HelloWorld
```

PowerShell

```
mkdir HelloWorld  
cd HelloWorld
```

5. Téléchargez le modèle ou le composant communautaire dans le dossier actuel. Utilisez la commande [component init](#).
 - Pour créer un dossier de composants à partir d'un modèle, exécutez la commande suivante. Remplacez *HelloWorld* par le nom du modèle et remplacez *python* par le nom du langage de programmation.

```
gdk component init --template HelloWorld --language python
```

- Pour créer un dossier de composants à partir d'un composant communautaire, exécutez la commande suivante. Remplacez *ComponentName* par le nom du composant communautaire.

```
gdk component init --repository ComponentName
```

 Note

Si vous utilisez GDK CLI v1.0.0, vous devez exécuter cette commande dans un dossier vide. La CLI GDK télécharge le modèle ou le composant communautaire dans le dossier actuel.

Si vous utilisez la CLI GDK v1.1.0 ou une version ultérieure, vous pouvez spécifier l'--nameargument pour spécifier le dossier dans lequel la CLI GDK télécharge le modèle ou le composant communautaire. Si vous utilisez cet argument, spécifiez un dossier qui n'existe pas. La CLI GDK crée le dossier pour vous. Si vous ne spécifiez pas cet argument, la CLI GDK utilise le dossier actuel, qui doit être vide.

6. La CLI GDK lit le [fichier de configuration de la CLI GDK](#), nommé `gdk-config.json`, pour créer et publier des composants. Ce fichier de configuration se trouve à la racine du dossier du composant. L'étape précédente crée ce fichier pour vous. Au cours de cette étape, vous effectuez `gdk-config.json` une mise à jour avec les informations relatives à votre composant. Procédez comme suit :
 - a. Ouvrez `gdk-config.json` dans un éditeur de texte.
 - b. (Facultatif) Modifiez le nom du composant. Le nom du composant est la clé de l'`component` objet.
 - c. Changez l'auteur du composant.
 - d. (Facultatif) Modifiez la version du composant. Spécifiez l'un des éléments suivants :
 - **NEXT_PATCH**— Lorsque vous choisissez cette option, la CLI GDK définit la version lorsque vous publiez le composant. La CLI GDK interroge le AWS IoT Greengrass service pour identifier la dernière version publiée du composant. Ensuite, il définit la version sur la version de correctif suivante après cette version. Si vous n'avez jamais publié le composant auparavant, la CLI GDK utilise la version `1.0.0`.
- Si vous choisissez cette option, vous ne pouvez pas utiliser la [CLI Greengrass](#) pour déployer et tester localement le composant sur votre ordinateur de développement local qui exécute le logiciel AWS IoT Greengrass Core. Pour activer les déploiements locaux, vous devez plutôt spécifier une version sémantique.
- Une version sémantique, telle que `1.0.0`. Les versions sémantiques utilisent une majeure. mineur. système de numérotation des patches. Pour plus d'informations, consultez la [spécification de version sémantique](#).

Si vous développez des composants sur un appareil principal Greengrass sur lequel vous souhaitez les déployer et les tester, choisissez cette option. Vous devez créer le composant avec une version spécifique pour créer des déploiements locaux avec la CLI [Greengrass](#).

- e. (Facultatif) Modifiez la configuration de construction du composant. La configuration de construction définit la manière dont la CLI GDK construit la source du composant en artefacts. Choisissez l'une des options suivantes pour `build_system` :
- `zip`— Regroupe le dossier du composant dans un fichier ZIP à définir comme seul artefact du composant. Choisissez cette option pour les types de composants suivants :
 - Composants utilisant des langages de programmation interprétés, tels que Python ou JavaScript.
 - Composants qui empaquetent des fichiers autres que du code, tels que des modèles d'apprentissage automatique ou d'autres ressources.

La CLI GDK compresse le dossier du composant dans un fichier zip portant le même nom que le dossier du composant. Par exemple, si le nom du dossier du composant est `HelloWorld`, la CLI GDK crée un fichier zip nommé `HelloWorld.zip`.

 Note

Si vous utilisez la version 1.0.0 de la CLI GDK sur un appareil Windows, les noms des dossiers de composants et des fichiers zip ne doivent contenir que des lettres minuscules.

Lorsque la CLI GDK comprime le dossier du composant dans un fichier zip, elle ignore les fichiers suivants :

- le fichier `gdk-config.json` ;
- Le fichier de recette (`recipe.json` ou `recipe.yaml`)
- Créez des dossiers, tels que `greengrass-build`
- `maven`— Exécute la `mvn clean package` commande pour intégrer la source du composant en artefacts. Choisissez cette option pour les composants qui utilisent [Maven](#), tels que les composants Java.

Sur les appareils Windows, cette fonctionnalité est disponible pour GDK CLI v1.1.0 et versions ultérieures.

- **gradle**— Exécute la `gradle build` commande pour intégrer la source du composant en artefacts. Choisissez cette option pour les composants qui utilisent [Gradle](#). Cette fonctionnalité est disponible pour GDK CLI v1.1.0 et versions ultérieures.

Le système de `gradle` construction prend en charge Kotlin DSL en tant que fichier de construction. Cette fonctionnalité est disponible pour GDK CLI v1.2.0 et versions ultérieures.

- **gradlew**— Exécute la `gradlew` commande pour intégrer la source du composant en artefacts. Choisissez cette option pour les composants qui utilisent le [Gradle Wrapper](#).

Cette fonctionnalité est disponible pour GDK CLI v1.2.0 et versions ultérieures.

- **custom**— Exécute une commande personnalisée pour intégrer la source du composant dans une recette et des artefacts. Spécifiez la commande personnalisée dans le `custom_build_command` paramètre.
- f. Si vous spécifiez `custom` pour `build_system`, ajoutez-le `custom_build_command` à l'objet `build`. Dans `custom_build_command`, spécifiez une chaîne unique ou une liste de chaînes, chaque chaîne étant un mot dans la commande. Par exemple, pour exécuter une commande de génération personnalisée pour un composant C++, vous pouvez spécifier `["cmake", "--build", "build", "--config", "Release"]`.
- g. Si vous utilisez la CLI GDK v1.1.0 ou une version ultérieure, vous pouvez spécifier l'argument `bucket` pour spécifier le compartiment S3 dans lequel la CLI GDK télécharge les artefacts du composant. Si vous ne spécifiez pas cet argument, la CLI GDK télécharge vers le compartiment S3 dont le nom est `bucket-region-accountId`, où `bucket` et `region` sont les valeurs que vous spécifiez dans `gdk-config.json`, ainsi que votre `accountId` Compte AWS identifiant. La CLI GDK crée le bucket s'il n'existe pas.

Modifiez la configuration de publication du composant. Procédez comme suit :

- i. Spécifiez le nom du compartiment S3 à utiliser pour héberger les artefacts des composants.
- ii. Spécifiez l' Région AWS endroit où la CLI GDK publie le composant.

Lorsque vous avez terminé cette étape, le `gdk-config.json` fichier peut ressembler à l'exemple suivant.

```
{
  "component": {
    "com.example.PythonHelloWorld": {
      "author": "Amazon",
      "version": "NEXT_PATCH",
      "build": {
        "build_system" : "zip"
      },
      "publish": {
        "bucket": "greengrass-component-artifacts",
        "region": "us-west-2"
      }
    }
  },
  "gdk_version": "1.0.0"
}
```

7. Mettez à jour le fichier de recette du composant, nommé `recipe.yaml` ou `recipe.json`. Procédez comme suit :
 - a. Si vous avez téléchargé un modèle ou un composant communautaire qui utilise le système de zip compilation, vérifiez que le nom de l'artefact zip correspond au nom du dossier du composant. La CLI GDK compresse le dossier des composants dans un fichier zip portant le même nom que le dossier des composants. La recette contient le nom de l'artefact zip dans la liste des artefacts du composant et dans les scripts de cycle de vie qui utilisent les fichiers de l'artefact zip. Mettez à jour les Lifecycle définitions Artifacts et de manière à ce que le nom du fichier zip corresponde au nom du dossier du composant. Les exemples de recettes partiels suivants mettent en évidence le nom du fichier zip dans les Lifecycle définitions Artifacts et.

JSON

```
{
  ...
  "Manifests": [
    {
      "Platform": {
```

```

    "os": "all"
  },
  "Artifacts": [
    {
      "URI": "s3://{COMPONENT_NAME}/{COMPONENT_VERSION}/HelloWorld.zip",
      "Unarchive": "ZIP"
    }
  ],
  "Lifecycle": {
    "Run": "python3 -u {artifacts:decompressedPath}/HelloWorld/main.py
{configuration:/Message}"
  }
}
]
}

```

YAML

```

---
...
Manifests:
  - Platform:
      os: all
    Artifacts:
      - URI: "s3://{BUCKET_NAME}/COMPONENT_NAME/
COMPONENT_VERSION/HelloWorld.zip"
        Unarchive: ZIP
    Lifecycle:
      Run: "python3 -u {artifacts:decompressedPath}/HelloWorld/main.py
{configuration:/Message}"

```

- b. (Facultatif) Mettez à jour la description du composant, la configuration par défaut, les artefacts, les scripts de cycle de vie et le support de la plate-forme. Pour de plus amples informations, veuillez consulter [AWS IoT Greengrass référence de recette de composant](#).

Lorsque vous avez terminé cette étape, le fichier de recette peut ressembler aux exemples suivants.

JSON

```

{
  "RecipeFormatVersion": "2020-01-25",

```

```

"ComponentName": "{COMPONENT_NAME}",
"ComponentVersion": "{COMPONENT_VERSION}",
"ComponentDescription": "This is a simple Hello World component written in
Python.",
"ComponentPublisher": "{COMPONENT_AUTHOR}",
"ComponentConfiguration": {
  "DefaultConfiguration": {
    "Message": "World"
  }
},
"Manifests": [
  {
    "Platform": {
      "os": "all"
    },
    "Artifacts": [
      {
        "URI": "s3://{COMPONENT_NAME}/{COMPONENT_VERSION}/HelloWorld.zip",
        "Unarchive": "ZIP"
      }
    ],
    "Lifecycle": {
      "Run": "python3 -u {artifacts:decompressedPath}/HelloWorld/main.py
{configuration:/Message}"
    }
  }
]
}

```

YAML

```

---
RecipeFormatVersion: "2020-01-25"
ComponentName: "{COMPONENT_NAME}"
ComponentVersion: "{COMPONENT_VERSION}"
ComponentDescription: "This is a simple Hello World component written in
Python."
ComponentPublisher: "{COMPONENT_AUTHOR}"
ComponentConfiguration:
  DefaultConfiguration:
    Message: "World"
Manifests:
  - Platform:

```



```
os: all
Artifacts:
  - URI: "s3://BUCKET_NAME/COMPONENT_NAME/COMPONENT_VERSION/HelloWorld.zip"
    Unarchive: ZIP
Lifecycle:
  Run: "python3 -u {artifacts:decompressedPath}/HelloWorld/main.py
{configuration:/Message}"
```

8. Développez et construisez le composant Greengrass. La commande [component build](#) produit une recette et des artefacts dans le greengrass-build dossier du dossier des composants. Exécutez la commande suivante.

```
gdk component build
```

Lorsque vous êtes prêt à tester votre composant, utilisez la CLI GDK pour le publier sur le AWS IoT Greengrass service. Vous pouvez ensuite déployer le composant sur les appareils principaux de Greengrass. Pour de plus amples informations, veuillez consulter [Publiez des composants à déployer sur vos appareils principaux](#).

Création d'un composant (commandes shell)

Suivez les instructions de cette section pour créer des dossiers de recettes et d'artefacts contenant du code source et des artefacts pour plusieurs composants.

Pour développer un composant Greengrass (commandes shell)

1. Créez un dossier pour vos composants avec des sous-dossiers pour les recettes et les artefacts. Exécutez les commandes suivantes sur votre appareil principal Greengrass pour créer ces dossiers et accéder au dossier des composants. Remplacez *~/greengrassv2* ou *%USERPROFILE%\greengrassv2* par le chemin d'accès au dossier à utiliser pour le développement local.

Linux or Unix

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

Windows Command Prompt (CMD)

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

```
cd %USERPROFILE%\greengrassv2
```

PowerShell

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

2. Utilisez un éditeur de texte pour créer un fichier de recette qui définit les métadonnées, les paramètres, les dépendances, le cycle de vie et les capacités de la plateforme de votre composant. Incluez la version du composant dans le nom du fichier de recette afin de pouvoir identifier quelle recette reflète quelle version de composant. Vous pouvez choisir le format YAML ou JSON pour votre recette.

Par exemple, sur un système basé sur Linux, vous pouvez exécuter la commande suivante pour utiliser GNU nano pour créer le fichier.

JSON

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

YAML

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

Note

AWS IoT Greengrass utilise des versions sémantiques pour les composants. Les versions sémantiques suivent une majeure. mineur. système de numéro de patch. Par exemple, la version 1.0.0 représente la première version majeure d'un composant. Pour plus d'informations, consultez la [spécification de version sémantique](#).

3. Définissez la recette de votre composant. Pour de plus amples informations, veuillez consulter [AWS IoT Greengrass référence de recette de composant](#).

Votre recette peut ressembler à l'exemple de recette Hello World suivant.

JSON

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

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.HelloWorld
ComponentVersion: '1.0.0'
ComponentDescription: My first AWS IoT Greengrass component.
```

```
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    Message: world
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      run: |
        python3 -u {artifacts:path}/hello_world.py "{configuration:/Message}"
  - Platform:
      os: windows
    Lifecycle:
      run: |
        py -3 -u {artifacts:path}/hello_world.py "{configuration:/Message}"
```

Cette recette exécute un script Python Hello World, qui peut ressembler à l'exemple de script suivant.

```
import sys

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

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

4. Créez un dossier pour la version du composant à développer. Nous vous recommandons d'utiliser un dossier distinct pour les artefacts de chaque version de composant afin de pouvoir identifier les artefacts correspondant à chaque version de composant. Exécutez la commande suivante.

Linux or Unix

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

Windows Command Prompt (CMD)

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

PowerShell

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

Important

Vous devez utiliser le format suivant pour le chemin du dossier d'artefacts. Incluez le nom et la version du composant que vous spécifiez dans la recette.

```
artifacts/componentName/componentVersion/
```

5. Créez les artefacts de votre composant dans le dossier que vous avez créé à l'étape précédente. Les artefacts peuvent inclure des logiciels, des images et tout autre fichier binaire utilisé par votre composant.

Lorsque votre composant est prêt, [testez-le](#).

Testez AWS IoT Greengrass les composants avec des déploiements locaux

Si vous développez un composant Greengrass sur un appareil principal, vous pouvez créer un déploiement local pour l'installer et le tester. Suivez les étapes décrites dans cette section pour créer un déploiement local.

Si vous développez le composant sur un autre ordinateur, tel qu'un ordinateur de développement local, vous ne pouvez pas créer de déploiement local. Publiez plutôt le composant sur le AWS IoT Greengrass service afin de pouvoir le déployer sur les appareils principaux de Greengrass pour le tester. Pour plus d'informations, consultez [Publiez des composants à déployer sur vos appareils principaux](#) et [Déployer AWS IoT Greengrass des composants sur des appareils](#).

Pour tester un composant sur un appareil principal Greengrass

1. Le périphérique principal enregistre les événements tels que les mises à jour des composants. Vous pouvez consulter ce fichier journal pour détecter et résoudre les erreurs liées à votre composant, telles qu'une recette non valide. Ce fichier journal affiche également les messages que votre composant imprime en sortie standard (stdout). Nous vous recommandons d'ouvrir une session de terminal supplémentaire sur votre appareil principal pour observer les nouveaux

messages du journal en temps réel. Ouvrez une nouvelle session de terminal, par exemple via SSH, et exécutez la commande suivante pour afficher les journaux. Remplacez */greengrass/v2* par le chemin d'accès au dossier AWS IoT Greengrass racine.

Linux or Unix

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

PowerShell

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

Vous pouvez également consulter le fichier journal de votre composant.

Linux or Unix

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

PowerShell

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

2. Dans votre session de terminal d'origine, exécutez la commande suivante pour mettre à jour le périphérique principal avec votre composant. */greengrass/v2* Remplacez-le par le chemin d'accès au dossier AWS IoT Greengrass racine et remplacez-le *~/greengrassv2* par le chemin d'accès à votre dossier de développement local.

Linux or Unix

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

Windows Command Prompt (CMD)

```
C:\greengrass\v2\bin\greengrass-cli deployment create ^  
--recipeDir %USERPROFILE%\greengrassv2\recipes ^  
--artifactDir %USERPROFILE%\greengrassv2\artifacts ^
```

```
--merge "com.example.HelloWorld=1.0.0"
```

PowerShell

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

Note

Vous pouvez également utiliser la `greengrass-cli deployment create` commande pour définir la valeur des paramètres de configuration de votre composant. Pour de plus amples informations, veuillez consulter [créer](#).

3. Utilisez la `greengrass-cli deployment status` commande pour suivre la progression du déploiement de votre composant.

Unix or Linux

```
sudo /greengrass/v2/bin/greengrass-cli deployment status \
-i deployment-id
```

Windows Command Prompt (CMD)

```
C:\greengrass\v2\bin\greengrass-cli deployment status ^
-i deployment-id
```

PowerShell

```
C:\greengrass\v2\bin\greengrass-cli deployment status `
-i deployment-id
```

4. Testez votre composant lorsqu'il s'exécute sur l'appareil principal de Greengrass. Lorsque vous aurez terminé cette version de votre composant, vous pourrez le télécharger sur le AWS IoT Greengrass service. Ensuite, vous pouvez déployer le composant sur d'autres périphériques principaux. Pour de plus amples informations, veuillez consulter [Publiez des composants à déployer sur vos appareils principaux](#).

Publiez des composants à déployer sur vos appareils principaux

Après avoir créé ou terminé une version d'un composant, vous pouvez le publier sur le AWS IoT Greengrass service. Ensuite, vous pouvez le déployer sur les appareils principaux de Greengrass.

Si vous utilisez la CLI du kit de [développement Greengrass \(CLI GDK\)](#) [pour développer et créer un composant](#), vous [pouvez utiliser la CLI GDK](#) pour publier le composant sur le. AWS Cloud Sinon, [utilisez les commandes shell intégrées et le AWS CLI](#) pour publier le composant.

Vous pouvez également l'utiliser AWS CloudFormation pour créer des composants et d'autres AWS ressources à partir de modèles. Pour plus d'informations, voir [Qu'est-ce que c'est AWS CloudFormation ?](#) et [AWS::GreengrassV2::ComponentVersion](#) dans le guide de l'utilisateur AWS CloudFormation .

Rubriques

- [Publier un composant \(GDK CLI\)](#)
- [Publier un composant \(commandes shell\)](#)

Publier un composant (GDK CLI)

Suivez les instructions de cette section pour publier un composant à l'aide de la CLI GDK. La CLI GDK télécharge les artefacts de construction dans un compartiment S3, met à jour l'artefact URIs dans la recette et crée le composant à partir de la recette. Vous spécifiez le compartiment S3 et la région à utiliser dans le [fichier de configuration de la CLI GDK](#).

Si vous utilisez la CLI GDK v1.1.0 ou une version ultérieure, vous pouvez spécifier l'--bucket argument pour spécifier le compartiment S3 dans lequel la CLI GDK télécharge les artefacts du composant. Si vous ne spécifiez pas cet argument, la CLI GDK télécharge vers le compartiment S3 dont le nom est `bucket-region-accountId`, où `bucket` et `region` sont les valeurs que vous spécifiez dans `gdk-config.json`, ainsi que votre `accountId` Compte AWS identifiant. La CLI GDK crée le bucket s'il n'existe pas.

Important

Les rôles principaux des appareils n'autorisent pas l'accès aux compartiments S3 par défaut. Si c'est la première fois que vous utilisez ce compartiment S3, vous devez ajouter des autorisations au rôle pour permettre aux appareils principaux de récupérer les artefacts des

composants de ce compartiment S3. Pour de plus amples informations, veuillez consulter [Autoriser l'accès aux compartiments S3 pour les artefacts des composants](#).

Pour publier un composant Greengrass (GDK CLI)

1. Ouvrez le dossier du composant dans une invite de commande ou dans un terminal.
2. Si ce n'est pas déjà fait, créez le composant Greengrass. La commande [component build](#) produit une recette et des artefacts dans le `greengrass-build` dossier du dossier des composants. Exécutez la commande suivante.

```
gdk component build
```

3. Publiez le composant dans le AWS Cloud. La commande de [publication du composant](#) télécharge les artefacts du composant sur Amazon S3 et met à jour la recette du composant avec l'URI de chaque artefact. Ensuite, il crée le composant dans le AWS IoT Greengrass service.

 Note

AWS IoT Greengrass calcule le condensé de chaque artefact lorsque vous créez le composant. Cela signifie que vous ne pouvez pas modifier les fichiers d'artefacts de votre compartiment S3 après avoir créé un composant. Dans ce cas, les déploiements incluant ce composant échoueront, car le résumé du fichier ne correspond pas. Si vous modifiez un fichier d'artefact, vous devez créer une nouvelle version du composant.

Si vous spécifiez `NEXT_PATCH` la version du composant dans le fichier de configuration de la CLI GDK, la CLI GDK utilise la version de correctif suivante qui n'existe pas encore dans le AWS IoT Greengrass service.

Exécutez la commande suivante.

```
gdk component publish
```

La sortie indique la version du composant créée par la CLI GDK.

Après avoir publié le composant, vous pouvez le déployer sur les appareils principaux. Pour de plus amples informations, veuillez consulter [Déployer AWS IoT Greengrass des composants sur des appareils](#).

Publier un composant (commandes shell)

Utilisez la procédure suivante pour publier un composant à l'aide des commandes shell et du AWS Command Line Interface (AWS CLI). Lorsque vous publiez un composant, vous devez effectuer les opérations suivantes :

1. Publiez les artefacts des composants dans un compartiment S3.
2. Ajoutez l'URI Amazon S3 de chaque artefact à la recette du composant.
3. Créez une version de composant à AWS IoT Greengrass partir de la recette du composant.

Note

Chaque version de composant que vous chargez doit être unique. Assurez-vous de télécharger la bonne version du composant, car vous ne pourrez pas la modifier après l'avoir chargée.

Vous pouvez suivre ces étapes pour publier un composant depuis votre ordinateur de développement ou votre appareil principal Greengrass.

Pour publier un composant (commandes shell)

1. Si le composant utilise une version qui existe dans le AWS IoT Greengrass service, vous devez modifier la version du composant. Ouvrez la recette dans un éditeur de texte, incrémentez la version et enregistrez le fichier. Choisissez une nouvelle version qui reflète les modifications que vous avez apportées au composant.

Note

AWS IoT Greengrass utilise des versions sémantiques pour les composants. Les versions sémantiques suivent une majeure. mineur. système de numéro de patch. Par

exemple, la version 1.0.0 représente la première version majeure d'un composant. Pour plus d'informations, consultez la [spécification de version sémantique](#).

2. Si votre composant contient des artefacts, procédez comme suit :

a. Publiez les artefacts du composant dans un compartiment S3 de votre Compte AWS.

 Tip

Nous vous recommandons d'inclure le nom et la version du composant dans le chemin d'accès à l'artefact dans le compartiment S3. Ce schéma de dénomination peut vous aider à conserver les artefacts utilisés par les versions précédentes du composant, afin que vous puissiez continuer à prendre en charge les versions précédentes du composant.

Exécutez la commande suivante pour publier un fichier d'artefact dans un compartiment S3. Remplacez `amzn-s3-demo-bucket` par le nom du bucket, puis par `artifacts/com.example.HelloWorld/1.0.0/artifact.py` le chemin d'accès au fichier d'artefact.

```
aws s3 cp artifacts/com.example.HelloWorld/1.0.0/artifact.py s3://amzn-s3-demo-bucket/artifacts/com.example.HelloWorld/1.0.0/artifact.py
```

 Important

Les rôles principaux des appareils n'autorisent pas l'accès aux compartiments S3 par défaut. Si c'est la première fois que vous utilisez ce compartiment S3, vous devez ajouter des autorisations au rôle pour permettre aux appareils principaux de récupérer les artefacts des composants de ce compartiment S3. Pour de plus amples informations, veuillez consulter [Autoriser l'accès aux compartiments S3 pour les artefacts des composants](#).

b. Ajoutez une liste nommée `Artifacts` à la recette du composant si elle n'est pas présente. La `Artifacts` liste apparaît dans chaque manifeste, qui définit les exigences du composant sur chaque plate-forme qu'il prend en charge (ou les exigences par défaut du composant pour toutes les plateformes).

- c. Ajoutez chaque artefact à la liste des artefacts ou mettez à jour l'URI des artefacts existants. L'URI Amazon S3 est composé du nom du compartiment et du chemin d'accès à l'objet artefact contenu dans le compartiment. L'Amazon S3 de vos artefacts URIs doit ressembler à l'exemple suivant.

```
s3://amzn-s3-demo-bucket/artifacts/com.example.HelloWorld/1.0.0/artifact.py
```

Une fois ces étapes terminées, votre recette devrait avoir une Artifacts liste semblable à la suivante.

JSON

```
{
  ...
  "Manifests": [
    {
      "Lifecycle": {
        ...
      },
      "Artifacts": [
        {
          "URI": "s3://amzn-s3-demo-bucket/artifacts/  
MyGreengrassComponent/1.0.0/artifact.py",
          "Unarchive": "NONE"
        }
      ]
    }
  ]
}
```

Note

Vous pouvez ajouter l'"Unarchive": "ZIP" option pour un artefact ZIP afin de configurer le logiciel AWS IoT Greengrass principal afin de décompresser l'artefact lors du déploiement du composant.

YAML

```
...
Manifests:
  - Lifecycle:
      ...
      Artifacts:
        - URI: s3://amzn-s3-demo-bucket/artifacts/MyGreengrassComponent/1.0.0/
          artifact.py
          Unarchive: NONE
```

 Note

Vous pouvez utiliser `Unarchive: ZIP` cette option pour configurer le logiciel AWS IoT Greengrass Core afin de décompresser un artefact ZIP lors du déploiement du composant. Pour plus d'informations sur l'utilisation des artefacts ZIP dans un composant, consultez la variable de recette [artifacts:DecompressedPath](#).

Pour plus d'informations sur les recettes, consultez [AWS IoT Greengrass référence de recette de composant](#).

3. Utilisez la AWS IoT Greengrass console pour créer un composant à partir du fichier de recette.

Exécutez la commande suivante pour créer le composant à partir d'un fichier de recette. Cette commande crée le composant et le publie en tant que AWS IoT Greengrass composant privé dans votre Compte AWS. Remplacez *path/to/recipeFile* par le chemin du fichier de recette.

```
aws greengrassv2 create-component-version --inline-recipe fileb://path/to/
recipeFile
```

Copiez le contenu `arn` de la réponse pour vérifier l'état du composant à l'étape suivante.

 Note

AWS IoT Greengrass calcule le condensé de chaque artefact lorsque vous créez le composant. Cela signifie que vous ne pouvez pas modifier les fichiers d'artefacts de

votre compartiment S3 après avoir créé un composant. Dans ce cas, les déploiements incluant ce composant échoueront, car le résumé du fichier ne correspond pas. Si vous modifiez un fichier d'artefact, vous devez créer une nouvelle version du composant.

- Chaque composant du AWS IoT Greengrass service possède un état. Exécutez la commande suivante pour confirmer l'état de la version du composant que vous publiez dans cette procédure. Remplacez *com.example.HelloWorld* et *1.0.0* par la version du composant à interroger. Remplacez le arn par l'ARN de l'étape précédente.

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

L'opération renvoie une réponse contenant les métadonnées du composant. Les métadonnées contiennent un status objet qui contient l'état du composant et les éventuelles erreurs, le cas échéant.

Lorsque l'état du composant est DEPLOYABLE défini, vous pouvez le déployer sur des appareils. Pour de plus amples informations, veuillez consulter [Déployer AWS IoT Greengrass des composants sur des appareils](#).

Interagissez avec les AWS services

Les appareils principaux de Greengrass utilisent des certificats X.509 pour se connecter à l' AWS IoT Core aide des protocoles d'authentification mutuelle TLS. Ces certificats permettent aux appareils d'interagir AWS IoT sans AWS informations d'identification, qui comprennent généralement un identifiant de clé d'accès et une clé d'accès secrète. D'autres AWS services nécessitent des AWS informations d'identification au lieu de certificats X.509 pour appeler les opérations d'API aux points de terminaison du service. AWS IoT Core dispose d'un fournisseur d'informations d'identification qui permet aux appareils d'utiliser leur certificat X.509 pour AWS authentifier les demandes. Le fournisseur AWS IoT d'informations d'identification authentifie les appareils à l'aide d'un certificat X.509 et émet des AWS informations d'identification sous la forme d'un jeton de sécurité temporaire à privilèges limités. Les appareils peuvent utiliser ce jeton pour signer et authentifier toute AWS demande. Cela élimine le besoin de stocker les AWS informations d'identification sur les appareils principaux de Greengrass. Pour plus d'informations, consultez la section [Autorisation des appels directs vers AWS des services](#) dans le Guide du AWS IoT Core développeur.

Pour récupérer les informations d'identification auprès de Greengrass AWS IoT, les appareils principaux utilisent AWS IoT un alias de rôle qui pointe vers un rôle IAM. Ce rôle IAM est appelé rôle d'échange de jetons. Vous créez l'alias de rôle et le rôle d'échange de jetons lorsque vous installez le logiciel AWS IoT Greengrass Core. Pour spécifier l'alias de rôle utilisé par un périphérique principal, configurez le `iotRoleAlias` paramètre du [Noyau de Greengrass](#).

Le fournisseur AWS IoT d'informations d'identification assume le rôle d'échange de jetons en votre nom pour fournir des AWS informations d'identification aux appareils principaux. Vous pouvez associer des politiques IAM appropriées à ce rôle pour permettre à vos appareils principaux d'accéder à vos AWS ressources, telles que les artefacts des composants dans les compartiments S3. Pour plus d'informations sur la configuration du rôle d'échange de jetons, consultez [Autoriser les appareils principaux à interagir avec les AWS services](#).

Les appareils Greengrass Core stockent les AWS informations d'identification en mémoire, qui expirent au bout d'une heure par défaut. Si le logiciel AWS IoT Greengrass Core redémarre, il doit à nouveau récupérer les informations d'identification. Vous pouvez utiliser cette [UpdateRoleAlias](#) opération pour configurer la durée de validité des informations d'identification.

AWS IoT Greengrass fournit un composant public, le composant du service d'échange de jetons, que vous pouvez définir comme une dépendance dans votre composant personnalisé pour interagir avec les AWS services. Le service d'échange de jetons fournit à votre composant une variable d'environnement qui définit l'URI d'un serveur local fournissant des AWS informations d'identification. `AWS_CONTAINER_CREDENTIALS_FULL_URI` Lorsque vous créez un client AWS SDK, le client vérifie la présence de cette variable d'environnement et se connecte au serveur local pour récupérer les AWS informations d'identification et les utilise pour signer les demandes d'API. Cela vous permet d'utiliser AWS SDKs d'autres outils pour appeler AWS des services dans vos composants. Pour de plus amples informations, veuillez consulter [Service d'échange de jetons](#).

Important

Support pour acquérir des AWS informations d'identification de cette manière a été ajouté AWS SDKs le 13 juillet 2016. Votre composant doit utiliser une version du AWS SDK créée à cette date ou après cette date. Pour plus d'informations, consultez la section [Utilisation d'un AWS SDK compatible](#) dans le manuel Amazon Elastic Container Service Developer Guide.

Pour obtenir des AWS informations d'identification dans votre composant personnalisé, `aws.greengrass.TokenExchangeService` définissez-le en tant que dépendance dans la recette

du composant. L'exemple de recette suivant définit un composant qui installe [boto3](#) et exécute un script Python qui utilise les informations d'AWS identification du service d'échange de jetons pour répertorier les buckets Amazon S3.

Note

Pour exécuter cet exemple de composant, votre appareil doit disposer de l'`s3:ListAllMyBuckets` autorisation. Pour de plus amples informations, veuillez consulter [Autoriser les appareils principaux à interagir avec les AWS services](#).

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.ListS3Buckets",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that uses the token exchange service to list S3 buckets.",
  "ComponentPublisher": "Amazon",
  "ComponentDependencies": {
    "aws.greengrass.TokenExchangeService": {
      "VersionRequirement": "^2.0.0",
      "DependencyType": "HARD"
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "install": "pip3 install --user boto3",
        "Run": "python3 -u {artifacts:path}/list_s3_buckets.py"
      }
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "install": "pip3 install --user boto3",
```



```

    "Run": "py -3 -u {artifacts:path}/list_s3_buckets.py"
  }
}
]
}

```

YAML

```

---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.ListS3Buckets
ComponentVersion: '1.0.0'
ComponentDescription: A component that uses the token exchange service to list S3
  buckets.
ComponentPublisher: Amazon
ComponentDependencies:
  aws.greengrass.TokenExchangeService:
    VersionRequirement: '^2.0.0'
    DependencyType: HARD
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      install:
        pip3 install --user boto3
      Run: |-
        python3 -u {artifacts:path}/list_s3_buckets.py
  - Platform:
      os: windows
    Lifecycle:
      install:
        pip3 install --user boto3
      Run: |-
        py -3 -u {artifacts:path}/list_s3_buckets.py

```

Cet exemple de composant exécute le script Python suivant, `list_s3_buckets.py` qui répertorie les compartiments Amazon S3.

```

import boto3
import os

try:

```

```
print("Creating boto3 S3 client...")
s3 = boto3.client('s3')
print("Successfully created boto3 S3 client")
except Exception as e:
    print("Failed to create boto3 s3 client. Error: " + str(e))
    exit(1)

try:
    print("Listing S3 buckets...")
    response = s3.list_buckets()
    for bucket in response['Buckets']:
        print(f'\t{bucket["Name"]}')
    print("Successfully listed S3 buckets")
except Exception as e:
    print("Failed to list S3 buckets. Error: " + str(e))
    exit(1)
```

Exécuter un conteneur Docker

Vous pouvez configurer AWS IoT Greengrass des composants pour exécuter un conteneur [Docker](#) à partir d'images stockées aux emplacements suivants :

- Référentiels d'images publics et privés dans Amazon Elastic Container Registry (Amazon ECR)
- Référentiel Docker Hub public
- Registre public de confiance Docker
- compartiment S3

Dans votre composant personnalisé, incluez l'URI de l'image Docker en tant qu'artefact pour récupérer l'image et l'exécuter sur le périphérique principal. Pour les images Amazon ECR et Docker Hub, vous pouvez utiliser le composant du [gestionnaire d'applications Docker](#) pour télécharger les images et gérer les informations d'identification des référentiels Amazon ECR privés.

Rubriques

- [Exigences](#)
- [Exécuter un conteneur Docker à partir d'une image publique dans Amazon ECR ou Docker Hub](#)
- [Exécuter un conteneur Docker à partir d'une image privée dans Amazon ECR](#)
- [Exécuter un conteneur Docker à partir d'une image dans Amazon S3](#)
- [Utiliser la communication interprocessus dans les composants du conteneur Docker](#)

- [Utiliser les AWS informations d'identification dans les composants du conteneur Docker \(Linux\)](#)
- [Utiliser le gestionnaire de flux dans les composants du conteneur Docker \(Linux\)](#)

Exigences

Pour exécuter un conteneur Docker dans un composant, vous avez besoin des éléments suivants :

- Un appareil Greengrass Core. Si vous n'en avez pas, veuillez consulter [Tutoriel : Débuter avec AWS IoT Greengrass V2](#).
- [Docker Engine](#) 1.9.1 ou version ultérieure installé sur le périphérique principal de Greengrass. La version 20.10 est la dernière version vérifiée pour fonctionner avec le logiciel AWS IoT Greengrass Core. Vous devez installer Docker directement sur le périphérique principal avant de déployer des composants qui exécutent des conteneurs Docker.

Tip

Vous pouvez également configurer le périphérique principal pour installer Docker Engine lors de l'installation du composant. Par exemple, le script d'installation suivant installe Docker Engine avant de charger l'image Docker. Ce script d'installation fonctionne sur les distributions Linux basées sur Debian, telles qu'Ubuntu. Si vous configurez le composant pour installer Docker Engine à l'aide de cette commande, vous devrez peut-être le `RequiresPrivilege` définir `true` dans le script de cycle de vie pour exécuter l'installation et `docker` les commandes. Pour de plus amples informations, veuillez consulter [AWS IoT Greengrass référence de recette de composant](#).

```
apt-get install docker-ce docker-ce-cli containerd.io && docker load -i  
{artifacts:path}/hello-world.tar
```

- L'utilisateur du système qui exécute un composant de conteneur Docker doit disposer des autorisations root ou administrateur, ou vous devez configurer Docker pour l'exécuter en tant qu'utilisateur non root ou non administrateur.
- Sur les appareils Linux, vous pouvez ajouter un utilisateur au `docker` groupe sans lequel vous pouvez appeler `docker` des commandes `sudo`.
- Sur les appareils Windows, vous pouvez ajouter un utilisateur au `docker-users` groupe pour appeler des `docker` commandes sans privilèges d'administrateur.

Linux or Unix

Pour ajouter `ggc_user` au `docker` groupe l'utilisateur non root que vous utilisez pour exécuter les composants du conteneur Docker, exécutez la commande suivante.

```
sudo usermod -aG docker ggc_user
```

Pour plus d'informations, consultez [Gérer Docker en tant qu'utilisateur non root](#).

Windows Command Prompt (CMD)

Pour ajouter `ggc_user` au `docker-users` groupe l'utilisateur que vous utilisez pour exécuter les composants du conteneur Docker, exécutez la commande suivante en tant qu'administrateur.

```
net localgroup docker-users ggc_user /add
```

Windows PowerShell

Pour ajouter `ggc_user` au `docker-users` groupe l'utilisateur que vous utilisez pour exécuter les composants du conteneur Docker, exécutez la commande suivante en tant qu'administrateur.

```
Add-LocalGroupMember -Group docker-users -Member ggc_user
```

- Fichiers accessibles par le composant de conteneur Docker [montés sous forme de volume](#) dans le conteneur Docker.
- Si vous [configurez le logiciel AWS IoT Greengrass Core pour utiliser un proxy réseau](#), vous devez [configurer Docker pour qu'il utilise le même serveur proxy](#).

Outre ces exigences, vous devez également satisfaire aux exigences suivantes si elles s'appliquent à votre environnement :

- Pour utiliser [Docker Compose](#) pour créer et démarrer vos conteneurs Docker, installez Docker Compose sur votre appareil principal Greengrass et téléchargez votre fichier Docker Compose dans un compartiment S3. Vous devez stocker votre fichier Compose dans un compartiment S3 au même Compte AWS endroit Région AWS que le composant. Pour un exemple d'utilisation de

la `docker-compose up` commande dans un composant personnalisé, consultez [Exécuter un conteneur Docker à partir d'une image publique dans Amazon ECR ou Docker Hub](#).

- Si vous utilisez un AWS IoT Greengrass proxy réseau, configurez le daemon Docker pour qu'il utilise un [serveur proxy](#).
- Si vos images Docker sont stockées dans Amazon ECR ou Docker Hub, incluez le [composant du gestionnaire de composants Docker](#) en tant que dépendance dans votre composant conteneur Docker. Vous devez démarrer le démon Docker sur l'appareil principal avant de déployer votre composant.

Incluez également l'image URIs en tant qu'artefacts de composant. L'image URIs doit être au format `docker:registry/image[:tag|@digest]` indiqué dans les exemples suivants :

- Image Amazon ECR privée : `docker:account-id.dkr.ecr.region.amazonaws.com/repository/image[:tag|@digest]`
- Image Amazon ECR publique : `docker:public.ecr.aws/repository/image[:tag|@digest]`
- Image publique du Docker Hub : `docker:name[:tag|@digest]`

Pour plus d'informations sur l'exécution de conteneurs Docker à partir d'images stockées dans des référentiels publics, consultez. [Exécuter un conteneur Docker à partir d'une image publique dans Amazon ECR ou Docker Hub](#)

- Si vos images Docker sont stockées dans un référentiel privé Amazon ECR, vous devez inclure le composant du service d'échange de jetons en tant que dépendance dans le composant conteneur Docker. En outre, le [rôle d'appareil Greengrass](#) doit autoriser les `ecr:GetDownloadUrlForLayer` actions `ecr:GetAuthorizationToken``ecr:BatchGetImage`, et, comme indiqué dans l'exemple de politique IAM suivant.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "ecr:GetAuthorizationToken",
        "ecr:BatchGetImage",
        "ecr:GetDownloadUrlForLayer"
      ],
    }
  ],
}
```

```
    "Resource": [
      "*"
    ],
    "Effect": "Allow"
  }
]
```

Pour plus d'informations sur l'exécution de conteneurs Docker à partir d'images stockées dans un référentiel privé Amazon ECR, consultez [Exécuter un conteneur Docker à partir d'une image privée dans Amazon ECR](#)

- Pour utiliser des images Docker stockées dans un référentiel privé Amazon ECR, celui-ci doit se trouver dans le même emplacement Région AWS que le périphérique principal.
- Si vos images Docker ou vos fichiers Compose sont stockés dans un compartiment S3, le rôle d'[appareil Greengrass](#) doit autoriser `s3:GetObject` les appareils principaux à télécharger les images en tant qu'artefacts de composants, comme illustré dans l'exemple de politique IAM suivant.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "s3:GetObject"
      ],
      "Resource": [
        "*"
      ],
      "Effect": "Allow"
    }
  ]
}
```

Pour plus d'informations sur l'exécution de conteneurs Docker à partir d'images stockées dans Amazon S3, consultez [Exécuter un conteneur Docker à partir d'une image dans Amazon S3](#).

- Pour utiliser la communication interprocessus (IPC), les AWS informations d'identification ou le gestionnaire de flux dans votre composant de conteneur Docker, vous devez spécifier des options

supplémentaires lorsque vous exécutez le conteneur Docker. Pour plus d'informations, consultez les ressources suivantes :

- [Utiliser la communication interprocessus dans les composants du conteneur Docker](#)
- [Utiliser les AWS informations d'identification dans les composants du conteneur Docker \(Linux\)](#)
- [Utiliser le gestionnaire de flux dans les composants du conteneur Docker \(Linux\)](#)

Exécuter un conteneur Docker à partir d'une image publique dans Amazon ECR ou Docker Hub

Cette section décrit comment créer un composant personnalisé qui utilise Docker Compose pour exécuter un conteneur Docker à partir d'images Docker stockées sur Amazon ECR et Docker Hub.

Pour exécuter un conteneur Docker à l'aide de Docker Compose

1. Créez et chargez un fichier Docker Compose dans un compartiment Amazon S3. Assurez-vous que le [rôle d'appareil Greengrass](#) `s3:GetObject` autorise l'appareil à accéder au fichier Compose. L'exemple de fichier Compose présenté dans l'exemple suivant inclut l'image Amazon CloudWatch Agent d'Amazon ECR et l'image MySQL de Docker Hub.

```
version: "3"
services:
  cloudwatchagent:
    image: "public.ecr.aws/cloudwatch-agent/cloudwatch-agent:latest"
  mysql:
    image: "mysql:8.0"
```

2. [Créez un composant personnalisé](#) sur votre appareil AWS IoT Greengrass principal. L'exemple de recette présenté dans l'exemple suivant possède les propriétés suivantes :
- Le composant du gestionnaire d'applications Docker en tant que dépendance. Ce composant permet AWS IoT Greengrass de télécharger des images depuis les référentiels publics Amazon ECR et Docker Hub.
 - Artefact de composant qui spécifie une image Docker dans un référentiel Amazon ECR public.
 - Artefact de composant qui spécifie une image Docker dans un référentiel Docker Hub public.
 - Artefact de composant qui spécifie le fichier Docker Compose qui inclut les conteneurs pour les images Docker que vous souhaitez exécuter.

- Script d'exécution du cycle de vie qui utilise [docker-compose up](#) pour créer et démarrer un conteneur à partir des images spécifiées.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.MyDockerComposeComponent",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that uses Docker Compose to run images
from public Amazon ECR and Docker Hub.",
  "ComponentPublisher": "Amazon",
  "ComponentDependencies": {
    "aws.greengrass.DockerApplicationManager": {
      "VersionRequirement": "~2.0.0"
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "all"
      },
      "Lifecycle": {
        "Run": "docker-compose -f {artifacts:path}/docker-compose.yaml up"
      },
      "Artifacts": [
        {
          "URI": "docker:public.ecr.aws/cloudwatch-agent/cloudwatch-
agent:latest"
        },
        {
          "URI": "docker:mysql:8.0"
        },
        {
          "URI": "s3://amzn-s3-demo-bucket/folder/docker-compose.yaml"
        }
      ]
    }
  ]
}
```


YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.MyDockerComposeComponent
ComponentVersion: '1.0.0'
ComponentDescription: 'A component that uses Docker Compose to run images from
  public Amazon ECR and Docker Hub.'
ComponentPublisher: Amazon
ComponentDependencies:
  aws.greengrass.DockerApplicationManager:
    VersionRequirement: ~2.0.0
Manifests:
  - Platform:
      os: all
    Lifecycle:
      Run: docker-compose -f {artifacts:path}/docker-compose.yaml up
    Artifacts:
      - URI: "docker:public.ecr.aws/cloudwatch-agent/cloudwatch-agent:latest"
      - URI: "docker:mysql:8.0"
      - URI: "s3://amzn-s3-demo-bucket/folder/docker-compose.yaml"
```

Note

Pour utiliser la communication interprocessus (IPC), les AWS informations d'identification ou le gestionnaire de flux dans votre composant de conteneur Docker, vous devez spécifier des options supplémentaires lorsque vous exécutez le conteneur Docker. Pour plus d'informations, consultez les ressources suivantes :

- [Utiliser la communication interprocessus dans les composants du conteneur Docker](#)
- [Utiliser les AWS informations d'identification dans les composants du conteneur Docker \(Linux\)](#)
- [Utiliser le gestionnaire de flux dans les composants du conteneur Docker \(Linux\)](#)

3. [Testez le composant](#) pour vérifier qu'il fonctionne comme prévu.

⚠ Important

Vous devez installer et démarrer le démon Docker avant de déployer le composant.

Après avoir déployé le composant localement, vous pouvez exécuter la commande [docker container ls](#) pour vérifier que votre conteneur fonctionne.

```
docker container ls
```

4. Lorsque le composant est prêt, téléchargez-le AWS IoT Greengrass vers pour le déployer sur d'autres appareils principaux. Pour de plus amples informations, veuillez consulter [Publiez des composants à déployer sur vos appareils principaux](#).

Exécuter un conteneur Docker à partir d'une image privée dans Amazon ECR

Cette section décrit comment créer un composant personnalisé qui exécute un conteneur Docker à partir d'une image Docker stockée dans un référentiel privé sur Amazon ECR.

Pour exécuter un conteneur Docker

1. [Créez un composant personnalisé](#) sur votre appareil AWS IoT Greengrass principal. Utilisez l'exemple de recette suivant, qui possède les propriétés suivantes :
 - Le composant du gestionnaire d'applications Docker en tant que dépendance. Ce composant permet AWS IoT Greengrass de gérer les informations d'identification pour télécharger des images depuis des référentiels privés.
 - Le composant du service d'échange de jetons en tant que dépendance. Ce composant permet de AWS IoT Greengrass récupérer les AWS informations d'identification pour interagir avec Amazon ECR.
 - Artefact de composant qui spécifie une image Docker dans un référentiel Amazon ECR privé.
 - Script d'exécution du cycle de vie qui utilise [docker run](#) pour créer et démarrer un conteneur à partir de l'image.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.MyPrivateDockerComponent",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that runs a Docker container from a
private Amazon ECR image.",
  "ComponentPublisher": "Amazon",
  "ComponentDependencies": {
    "aws.greengrass.DockerApplicationManager": {
      "VersionRequirement": "~2.0.0"
    },
    "aws.greengrass.TokenExchangeService": {
      "VersionRequirement": "~2.0.0"
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "all"
      },
      "Lifecycle": {
        "Run": "docker run account-
id.dkr.ecr.region.amazonaws.com/repository[:tag|@digest]"
      },
      "Artifacts": [
        {
          "URI": "docker:account-
id.dkr.ecr.region.amazonaws.com/repository[:tag|@digest]"
        }
      ]
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.MyPrivateDockerComponent
```

```
ComponentVersion: '1.0.0'
ComponentDescription: 'A component that runs a Docker container from a private
  Amazon ECR image.'
ComponentPublisher: Amazon
ComponentDependencies:
  aws.greengrass.DockerApplicationManager:
    VersionRequirement: ~2.0.0
  aws.greengrass.TokenExchangeService:
    VersionRequirement: ~2.0.0
Manifests:
  - Platform:
      os: all
    Lifecycle:
      Run: docker run account-id.dkr.ecr.region.amazonaws.com/repository[:tag/
@digest]
    Artifacts:
      - URI: "docker:account-id.dkr.ecr.region.amazonaws.com/repository[:tag/
@digest]"
```

Note

Pour utiliser la communication interprocessus (IPC), les AWS informations d'identification ou le gestionnaire de flux dans votre composant de conteneur Docker, vous devez spécifier des options supplémentaires lorsque vous exécutez le conteneur Docker. Pour plus d'informations, consultez les ressources suivantes :

- [Utiliser la communication interprocessus dans les composants du conteneur Docker](#)
- [Utiliser les AWS informations d'identification dans les composants du conteneur Docker \(Linux\)](#)
- [Utiliser le gestionnaire de flux dans les composants du conteneur Docker \(Linux\)](#)

2. [Testez le composant](#) pour vérifier qu'il fonctionne comme prévu.

Important

Vous devez installer et démarrer le démon Docker avant de déployer le composant.

Après avoir déployé le composant localement, vous pouvez exécuter la commande [docker container ls](#) pour vérifier que votre conteneur fonctionne.

```
docker container ls
```

3. Téléchargez le composant AWS IoT Greengrass vers pour le déployer sur d'autres appareils principaux. Pour de plus amples informations, veuillez consulter [Publiez des composants à déployer sur vos appareils principaux](#).

Exécuter un conteneur Docker à partir d'une image dans Amazon S3

Cette section décrit comment exécuter un conteneur Docker dans un composant à partir d'une image Docker stockée dans Amazon S3.

Pour exécuter un conteneur Docker dans un composant à partir d'une image dans Amazon S3

1. Exécutez la commande [docker save](#) pour créer une sauvegarde d'un conteneur Docker. Vous fournissez cette sauvegarde en tant qu'artefact de composant sur AWS IoT Greengrass lequel exécuter le conteneur. Remplacez *hello-world* par le nom de l'image, puis *hello-world.tar* par le nom du fichier d'archive à créer.

```
docker save hello-world > artifacts/com.example.MyDockerComponent/1.0.0/hello-world.tar
```

2. [Créez un composant personnalisé](#) sur votre appareil AWS IoT Greengrass principal. Utilisez l'exemple de recette suivant, qui possède les propriétés suivantes :
 - Script d'installation du cycle de vie qui utilise le chargement de [docker pour charger](#) une image Docker à partir d'une archive.
 - Script d'exécution du cycle de vie qui utilise [docker run](#) pour créer et démarrer un conteneur à partir de l'image. L'option `--rm` nettoie le conteneur à sa sortie.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.MyS3DockerComponent",
```

```
"ComponentVersion": "1.0.0",
"ComponentDescription": "A component that runs a Docker container from an
image in an S3 bucket.",
"ComponentPublisher": "Amazon",
"Manifests": [
  {
    "Platform": {
      "os": "linux"
    },
    "Lifecycle": {
      "install": {
        "Script": "docker load -i {artifacts:path}/hello-world.tar"
      },
      "Run": {
        "Script": "docker run --rm hello-world"
      }
    }
  }
]
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.MyS3DockerComponent
ComponentVersion: '1.0.0'
ComponentDescription: 'A component that runs a Docker container from an image in
an S3 bucket.'
ComponentPublisher: Amazon
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      install:
        Script: docker load -i {artifacts:path}/hello-world.tar
      Run:
        Script: docker run --rm hello-world
```

Note

Pour utiliser la communication interprocessus (IPC), les AWS informations d'identification ou le gestionnaire de flux dans votre composant de conteneur Docker, vous devez spécifier des options supplémentaires lorsque vous exécutez le conteneur Docker. Pour plus d'informations, consultez les ressources suivantes :

- [Utiliser la communication interprocessus dans les composants du conteneur Docker](#)
- [Utiliser les AWS informations d'identification dans les composants du conteneur Docker \(Linux\)](#)
- [Utiliser le gestionnaire de flux dans les composants du conteneur Docker \(Linux\)](#)

3. [Testez le composant](#) pour vérifier qu'il fonctionne comme prévu.

Après avoir déployé le composant localement, vous pouvez exécuter la commande [docker container ls](#) pour vérifier que votre conteneur fonctionne.

```
docker container ls
```

4. Lorsque le composant est prêt, téléchargez l'archive d'images Docker dans un compartiment S3 et ajoutez son URI à la recette du composant. Vous pouvez ensuite télécharger le composant pour le AWS IoT Greengrass déployer sur d'autres appareils principaux. Pour de plus amples informations, veuillez consulter [Publiez des composants à déployer sur vos appareils principaux](#).

Lorsque vous avez terminé, la recette du composant doit ressembler à l'exemple suivant.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.MyS3DockerComponent",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that runs a Docker container from an image in an S3 bucket.",
  "ComponentPublisher": "Amazon",
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      }
    }
  ]
}
```

```

    },
    "Lifecycle": {
      "install": {
        "Script": "docker load -i {artifacts:path}/hello-world.tar"
      },
      "Run": {
        "Script": "docker run --rm hello-world"
      }
    },
    "Artifacts": [
      {
        "URI": "s3://amzn-s3-demo-bucket/artifacts/  
com.example.MyDockerComponent/1.0.0/hello-world.tar"
      }
    ]
  }
}

```

YAML

```

---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.MyS3DockerComponent
ComponentVersion: '1.0.0'
ComponentDescription: 'A component that runs a Docker container from an image in
  an S3 bucket.'
ComponentPublisher: Amazon
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      install:
        Script: docker load -i {artifacts:path}/hello-world.tar
      Run:
        Script: docker run --rm hello-world
    Artifacts:
      - URI: s3://amzn-s3-demo-bucket/artifacts/  
com.example.MyDockerComponent/1.0.0/hello-world.tar

```


Utiliser la communication interprocessus dans les composants du conteneur Docker

Vous pouvez utiliser la bibliothèque de communication interprocessus (IPC) Greengrass dans le Kit SDK des appareils AWS IoT pour communiquer avec le noyau de Greengrass, les autres composants de Greengrass et AWS IoT Core. Pour de plus amples informations, veuillez consulter [Utilisez le Kit SDK des appareils AWS IoT pour communiquer avec le noyau de Greengrass, les autres composants et AWS IoT Core](#).

Pour utiliser IPC dans un composant de conteneur Docker, vous devez exécuter le conteneur Docker avec les paramètres suivants :

- Montez le socket IPC dans le conteneur. Le noyau Greengrass fournit le chemin du fichier du socket IPC dans la `AWS_GG_NUCLEUS_DOMAIN_SOCKET_FILEPATH_FOR_COMPONENT` variable d'environnement.
- Définissez les variables d'`AWS_GG_NUCLEUS_DOMAIN_SOCKET_FILEPATH_FOR_COMPONENT` environnement `SVCUID` et sur les valeurs que le noyau Greengrass fournit aux composants. Votre composant utilise ces variables d'environnement pour authentifier les connexions au noyau Greengrass.

Exemple Exemple de recette : publier un message MQTT sur AWS IoT Core (Python)

La recette suivante définit un exemple de composant de conteneur Docker qui publie un message MQTT sur AWS IoT Core. Cette recette possède les propriétés suivantes :

- Une politique d'autorisation (`accessControl`) qui permet au composant de publier des messages MQTT AWS IoT Core sur tous les sujets. Pour plus d'informations, consultez [Autoriser les composants à effectuer des opérations IPC](#) et autorisation [AWS IoT Core MQTT IPC](#).
- Artefact de composant qui spécifie une image Docker en tant qu'archive TAR dans Amazon S3.
- Script d'installation du cycle de vie qui charge l'image Docker depuis l'archive TAR.
- Script d'exécution du cycle de vie qui exécute un conteneur Docker à partir de l'image. La commande [Docker run](#) comporte les arguments suivants :
 - L'-v argument monte le socket IPC Greengrass dans le conteneur.
 - Les deux premiers -e arguments définissent les variables d'environnement requises dans le conteneur Docker.
 - Les -e arguments supplémentaires définissent les variables d'environnement utilisées dans cet exemple.

- L'`--rm` argument nettoie le conteneur à sa sortie.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.python.docker.PublishToIoTCore",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "Uses interprocess communication to publish an MQTT
message to IoT Core.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "topic": "test/topic/java",
      "message": "Hello, World!",
      "qos": "1",
      "accessControl": {
        "aws.greengrass.ipc.mqttproxy": {
          "com.example.python.docker.PublishToIoTCore:pubsub:1": {
            "policyDescription": "Allows access to publish to IoT Core on all
topics.",
            "operations": [
              "aws.greengrass#PublishToIoTCore"
            ],
            "resources": [
              "*"
            ]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "all"
      },
      "Lifecycle": {
        "install": "docker load -i {artifacts:path}/publish-to-iot-core.tar",
        "Run": "docker run -v $AWS_GG_NUCLEUS_DOMAIN_SOCKET_FILEPATH_FOR_COMPONENT:
$AWS_GG_NUCLEUS_DOMAIN_SOCKET_FILEPATH_FOR_COMPONENT -e SVCUID -e
AWS_GG_NUCLEUS_DOMAIN_SOCKET_FILEPATH_FOR_COMPONENT -e MQTT_TOPIC="
```

```

\"{configuration:/topic}\" -e MQTT_MESSAGE=\"{configuration:/message}\" -e MQTT_QOS=
\"{configuration:/qos}\" --rm publish-to-iot-core"
    },
    "Artifacts": [
        {
            "URI": "s3://amzn-s3-demo-bucket/artifacts/
com.example.python.docker.PublishToIoTCore/1.0.0/publish-to-iot-core.tar"
        }
    ]
}
]
}

```

YAML

```

RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.python.docker.PublishToIoTCore
ComponentVersion: 1.0.0
ComponentDescription: Uses interprocess communication to publish an MQTT message to
IoT Core.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    topic: 'test/topic/java'
    message: 'Hello, World!'
    qos: '1'
  accessControl:
    aws.greengrass.ipc.mqttproxy:
      'com.example.python.docker.PublishToIoTCore:pubsub:1':
        policyDescription: Allows access to publish to IoT Core on all topics.
        operations:
          - 'aws.greengrass#PublishToIoTCore'
        resources:
          - '*'
Manifests:
  - Platform:
      os: all
    Lifecycle:
      install: 'docker load -i {artifacts:path}/publish-to-iot-core.tar'
      Run: |
        docker run \
          -v $AWS_GG_NUCLEUS_DOMAIN_SOCKET_FILEPATH_FOR_COMPONENT:
$AWS_GG_NUCLEUS_DOMAIN_SOCKET_FILEPATH_FOR_COMPONENT \

```

```
-e SVCUID \  
-e AWS_GG_NUCLEUS_DOMAIN_SOCKET_FILEPATH_FOR_COMPONENT \  
-e MQTT_TOPIC="{configuration:/topic}" \  
-e MQTT_MESSAGE="{configuration:/message}" \  
-e MQTT_QOS="{configuration:/qos}" \  
--rm publish-to-iot-core
```

Artifacts:

```
- URI: s3://amzn-s3-demo-bucket/artifacts/  
com.example.python.docker.PublishToIoTCore/1.0.0/publish-to-iot-core.tar
```

Utiliser les AWS informations d'identification dans les composants du conteneur Docker (Linux)

Vous pouvez utiliser le [composant de service d'échange de jetons](#) pour interagir avec les AWS services des composants Greengrass. Ce composant fournit des AWS informations d'identification provenant du [rôle d'échange de jetons](#) du périphérique principal à l'aide d'un serveur de conteneurs local. Pour de plus amples informations, veuillez consulter [Interagissez avec les AWS services](#).

Note

L'exemple présenté dans cette section ne fonctionne que sur les appareils principaux de Linux.

Pour utiliser les AWS informations d'identification du service d'échange de jetons dans un composant de conteneur Docker, vous devez exécuter le conteneur Docker avec les paramètres suivants :

- Fournissez l'accès au réseau hôte à l'aide de l'`--network=host` argument. Cette option permet au conteneur Docker de se connecter au service d'échange de jetons local pour récupérer les AWS informations d'identification. Cet argument ne fonctionne que sur Docker pour Linux.

Warning

Cette option permet au conteneur d'accéder à toutes les interfaces réseau locales de l'hôte. Elle est donc moins sécurisée que si vous exécutez des conteneurs Docker sans cet accès au réseau hôte. Tenez-en compte lorsque vous développez et exécutez des composants de conteneur Docker qui utilisent cette option. Pour plus d'informations, consultez [Network : host](#) dans la documentation Docker.

- Définissez les variables d'AWS_CONTAINER_AUTHORIZATION_TOKENenvironnement AWS_CONTAINER_CREDENTIALS_FULL_URI et sur les valeurs que le noyau Greengrass fournit aux composants. AWS SDKs utilisent ces variables d'environnement pour récupérer les AWS informations d'identification.

Exemple Exemple de recette : répertorier les compartiments S3 dans un composant de conteneur Docker (Python)

La recette suivante définit un exemple de composant de conteneur Docker qui répertorie les compartiments S3 de votre. Compte AWS Cette recette possède les propriétés suivantes :

- Le composant du service d'échange de jetons en tant que dépendance. Cette dépendance permet au composant de récupérer des AWS informations d'identification pour interagir avec d'autres AWS services.
- Artefact de composant qui spécifie une image Docker sous forme d'archive tar dans Amazon S3.
- Script d'installation du cycle de vie qui charge l'image Docker depuis l'archive TAR.
- Script d'exécution du cycle de vie qui exécute un conteneur Docker à partir de l'image. La commande [Docker run](#) comporte les arguments suivants :
 - L'- -network=hostargument fournit au conteneur l'accès au réseau hôte, afin que le conteneur puisse se connecter au service d'échange de jetons.
 - L'-eargument définit les variables d'environnement requises dans le conteneur Docker.
 - L'- -rmargument nettoie le conteneur à sa sortie.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.python.docker.ListS3Buckets",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "Uses the token exchange service to lists your S3 buckets.",
  "ComponentPublisher": "Amazon",
  "ComponentDependencies": {
    "aws.greengrass.TokenExchangeService": {
      "VersionRequirement": "^2.0.0",
      "DependencyType": "HARD"
    }
  },
}
```

```

"Manifests": [
  {
    "Platform": {
      "os": "linux"
    },
    "Lifecycle": {
      "install": "docker load -i {artifacts:path}/list-s3-buckets.tar",
      "Run": "docker run --network=host -e AWS_CONTAINER_AUTHORIZATION_TOKEN -e
AWS_CONTAINER_CREDENTIALS_FULL_URI --rm list-s3-buckets"
    },
    "Artifacts": [
      {
        "URI": "s3://amzn-s3-demo-bucket/artifacts/
com.example.python.docker.ListS3Buckets/1.0.0/list-s3-buckets.tar"
      }
    ]
  }
]
}

```

YAML

```

RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.python.docker.ListS3Buckets
ComponentVersion: 1.0.0
ComponentDescription: Uses the token exchange service to lists your S3 buckets.
ComponentPublisher: Amazon
ComponentDependencies:
  aws.greengrass.TokenExchangeService:
    VersionRequirement: ^2.0.0
    DependencyType: HARD
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      install: 'docker load -i {artifacts:path}/list-s3-buckets.tar'
      Run: |
        docker run \
          --network=host \
          -e AWS_CONTAINER_AUTHORIZATION_TOKEN \
          -e AWS_CONTAINER_CREDENTIALS_FULL_URI \
          --rm list-s3-buckets
    Artifacts:

```

```
- URI: s3://amzn-s3-demo-bucket/artifacts/  
com.example.python.docker.ListS3Buckets/1.0.0/list-s3-buckets.tar
```

Utiliser le gestionnaire de flux dans les composants du conteneur Docker (Linux)

Vous pouvez utiliser le [composant gestionnaire de flux](#) pour gérer les flux de données dans les composants Greengrass. Ce composant vous permet de traiter des flux de données et de transférer de gros volumes de données IoT vers le AWS Cloud. AWS IoT Greengrass fournit un SDK de gestion de flux que vous utilisez pour interagir avec le composant de gestionnaire de flux. Pour de plus amples informations, veuillez consulter [Gérez les flux de données sur les appareils principaux de Greengrass](#).

Note

L'exemple présenté dans cette section ne fonctionne que sur les appareils principaux de Linux.

Pour utiliser le SDK du gestionnaire de flux dans un composant de conteneur Docker, vous devez exécuter le conteneur Docker avec les paramètres suivants :

- Fournissez l'accès au réseau hôte à l'aide de l'option `--network=host`. Cette option permet au conteneur Docker d'interagir avec le composant du gestionnaire de flux via une connexion TLS locale. Cet argument ne fonctionne que sur Docker pour Linux

Warning

Cette option permet au conteneur d'accéder à toutes les interfaces réseau locales de l'hôte. Elle est donc moins sécurisée que si vous exécutez des conteneurs Docker sans cet accès au réseau hôte. Tenez-en compte lorsque vous développez et exécutez des composants de conteneur Docker qui utilisent cette option. Pour plus d'informations, consultez [Network : host](#) dans la documentation Docker.

- Si vous configurez le composant du gestionnaire de flux pour exiger une authentification, qui est le comportement par défaut, définissez la variable d'environnement `AWS_CONTAINER_CREDENTIALS_FULL_URI` sur la valeur que le noyau Greengrass fournit aux composants. Pour plus d'informations, consultez la section [Configuration du gestionnaire de flux](#).

- Si vous configurez le composant du gestionnaire de flux pour utiliser un port autre que celui par défaut, utilisez la [communication interprocessus \(IPC\)](#) pour obtenir le port à partir de la configuration du composant du gestionnaire de flux. Vous devez exécuter le conteneur Docker avec des options supplémentaires pour utiliser IPC. Pour plus d'informations, consultez les ressources suivantes :
- [Connectez-vous au gestionnaire de flux dans le code de l'application](#)
- [Utiliser la communication interprocessus dans les composants du conteneur Docker](#)

Exemple Exemple de recette : Diffuser un fichier vers un compartiment S3 dans un composant de conteneur Docker (Python)

La recette suivante définit un exemple de composant de conteneur Docker qui crée un fichier et le diffuse vers un compartiment S3. Cette recette possède les propriétés suivantes :

- Le composant du gestionnaire de flux en tant que dépendance. Cette dépendance permet au composant d'utiliser le SDK du gestionnaire de flux pour interagir avec le composant du gestionnaire de flux.
- Artefact de composant qui spécifie une image Docker en tant qu'archive TAR dans Amazon S3.
- Script d'installation du cycle de vie qui charge l'image Docker depuis l'archive TAR.
- Script d'exécution du cycle de vie qui exécute un conteneur Docker à partir de l'image. La commande [Docker run](#) comporte les arguments suivants :
- L'-network=host argument fournit au conteneur l'accès au réseau hôte, afin que le conteneur puisse se connecter au composant du gestionnaire de flux.
- Le premier -e argument définit la variable d'AWS_CONTAINER_AUTHORIZATION_TOKEN environnement requise dans le conteneur Docker.
- Les -e arguments supplémentaires définissent les variables d'environnement utilisées dans cet exemple.
- L'-v argument monte le [dossier de travail](#) du composant dans le conteneur. Cet exemple crée un fichier dans le dossier de travail pour le télécharger sur Amazon S3 à l'aide du gestionnaire de flux.
- L'-rm argument nettoie le conteneur à sa sortie.

JSON

```
{
```



```

"RecipeFormatVersion": "2020-01-25",
"ComponentName": "com.example.python.docker.StreamFileToS3",
"ComponentVersion": "1.0.0",
"ComponentDescription": "Creates a text file and uses stream manager to stream the
file to S3.",
"ComponentPublisher": "Amazon",
"ComponentDependencies": {
  "aws.greengrass.StreamManager": {
    "VersionRequirement": "^2.0.0",
    "DependencyType": "HARD"
  }
},
"ComponentConfiguration": {
  "DefaultConfiguration": {
    "bucketName": ""
  }
},
"Manifests": [
  {
    "Platform": {
      "os": "linux"
    },
    "Lifecycle": {
      "install": "docker load -i {artifacts:path}/stream-file-to-s3.tar",
      "Run": "docker run --network=host -e AWS_CONTAINER_AUTHORIZATION_TOKEN
-e BUCKET_NAME=\"{configuration:/bucketName}\" -e WORK_PATH=\"{work:path}\" -v
{work:path}:{work:path} --rm stream-file-to-s3"
    },
    "Artifacts": [
      {
        "URI": "s3://amzn-s3-demo-bucket/artifacts/
com.example.python.docker.StreamFileToS3/1.0.0/stream-file-to-s3.tar"
      }
    ]
  }
]
}

```

YAML

```

RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.python.docker.StreamFileToS3
ComponentVersion: 1.0.0

```

```

ComponentDescription: Creates a text file and uses stream manager to stream the file
to S3.
ComponentPublisher: Amazon
ComponentDependencies:
  aws.greengrass.StreamManager:
    VersionRequirement: ^2.0.0
    DependencyType: HARD
ComponentConfiguration:
  DefaultConfiguration:
    bucketName: ''
Manifests:
  - Platform:
    os: linux
  Lifecycle:
    install: 'docker load -i {artifacts:path}/stream-file-to-s3.tar'
    Run: |
      docker run \
        --network=host \
        -e AWS_CONTAINER_AUTHORIZATION_TOKEN \
        -e BUCKET_NAME="{configuration:/bucketName}" \
        -e WORK_PATH="{work:path}" \
        -v {work:path}:{work:path} \
        --rm stream-file-to-s3
  Artifacts:
    - URI: s3://amzn-s3-demo-bucket/artifacts/
      com.example.python.docker.StreamFileToS3/1.0.0/stream-file-to-s3.tar

```

AWS IoT Greengrass référence de recette de composant

La recette du composant est un fichier qui définit les détails, les dépendances, les artefacts et les cycles de vie d'un composant. Le cycle de vie du composant indique les commandes à exécuter pour installer, exécuter et arrêter le composant, par exemple. Le AWS IoT Greengrass noyau utilise les cycles de vie que vous définissez dans la recette pour installer et exécuter les composants. Le AWS IoT Greengrass service utilise la recette pour identifier les dépendances et les artefacts à déployer sur vos appareils principaux lorsque vous déployez le composant.

Dans la recette, vous pouvez définir des dépendances et des cycles de vie uniques pour chaque plate-forme prise en charge par un composant. Vous pouvez utiliser cette fonctionnalité pour déployer un composant sur des appareils dotés de plusieurs plateformes ayant des exigences différentes. Vous pouvez également l'utiliser pour AWS IoT Greengrass empêcher l'installation d'un composant sur des appareils qui ne le prennent pas en charge.

Chaque recette contient une liste de manifestes. Chaque manifeste spécifie un ensemble d'exigences relatives à la plate-forme, ainsi que le cycle de vie et les artefacts à utiliser pour les principaux appareils dont la plate-forme répond à ces exigences. Le périphérique principal utilise le premier manifeste avec les exigences de plate-forme auxquelles le périphérique répond. Spécifiez un manifeste sans aucune exigence de plate-forme pour correspondre à un périphérique principal.

Vous pouvez également spécifier un cycle de vie global qui ne figure pas dans un manifeste. Dans le cycle de vie global, vous pouvez utiliser des clés de sélection qui identifient les sous-sections du cycle de vie. Vous pouvez ensuite spécifier ces clés de sélection dans un manifeste afin d'utiliser ces sections du cycle de vie global en plus du cycle de vie du manifeste. Le périphérique principal utilise les clés de sélection du manifeste uniquement si le manifeste ne définit pas de cycle de vie. Vous pouvez utiliser la `all` sélection dans un manifeste pour faire correspondre des sections du cycle de vie global sans clé de sélection.

Une fois que le logiciel AWS IoT Greengrass Core a sélectionné un manifeste correspondant au périphérique principal, il procède comme suit pour identifier les étapes du cycle de vie à utiliser :

- Si le manifeste sélectionné définit un cycle de vie, le périphérique principal utilise ce cycle de vie.
- Si le manifeste sélectionné ne définit pas de cycle de vie, le périphérique principal utilise le cycle de vie global. Le dispositif principal effectue les opérations suivantes pour identifier les sections du cycle de vie global à utiliser :
 - Si le manifeste définit des clés de sélection, le périphérique principal utilise les sections du cycle de vie global qui contiennent les clés de sélection du manifeste.
 - Si le manifeste ne définit pas de clés de sélection, le périphérique principal utilise les sections du cycle de vie global qui ne comportent pas de clés de sélection. Ce comportement est équivalent à un manifeste qui définit la `all` sélection.

Important

Un périphérique principal doit répondre aux exigences de plate-forme d'au moins un manifeste pour installer le composant. Si aucun manifeste ne correspond au périphérique principal, le logiciel AWS IoT Greengrass Core n'installe pas le composant et le déploiement échoue.

Vous pouvez définir des recettes au format [JSON](#) ou [YAML](#). La section des exemples de recettes inclut des recettes dans chaque format.

Rubriques

- [Validation des recettes](#)
- [Format de recette](#)
- [Variables de recette](#)
- [Exemples de recettes](#)

Validation des recettes

Greengrass valide une recette de composant JSON ou YAML lors de la création d'une version de composant. Cette validation de recette vérifie la présence d'erreurs courantes dans la recette de votre composant JSON ou YAML afin d'éviter d'éventuels problèmes de déploiement. La validation vérifie la recette pour détecter les erreurs courantes (par exemple, les virgules, les accolades et les champs manquants) et pour s'assurer que la recette est bien formée.

Si vous recevez un message d'erreur de validation de recette, vérifiez qu'il n'y a pas de virgules, d'accolades ou de champs manquants dans votre recette. Vérifiez qu'aucun champ ne vous manque en examinant le [format de la recette](#).

Format de recette

Lorsque vous définissez une recette pour un composant, vous spécifiez les informations suivantes dans le document de recette. La même structure s'applique aux recettes aux formats YAML et JSON.

RecipeFormatVersion

Version du modèle pour la recette. Choisissez l'option suivante :

- 2020-01-25

ComponentName

Nom du composant défini par cette recette. Le nom du composant doit être unique Compte AWS dans votre région.

Conseils

- Utilisez le format de nom de domaine inverse pour éviter les collisions de noms au sein de votre entreprise. Par exemple, si votre entreprise possède un projet d'énergie solaire `example.com` et que vous y travaillez, vous pouvez donner un nom à votre composant

Hello Worldcom.example.solar.HelloWorld. Cela permet d'éviter les collisions de noms de composants au sein de votre entreprise.

- Évitez le `aws.greengrass` préfixe dans les noms de vos composants. AWS IoT Greengrass utilise ce préfixe pour les [composants publics](#) qu'il fournit. Si vous choisissez le même nom qu'un composant public, celui-ci le remplace. AWS IoT Greengrass Fournit ensuite votre composant au lieu du composant public lorsqu'il déploie des composants dépendants de ce composant public. Cette fonctionnalité vous permet de modifier le comportement des composants publics, mais elle peut également endommager d'autres composants si vous n'avez pas l'intention de remplacer un composant public.

ComponentVersion

Version du composant. La valeur maximale pour les valeurs majeures, mineures et patchs est de 999999.

Note

AWS IoT Greengrass utilise des versions sémantiques pour les composants. Les versions sémantiques suivent une majeure. mineur. système de numéro de patch. Par exemple, la version `1.0.0` représente la première version majeure d'un composant. Pour plus d'informations, consultez la [spécification de version sémantique](#).

ComponentDescription

(Facultatif) Description du composant.

ComponentPublisher

L'éditeur ou l'auteur du composant.

ComponentConfiguration

(Facultatif) Objet qui définit la configuration ou les paramètres du composant. Vous définissez la configuration par défaut, puis lorsque vous déployez le composant, vous pouvez spécifier l'objet de configuration à fournir au composant. La configuration des composants prend en charge les paramètres et les structures imbriqués. Cet objet contient les informations suivantes :

DefaultConfiguration

Objet qui définit la configuration par défaut du composant. Vous définissez la structure de cet objet.

Note

AWS IoT Greengrass utilise JSON pour les valeurs de configuration. Le JSON spécifie un type de nombre mais ne fait pas la différence entre les nombres entiers et les nombres flottants. Par conséquent, les valeurs de configuration peuvent être converties en valeurs flottantes. AWS IoT Greengrass Pour garantir que votre composant utilise le type de données approprié, nous vous recommandons de définir les valeurs de configuration numériques sous forme de chaînes. Ensuite, demandez à votre composant de les analyser sous forme de nombres entiers ou de nombres flottants. Cela garantit que vos valeurs de configuration sont du même type dans la configuration et sur votre appareil principal.

ComponentDependencies

(Facultatif) Un dictionnaire d'objets qui définissent chacun une dépendance de composant pour le composant. La clé de chaque objet identifie le nom de la dépendance du composant. AWS IoT Greengrass installe les dépendances des composants lors de l'installation du composant. AWS IoT Greengrass attend le début des dépendances avant de démarrer le composant. Chaque objet contient les informations suivantes :

VersionRequirement

La contrainte de version sémantique de style npm qui définit les versions de composants compatibles pour cette dépendance. Vous pouvez spécifier une version ou une plage de versions. Pour plus d'informations, consultez le calculateur de [version sémantique npm](#).

DependencyType

(Facultatif) Type de cette dépendance. Choisissez parmi les options suivantes.

- SOFT – Le composant ne redémarre pas si la dépendance change d'état.
- HARD – Le composant redémarre si la dépendance change d'état.

La valeur par défaut est HARD.

ComponentType

(Facultatif) Type de composant.

Note

Il est déconseillé de spécifier le type de composant dans une recette. AWS IoT Greengrass définit le type pour vous lorsque vous créez un composant.

Il peut s'agir de l'un des types suivants :

- `aws.greengrass.generic`— Le composant exécute des commandes ou fournit des artefacts.
- `aws.greengrass.lambda`— Le composant exécute une fonction Lambda à l'aide du composant [Lambda](#) Launcher. Le `ComponentSource` paramètre spécifie l'ARN de la fonction Lambda exécutée par ce composant.

Nous vous déconseillons d'utiliser cette option, car elle est définie AWS IoT Greengrass lorsque vous créez un composant à partir d'une fonction Lambda. Pour de plus amples informations, veuillez consulter [Exécuter AWS Lambda des fonctions](#).

- `aws.greengrass.plugin`— Le composant s'exécute dans la même machine virtuelle Java (JVM) que le noyau Greengrass. Si vous déployez ou redémarrez un composant du plugin, le noyau Greengrass redémarre.

Les composants du plugin utilisent le même fichier journal que le noyau Greengrass. Pour de plus amples informations, veuillez consulter [AWS IoT Greengrass Journaux de surveillance](#).

Nous vous déconseillons d'utiliser cette option dans les recettes de composants, car elle est destinée aux composants AWS fournis écrits en Java qui s'interfaçent directement avec le noyau de Greengrass. Pour plus d'informations sur les composants publics qui sont des plug-ins, consultez [AWS-composants fournis](#).

- `aws.greengrass.nucleus`— Le composant du noyau. Pour de plus amples informations, veuillez consulter [Noyau de Greengrass](#).

Nous vous déconseillons d'utiliser cette option dans les recettes de composants. Il est destiné au composant Greengrass nucleus, qui fournit les fonctionnalités minimales du logiciel AWS IoT Greengrass Core.

La valeur par défaut `aws.greengrass.generic` est lorsque vous créez un composant à partir d'une recette ou `aws.greengrass.lambda` lorsque vous créez un composant à partir d'une fonction Lambda.

Pour de plus amples informations, veuillez consulter [Types de composants](#).

ComponentSource

(Facultatif) L'ARN de la fonction Lambda exécutée par un composant.

Il est déconseillé de spécifier la source du composant dans une recette. AWS IoT Greengrass définit ce paramètre pour vous lorsque vous créez un composant à partir d'une fonction Lambda. Pour de plus amples informations, veuillez consulter [Exécuter AWS Lambda des fonctions](#).

Manifests

Liste d'objets qui définissent chacun le cycle de vie, les paramètres et les exigences d'un composant pour une plate-forme. Si un périphérique principal répond aux exigences de plate-forme de plusieurs manifestes, AWS IoT Greengrass utilise le premier manifeste correspondant au périphérique principal. Pour garantir que les appareils principaux utilisent le bon manifeste, définissez d'abord les manifestes avec des exigences de plate-forme plus strictes. Un manifeste qui s'applique à toutes les plateformes doit être le dernier manifeste de la liste.

Important

Un périphérique principal doit répondre aux exigences de plate-forme d'au moins un manifeste pour installer le composant. Si aucun manifeste ne correspond au périphérique principal, le logiciel AWS IoT Greengrass Core n'installe pas le composant et le déploiement échoue.

Chaque objet contient les informations suivantes :

Name

(Facultatif) Nom convivial pour la plate-forme définie dans ce manifeste.

Si vous omettez ce paramètre, AWS IoT Greengrass crée un nom à partir de la plateforme `os` et `architecture`.

Platform

(Facultatif) Objet qui définit la plate-forme à laquelle s'applique ce manifeste. Omettez ce paramètre pour définir un manifeste qui s'applique à toutes les plateformes.

Cet objet spécifie les paires clé-valeur relatives à la plate-forme sur laquelle un périphérique principal s'exécute. Lorsque vous déployez ce composant, le logiciel AWS IoT Greengrass Core compare ces paires clé-valeur avec les attributs de plate-forme du périphérique principal. Le logiciel de AWS IoT Greengrass base définit toujours `os` et `architecture`, et il peut définir des attributs supplémentaires. Vous pouvez spécifier des attributs de plate-forme personnalisés pour un appareil principal lorsque vous déployez le composant Greengrass nucleus. Pour plus d'informations, consultez le [paramètre Platform overrides](#) du composant [Greengrass nucleus](#).

Pour chaque paire clé-valeur, vous pouvez spécifier l'une des valeurs suivantes :

- Une valeur exacte, telle que `linux` ou `windows`. Les valeurs exactes doivent commencer par une lettre ou un chiffre.
- `*`, qui correspond à n'importe quelle valeur. Cela correspond également lorsqu'une valeur n'est pas présente.
- Une expression régulière de style Java, telle que `/windows|linux/`. L'expression régulière doit commencer et se terminer par une barre oblique (`/`). Par exemple, l'expression régulière `/.+ /` correspond à toute valeur non vide.

Cet objet contient les informations suivantes :

`runtime`

Le [runtime Greengrass Nucleus](#) pour la plate-forme prise en charge par ce manifeste. Lors de la définition de plusieurs manifestes avec la plate-forme `runtime`, les valeurs d'exécution prises en charge dans une recette sont `aws_nucleus_lite` et `*` uniquement. Pour cibler un appareil classique, le champ d'exécution NE DOIT PAS être spécifié dans la recette. Les environnements d'exécution Greengrass Nucleus pris en charge incluent les valeurs suivantes :

- `*`
- `aws_nucleus_lite`

`os`

(Facultatif) Nom du système d'exploitation de la plate-forme prise en charge par ce manifeste. Les plateformes courantes incluent les valeurs suivantes :

- linux
- windows
- darwin (macOS)

architecture

(Facultatif) Architecture du processeur de la plate-forme prise en charge par ce manifeste. Les architectures courantes incluent les valeurs suivantes :

- amd64
- arm
- aarch64
- x86

architecture.detail

(Facultatif) Détail de l'architecture du processeur pour la plate-forme prise en charge par ce manifeste. Les détails courants de l'architecture incluent les valeurs suivantes :

- arm61
- arm71
- arm81

key

(Facultatif) Attribut de plateforme que vous définissez pour ce manifeste. Remplacez **Key** par le nom de l'attribut de plateforme. Le logiciel AWS IoT Greengrass Core associe cet attribut de plate-forme aux paires clé-valeur que vous spécifiez dans la configuration des composants du noyau Greengrass. Pour plus d'informations, consultez le [paramètre Platform overrides](#) du composant [Greengrass nucleus](#).

Tip

Utilisez le format de nom de domaine inverse pour éviter les collisions de noms au sein de votre entreprise. Par exemple, si votre entreprise possède un projet radio `example.com` et que vous y travaillez, vous pouvez nommer un attribut de plateforme personnalisé `com.example.radio.RadioModule`. Cela permet d'éviter les collisions entre les noms d'attributs de plateforme au sein de votre entreprise.

Par exemple, vous pouvez définir un attribut de plate-forme pour spécifier un manifeste différent en fonction du module radio disponible sur un périphérique principal.

`com.example.radio.RadioModule` Chaque manifeste peut inclure différents artefacts qui s'appliquent à différentes configurations matérielles, afin que vous puissiez déployer l'ensemble minimal de logiciels sur le périphérique principal.

Lifecycle

Objet ou chaîne qui définit comment installer et exécuter le composant sur la plate-forme définie par ce manifeste. Vous pouvez également définir un [cycle de vie global](#) qui s'applique à toutes les plateformes. Le périphérique principal utilise le cycle de vie global uniquement si le manifeste à utiliser ne spécifie pas de cycle de vie.

Note

Vous définissez ce cycle de vie dans un manifeste. Les étapes du cycle de vie que vous spécifiez ici s'appliquent uniquement à la plate-forme définie par ce manifeste. Vous pouvez également définir un [cycle de vie global](#) qui s'applique à toutes les plateformes.

Cet objet ou cette chaîne contient les informations suivantes :

`Setenv`

(Facultatif) Un dictionnaire de variables d'environnement à fournir à tous les scripts de cycle de vie. Vous pouvez remplacer ces variables d'environnement `Setenv` dans chaque script de cycle de vie.

`install`

(Facultatif) Objet ou chaîne qui définit le script à exécuter lors de l'installation du composant. Le logiciel de AWS IoT Greengrass base exécute également cette étape du cycle de vie à chaque lancement du logiciel.

Si le `install` script se termine avec un code de réussite, le composant passe à l'`INSTALLED` état.

Cet objet ou cette chaîne contient les informations suivantes :

`Script`

Le script à exécuter.

RequiresPrivilege

(Facultatif) Vous pouvez exécuter le script avec les privilèges root. Si vous définissez cette option sur `true`, le logiciel AWS IoT Greengrass Core exécute ce script de cycle de vie en tant qu'utilisateur root plutôt qu'en tant qu'utilisateur système que vous configurez pour exécuter ce composant. La valeur par défaut est `false`.

SkipIf

(Facultatif) Vérification visant à déterminer s'il faut exécuter le script ou non. Vous pouvez définir pour vérifier si un exécutable se trouve sur le chemin ou si un fichier existe. Si le résultat est vrai, le logiciel AWS IoT Greengrass Core ignore l'étape. Choisissez l'une des vérifications suivantes :

- `onpath` *runnable*— Vérifie si un exécutable se trouve sur le chemin du système. Par exemple, utilisez cette option `onpath python3` pour ignorer cette étape du cycle de vie si Python 3 est disponible.
- `exists` *file*— Vérifie si un fichier existe. Par exemple, utilisez-le `exists /tmp/my-configuration.db` pour ignorer cette étape du cycle de vie si elle `/tmp/my-configuration.db` est présente.

Timeout

(Facultatif) Durée maximale en secondes pendant laquelle le script peut être exécuté avant que le logiciel AWS IoT Greengrass Core n'arrête le processus.

Par défaut : 120 secondes

Setenv

(Facultatif) Le dictionnaire des variables d'environnement à fournir au script. Ces variables d'environnement remplacent les variables que vous fournissez. `Lifecycle.Setenv`

run

(Facultatif) Objet ou chaîne qui définit le script à exécuter au démarrage du composant.

Le composant entre dans cet `RUNNING` état lors de l'exécution de cette étape du cycle de vie. Si le `run` script se termine avec un code de réussite, le composant passe à l'`STOPPING` état. Si un `shutdown` script est spécifié, il s'exécute ; dans le cas contraire, le composant passe à l'`FINISHED` état.

Les composants qui dépendent de ce composant démarrent lors de l'exécution de cette étape du cycle de vie. Pour exécuter un processus en arrière-plan, tel qu'un service utilisé par des composants dépendants, utilisez plutôt l'étape `startup` du cycle de vie.

Lorsque vous déployez des composants dotés d'un `run` cycle de vie, le périphérique principal peut signaler que le déploiement est terminé dès que ce script de cycle de vie s'exécute. Par conséquent, le déploiement peut être complet et réussi même si le script de `run` cycle de vie échoue peu après son exécution. Si vous souhaitez que l'état du déploiement dépende du résultat du script de démarrage du composant, utilisez plutôt l'étape `startup` du cycle de vie.

 Note

Vous ne pouvez en définir qu'un `startup` ou un `run` cycle de vie.

Cet objet ou cette chaîne contient les informations suivantes :

Script

Le script à exécuter.

RequiresPrivilege

(Facultatif) Vous pouvez exécuter le script avec les privilèges root. Si vous définissez cette option sur `true`, le logiciel AWS IoT Greengrass Core exécute ce script de cycle de vie en tant qu'utilisateur root plutôt qu'en tant qu'utilisateur système que vous configurez pour exécuter ce composant. La valeur par défaut est `false`.

SkipIf

(Facultatif) Vérification visant à déterminer s'il faut exécuter le script ou non. Vous pouvez définir pour vérifier si un exécutable se trouve sur le chemin ou si un fichier existe. Si le résultat est vrai, le logiciel AWS IoT Greengrass Core ignore l'étape.

Choisissez l'une des vérifications suivantes :

- `onpath` *runnable*— Vérifie si un exécutable se trouve sur le chemin du système. Par exemple, utilisez cette option `onpath python3` pour ignorer cette étape du cycle de vie si Python 3 est disponible.
- `exists` *file*— Vérifie si un fichier existe. Par exemple, utilisez-le `exists /tmp/my-configuration.db` pour ignorer cette étape du cycle de vie si elle `/tmp/my-configuration.db` est présente.

Timeout

(Facultatif) Durée maximale en secondes pendant laquelle le script peut être exécuté avant que le logiciel AWS IoT Greengrass Core n'arrête le processus.

Cette étape du cycle de vie n'expire pas par défaut. Si vous omettez ce délai, le `run` script s'exécute jusqu'à sa fin.

Setenv

(Facultatif) Le dictionnaire des variables d'environnement à fournir au script.

Ces variables d'environnement remplacent les variables que vous fournissez.

`Lifecycle.Setenv`

startup

(Facultatif) Objet ou chaîne qui définit le processus d'arrière-plan à exécuter au démarrage du composant.

`startup` À utiliser pour exécuter une commande qui doit se terminer correctement ou mettre à jour l'état du composant pour `RUNNING` que les composants dépendants puissent démarrer. Utilisez l'opération [UpdateState](#) IPC pour définir l'état du composant sur `RUNNING` ou `ERROR` quand le composant lance un script qui ne se ferme pas. Par exemple, vous pouvez définir une `startup` étape qui lance le processus MySQL avec `/etc/init.d/mysqld start`.

Le composant entre dans cet `STARTING` état lors de l'exécution de cette étape du cycle de vie. Si le `startup` script se termine avec un code de réussite, le composant passe à l'`RUNNING` état. Les composants dépendants peuvent alors démarrer.

Lorsque vous déployez des composants dotés d'un `startup` cycle de vie, le périphérique principal peut signaler que le déploiement est terminé une fois que ce script de cycle de vie se termine ou indique son état. En d'autres termes, le déploiement se poursuit `IN_PROGRESS` jusqu'à ce que les scripts de démarrage de tous les composants quittent ou signalent un état.

Note

Vous ne pouvez en définir qu'un `startup` ou un `run` cycle de vie.

Cet objet ou cette chaîne contient les informations suivantes :

Script

Le script à exécuter.

RequiresPrivilege

(Facultatif) Vous pouvez exécuter le script avec les privilèges root. Si vous définissez cette option sur `true`, le logiciel AWS IoT Greengrass Core exécute ce script de cycle de vie en tant qu'utilisateur root plutôt qu'en tant qu'utilisateur système que vous configurez pour exécuter ce composant. La valeur par défaut est `false`.

Skipif

(Facultatif) Vérification visant à déterminer s'il faut exécuter le script ou non. Vous pouvez définir pour vérifier si un exécutable se trouve sur le chemin ou si un fichier existe. Si le résultat est vrai, le logiciel AWS IoT Greengrass Core ignore l'étape. Choisissez l'une des vérifications suivantes :

- `onpath` ***runnable***— Vérifie si un exécutable se trouve sur le chemin du système. Par exemple, utilisez cette option `onpath python3` pour ignorer cette étape du cycle de vie si Python 3 est disponible.
- `exists` ***file***— Vérifie si un fichier existe. Par exemple, utilisez-le `exists /tmp/my-configuration.db` pour ignorer cette étape du cycle de vie si elle `/tmp/my-configuration.db` est présente.

Timeout

(Facultatif) Durée maximale en secondes pendant laquelle le script peut être exécuté avant que le logiciel AWS IoT Greengrass Core n'arrête le processus.

Par défaut : 120 secondes

Setenv

(Facultatif) Le dictionnaire des variables d'environnement à fournir au script. Ces variables d'environnement remplacent les variables que vous fournissez.
`Lifecycle.Setenv`

shutdown

(Facultatif) Objet ou chaîne qui définit le script à exécuter lorsque le composant s'arrête. Utilisez le cycle de vie d'arrêt pour exécuter le code que vous souhaitez exécuter lorsque le

composant est dans STOPPING cet état. Le cycle de vie d'arrêt peut être utilisé pour arrêter un processus lancé par les run scripts startup or.

Si vous lancez un processus en arrière-plan dans startup, utilisez l'shutdown étape pour arrêter ce processus lorsque le composant s'arrête. Par exemple, vous pouvez définir une shutdown étape qui arrête le processus MySQL avec `/etc/init.d/mysqld stop`.

Le shutdown script s'exécute une fois que le composant est entré dans l'STOPPING état. Si le script s'exécute correctement, le composant passe à l'FINISHED état.

Cet objet ou cette chaîne contient les informations suivantes :

Script

Le script à exécuter.

RequiresPrivilege

(Facultatif) Vous pouvez exécuter le script avec les privilèges root. Si vous définissez cette option sur `true`, le logiciel AWS IoT Greengrass Core exécute ce script de cycle de vie en tant qu'utilisateur root plutôt qu'en tant qu'utilisateur système que vous configurez pour exécuter ce composant. La valeur par défaut est `false`.

Skipif

(Facultatif) Vérification visant à déterminer s'il faut exécuter le script ou non. Vous pouvez définir pour vérifier si un exécutable se trouve sur le chemin ou si un fichier existe. Si le résultat est vrai, le logiciel AWS IoT Greengrass Core ignore l'étape. Choisissez l'une des vérifications suivantes :

- `onpath runnable`— Vérifie si un exécutable se trouve sur le chemin du système. Par exemple, utilisez cette option `onpath python3` pour ignorer cette étape du cycle de vie si Python 3 est disponible.
- `exists file`— Vérifie si un fichier existe. Par exemple, utilisez-le `exists /tmp/my-configuration.db` pour ignorer cette étape du cycle de vie si elle `/tmp/my-configuration.db` est présente.

Timeout

(Facultatif) Durée maximale en secondes pendant laquelle le script peut être exécuté avant que le logiciel AWS IoT Greengrass Core n'arrête le processus.

Par défaut : 15 secondes.

Setenv

(Facultatif) Le dictionnaire des variables d'environnement à fournir au script. Ces variables d'environnement remplacent les variables que vous fournissez.

Lifecycle.Setenv

recover

(Facultatif) Objet ou chaîne qui définit le script à exécuter lorsque le composant rencontre une erreur.

Cette étape s'exécute lorsqu'un composant entre dans `ERROR` cet état. Si le composant devient `ERROR` trois fois sans restauration réussie, le composant passe à l'`BROKEN` état actuel. Pour réparer un `BROKEN` composant, vous devez le déployer à nouveau.

Cet objet ou cette chaîne contient les informations suivantes :

Script

Le script à exécuter.

RequiresPrivilege

(Facultatif) Vous pouvez exécuter le script avec les privilèges root. Si vous définissez cette option sur `true`, le logiciel AWS IoT Greengrass Core exécute ce script de cycle de vie en tant qu'utilisateur root plutôt qu'en tant qu'utilisateur système que vous configurez pour exécuter ce composant. La valeur par défaut est `false`.

Skipif

(Facultatif) Vérification visant à déterminer s'il faut exécuter le script ou non. Vous pouvez définir pour vérifier si un exécutable se trouve sur le chemin ou si un fichier existe. Si le résultat est vrai, le logiciel AWS IoT Greengrass Core ignore l'étape. Choisissez l'une des vérifications suivantes :

- `onpath` *runnable*— Vérifie si un exécutable se trouve sur le chemin du système. Par exemple, utilisez cette option `onpath python3` pour ignorer cette étape du cycle de vie si Python 3 est disponible.
- `exists` *file*— Vérifie si un fichier existe. Par exemple, utilisez-le `exists /tmp/my-configuration.db` pour ignorer cette étape du cycle de vie si elle `/tmp/my-configuration.db` est présente.

Timeout

(Facultatif) Durée maximale en secondes pendant laquelle le script peut être exécuté avant que le logiciel AWS IoT Greengrass Core n'arrête le processus.

Par défaut : 60 secondes.

Setenv

(Facultatif) Le dictionnaire des variables d'environnement à fournir au script.

Ces variables d'environnement remplacent les variables que vous fournissez.

`Lifecycle.Setenv`

bootstrap

(Facultatif) Objet ou chaîne qui définit un script nécessitant le AWS IoT Greengrass redémarrage du logiciel ou du périphérique principal. Cela vous permet de développer un composant qui redémarre après avoir installé des mises à jour du système d'exploitation ou des mises à jour d'exécution, par exemple.

Note

Pour installer des mises à jour ou des dépendances qui ne nécessitent pas le redémarrage du logiciel ou de l'appareil AWS IoT Greengrass Core, utilisez le [cycle de vie d'installation](#).

Cette étape du cycle de vie s'exécute avant l'étape du cycle de vie d'installation dans les cas suivants lorsque le logiciel AWS IoT Greengrass Core déploie le composant :

- Le composant est déployé sur le périphérique principal pour la première fois.
- La version du composant change.
- Le script bootstrap change à la suite d'une mise à jour de la configuration d'un composant.

Une fois que le logiciel AWS IoT Greengrass Core a terminé l'étape d'amorçage pour tous les composants dotés d'une étape d'amorçage lors d'un déploiement, le logiciel redémarre.

Important

Vous devez configurer le logiciel AWS IoT Greengrass Core en tant que service système pour redémarrer le logiciel AWS IoT Greengrass Core ou le périphérique

principal. Si vous ne configurez pas le logiciel AWS IoT Greengrass Core en tant que service système, le logiciel ne redémarrera pas. Pour de plus amples informations, veuillez consulter [Configurer le noyau Greengrass en tant que service système](#).

Cet objet ou cette chaîne contient les informations suivantes :

`BootstrapOnRollback`

 Note

Lorsque cette fonctionnalité est activée, elle ne `BootstrapOnRollback` s'exécute que pour les composants qui ont terminé ou tenté d'exécuter les étapes du cycle de vie du bootstrap dans le cadre d'un déploiement cible ayant échoué. Cette fonctionnalité est disponible pour les versions 2.12.0 et ultérieures de Greengrass nucleus.

(Facultatif) Vous pouvez exécuter les étapes du cycle de vie du bootstrap dans le cadre d'un déploiement rétroactif. Si vous définissez cette option sur `true`, les étapes du cycle de vie du bootstrap définies dans le cadre d'un déploiement de restauration seront exécutées. En cas d'échec d'un déploiement, la version précédente du cycle de vie d'amorçage du composant s'exécute à nouveau lors d'un déploiement rétroactif.

La valeur par défaut est `false`.

`Script`

Le script à exécuter. Le code de sortie de ce script définit l'instruction de redémarrage. Utilisez les codes de sortie suivants :

- `0`— Ne redémarrez pas le logiciel AWS IoT Greengrass principal ou le périphérique principal. Le logiciel AWS IoT Greengrass Core redémarre toujours après le démarrage de tous les composants.
- `100`— Demande de redémarrage du logiciel AWS IoT Greengrass Core.
- `101`— Demande de redémarrage de l'appareil principal.

Les codes de sortie 100 à 199 sont réservés à un comportement spécial. Les autres codes de sortie représentent des erreurs de script.

RequiresPrivilege

(Facultatif) Vous pouvez exécuter le script avec les privilèges root. Si vous définissez cette option sur `true`, le logiciel AWS IoT Greengrass Core exécute ce script de cycle de vie en tant qu'utilisateur root plutôt qu'en tant qu'utilisateur système que vous configurez pour exécuter ce composant. La valeur par défaut est `false`.

Timeout

(Facultatif) Durée maximale en secondes pendant laquelle le script peut être exécuté avant que le logiciel AWS IoT Greengrass Core n'arrête le processus.

Par défaut : 120 secondes

Setenv

(Facultatif) Le dictionnaire des variables d'environnement à fournir au script. Ces variables d'environnement remplacent les variables que vous fournissez.

`Lifecycle.Setenv`

Selections

(Facultatif) Liste de clés de sélection qui spécifient les sections du [cycle de vie global](#) à exécuter pour ce manifeste. Dans le cycle de vie global, vous pouvez définir les étapes du cycle de vie à l'aide de clés de sélection à n'importe quel niveau pour sélectionner des sous-sections du cycle de vie. Ensuite, le périphérique principal utilise les sections qui correspondent aux clés de sélection de ce manifeste. Pour plus d'informations, consultez les [exemples de cycle de vie global](#).

Important

Le périphérique principal utilise les sélections du cycle de vie global uniquement si ce manifeste ne définit pas de cycle de vie.

Vous pouvez spécifier la clé `all` de sélection pour exécuter les sections du cycle de vie global dépourvues de clés de sélection.

Artifacts

(Facultatif) Liste d'objets qui définissent chacun un artefact binaire pour le composant de la plate-forme défini par ce manifeste. Par exemple, vous pouvez définir du code ou des images comme des artefacts.

Lorsque le composant est déployé, le logiciel AWS IoT Greengrass Core télécharge l'artefact dans un dossier du périphérique principal. Vous pouvez également définir les artefacts comme des fichiers d'archive que le logiciel extrait après les avoir téléchargés.

Vous pouvez utiliser [des variables de recette](#) pour obtenir les chemins d'accès aux dossiers dans lesquels les artefacts sont installés sur le périphérique principal.

- Fichiers normaux — Utilisez la [variable de recette `artifacts:path`](#) pour obtenir le chemin d'accès au dossier contenant les artefacts. Par exemple, spécifiez `{artifacts:path}/my_script.py` dans une recette le chemin d'accès à un artefact contenant l'URI `s3://amzn-s3-demo-bucket/path/to/my_script.py`.
- Archives extraites — Utilisez la [variable de recette `Artifacts:DecompressedPath` pour obtenir le chemin](#) d'accès au dossier contenant les artefacts d'archive extraits. Le logiciel AWS IoT Greengrass Core extrait chaque archive dans un dossier portant le même nom que l'archive. Par exemple, spécifiez `{artifacts:decompressedPath}/my_archive/my_script.py` dans une recette le chemin d'accès `my_script.py` à l'artefact d'archive contenant l'URI `s3://amzn-s3-demo-bucket/path/to/my_archive.zip`.

Note

Lorsque vous développez un composant avec un artefact d'archive sur un périphérique principal local, il se peut que vous ne disposiez pas d'URI pour cet artefact. Pour tester votre composant avec une `Unarchive` option qui extrait l'artefact, spécifiez une URI dans laquelle le nom du fichier correspond au nom de votre fichier d'artefact d'archive. Vous pouvez spécifier l'URI dans lequel vous comptez télécharger l'artefact d'archive, ou vous pouvez spécifier un nouvel URI réservé. Par exemple, pour extraire l'`my_archive.zip` artefact lors d'un déploiement local, vous pouvez spécifier `s3://amzn-s3-demo-bucket/my_archive.zip`.

Chaque objet contient les informations suivantes :

Uri

L'URI d'un artefact dans un compartiment S3. Le logiciel AWS IoT Greengrass Core extrait l'artefact à partir de cet URI lors de l'installation du composant, sauf si l'artefact existe déjà sur le périphérique. Chaque artefact doit avoir un nom de fichier unique dans chaque manifeste.

Unarchive

(Facultatif) Type d'archive à décompresser. Sélectionnez parmi les options suivantes :

- **NONE**— Le fichier n'est pas une archive à décompresser. Le logiciel AWS IoT Greengrass Core installe l'artefact dans un dossier du périphérique principal. Vous pouvez utiliser la [variable de recette `artifacts:path`](#) pour obtenir le chemin d'accès à ce dossier.
- **ZIP**— Le fichier est une archive ZIP. Le logiciel AWS IoT Greengrass Core extrait l'archive dans un dossier portant le même nom que l'archive. Vous pouvez utiliser la [variable de recette `Artifacts:DecompressedPath` pour obtenir le chemin](#) d'accès au dossier qui contient ce dossier.

La valeur par défaut est **NONE**.

Permission

(Facultatif) Objet qui définit les autorisations d'accès à définir pour ce fichier d'artefact. Vous pouvez définir l'autorisation de lecture et l'autorisation d'exécution.

Note

Vous ne pouvez pas définir l'autorisation d'écriture, car le logiciel AWS IoT Greengrass Core n'autorise pas les composants à modifier les fichiers d'artefacts dans le dossier d'artefacts. Pour modifier un fichier d'artefact dans un composant, copiez-le dans un autre emplacement ou publiez et déployez un nouveau fichier d'artefact.

Si vous définissez un artefact comme une archive à décompresser, le logiciel AWS IoT Greengrass Core définit ces autorisations d'accès sur les fichiers qu'il décompresse de l'archive. Le logiciel AWS IoT Greengrass Core définit les autorisations d'accès au dossier sur **ALL for Read etExecute**. Cela permet aux composants de visualiser les fichiers décompressés dans le dossier. Pour définir des autorisations sur des fichiers individuels de l'archive, vous pouvez définir les autorisations dans le [script du cycle de vie d'installation](#).

Cet objet contient les informations suivantes :

Read

(Facultatif) L'autorisation de lecture à définir pour ce fichier d'artefact. Pour autoriser d'autres composants à accéder à cet artefact, tels que les composants qui dépendent de ce composant, spécifiez ALL. Sélectionnez parmi les options suivantes :

- NONE— Le fichier n'est pas lisible.
- OWNER— Le fichier est lisible par l'utilisateur du système que vous configurez pour exécuter ce composant.
- ALL— Le fichier est lisible par tous les utilisateurs.

La valeur par défaut est OWNER.

Execute

(Facultatif) L'autorisation d'exécution à définir pour ce fichier d'artefact. L'Execute autorisation implique l'Read autorisation. Par exemple, si vous spécifiez ALL pour Execute, tous les utilisateurs peuvent lire et exécuter ce fichier d'artefact.

Sélectionnez parmi les options suivantes :

- NONE— Le fichier n'est pas exécutable.
- OWNER— Le fichier peut être exécuté par l'utilisateur du système que vous configurez pour exécuter le composant.
- ALL— Le fichier est exécutable par tous les utilisateurs.

La valeur par défaut est NONE.

Digest

(Lecture seule) Le hachage du condensé cryptographique de l'artefact. Lorsque vous créez un composant, il AWS IoT Greengrass utilise un algorithme de hachage pour calculer le hachage du fichier d'artefact. Ensuite, lorsque vous déployez le composant, le noyau Greengrass calcule le hachage de l'artefact téléchargé et compare le hachage avec ce condensé pour vérifier l'artefact avant l'installation. Si le hachage ne correspond pas au résumé, le déploiement échoue.

Si vous définissez ce paramètre, il AWS IoT Greengrass remplace la valeur que vous avez définie lors de la création du composant.

Algorithm

(Lecture seule) Algorithme de hachage AWS IoT Greengrass utilisé pour calculer le hachage condensé de l'artefact.

Si vous définissez ce paramètre, il AWS IoT Greengrass remplace la valeur que vous avez définie lors de la création du composant.

Lifecycle

Objet qui définit le mode d'installation et d'exécution du composant. Le périphérique principal utilise le cycle de vie global uniquement si le [manifeste](#) à utiliser ne spécifie pas de cycle de vie.

Note

Vous définissez ce cycle de vie en dehors d'un manifeste. Vous pouvez également définir un [cycle de vie du manifeste](#) qui s'applique aux plateformes correspondant à ce manifeste.

Dans le cycle de vie global, vous pouvez spécifier les cycles de vie qui s'exécutent pour certaines [clés de sélection](#) que vous spécifiez dans chaque manifeste. Les clés de sélection sont des chaînes qui identifient les sections du cycle de vie global à exécuter pour chaque manifeste.

La clé `all` de sélection est la valeur par défaut pour toutes les sections sans clé de sélection. Cela signifie que vous pouvez spécifier la clé de `all` sélection dans un manifeste pour exécuter les sections du cycle de vie global sans clés de sélection. Il n'est pas nécessaire de spécifier la clé `all` de sélection dans le cycle de vie global.

Si un manifeste ne définit pas de cycle de vie ou de clés de sélection, le périphérique principal utilise par défaut la `all` sélection. Cela signifie que dans ce cas, le périphérique principal utilise les sections du cycle de vie global qui n'utilisent pas de touches de sélection.

Cet objet contient les mêmes informations que le [cycle de vie du manifeste](#), mais vous pouvez spécifier des clés de sélection à n'importe quel niveau pour sélectionner des sous-sections du cycle de vie.

 Tip

Nous vous recommandons de n'utiliser que des lettres minuscules pour chaque clé de sélection afin d'éviter les conflits entre les clés de sélection et les clés de cycle de vie. Les clés du cycle de vie commencent par une majuscule.

Exemple Exemple de cycle de vie global avec des clés de sélection de haut niveau

```
Lifecycle:
  key1:
    install:
      SkipIf: either onpath executable or exists file
      Script: command1
  key2:
    install:
      Script: command2
  all:
    install:
      Script: command3
```

Exemple Exemple de cycle de vie global avec des clés de sélection de bas niveau

```
Lifecycle:
  install:
    Script:
      key1: command1
      key2: command2
      all: command3
```

Exemple Exemple de cycle de vie global avec plusieurs niveaux de clés de sélection

```
Lifecycle:
  key1:
    install:
      SkipIf: either onpath executable or exists file
      Script: command1
  key2:
    install:
      Script: command2
  all:
```

```
install:
  Script:
    key3: command3
    key4: command4
    all: command5
```

Variables de recette

Les variables de recette exposent les informations du composant et du noyau actuels que vous pouvez utiliser dans vos recettes. Par exemple, vous pouvez utiliser une variable de recette pour transmettre les paramètres de configuration des composants à une application que vous exécutez dans un script de cycle de vie.

Vous pouvez utiliser des variables de recette dans les sections suivantes des recettes de composants :

- Définitions du cycle de vie.
- Définitions de configuration des composants, si vous utilisez [Greengrass noyau](#) v2.6.0 ou version ultérieure et que vous définissez l'[interpolateComponentConfiguration](#) option de configuration sur `true`. Vous pouvez également utiliser des variables de recettes lorsque vous [déployez des mises à jour de configuration de composants](#).

Les variables de recette utilisent `{recipe_variable}` la syntaxe. Les bretelles frisées indiquent une variable de recette.

AWS IoT Greengrass prend en charge les variables de recette suivantes :

`component_dependency_name:configuration:json_pointer`

La valeur d'un paramètre de configuration pour le composant défini par cette recette ou pour un composant dont dépend ce composant.

Vous pouvez utiliser cette variable pour fournir un paramètre à un script que vous exécutez pendant le cycle de vie du composant.

Note

AWS IoT Greengrass prend en charge cette variable de recette uniquement dans les définitions du cycle de vie des composants.

Cette variable de recette contient les entrées suivantes :

- `component_dependency_name`— (Facultatif) Nom de la dépendance du composant à interroger. Omettez ce segment pour interroger le composant défini par cette recette. Vous ne pouvez spécifier que des dépendances directes.
- `json_pointer`— Le pointeur JSON vers la valeur de configuration à évaluer. Les pointeurs JSON commencent par une barre oblique/. Pour identifier une valeur dans une configuration de composants imbriqués, utilisez des barres obliques (/) pour séparer les clés de chaque niveau de la configuration. Vous pouvez utiliser un nombre comme clé pour spécifier un index dans une liste. Pour plus d'informations, consultez la [spécification du pointeur JSON](#).

AWS IoT Greengrass Core utilise des pointeurs JSON pour les recettes au format YAML.

Le pointeur JSON peut faire référence aux types de nœuds suivants :

- Un nœud de valeur. AWS IoT Greengrass Core remplace la variable de recette par la représentation sous forme de chaîne de la valeur. Les valeurs nulles sont converties `null` en chaîne.
- Un nœud d'objet. AWS IoT Greengrass Core remplace la variable de recette par la représentation sous forme de chaîne JSON sérialisée de cet objet.
- Pas de nœud. AWS IoT Greengrass Core ne remplace pas la variable de recette.

Par exemple, la variable de `{configuration:/Message}` recette récupère la valeur de la `Message` clé dans la configuration du composant. La variable de `{com.example.MyComponentDependency:configuration:/server/port}` recette récupère la valeur de `port` dans l'objet de `server` configuration d'une dépendance de composant.

`component_dependency_name:artifacts:path`

Le chemin racine des artefacts pour le composant défini par cette recette ou pour un composant dont dépend ce composant.

Lors de l'installation d'un composant, AWS IoT Greengrass copie les artefacts du composant dans le dossier exposé par cette variable. Vous pouvez utiliser cette variable pour identifier l'emplacement d'un script à exécuter dans le cycle de vie du composant, par exemple.

Le dossier situé sur ce chemin est en lecture seule. Pour modifier les fichiers d'artefacts, copiez-les vers un autre emplacement, tel que le répertoire de travail actuel (`$PWD` ou `.`). Modifiez ensuite les fichiers qui s'y trouvent.

Pour lire ou exécuter un artefact à partir d'une dépendance à un composant, l'artefact Read ou l'Execute autorisation de cet artefact doivent être. ALL Pour plus d'informations, consultez les [autorisations d'artefact](#) que vous définissez dans la recette du composant.

Cette variable de recette contient les entrées suivantes :

- `component_dependency_name`— (Facultatif) Nom de la dépendance du composant à interroger. Omettez ce segment pour interroger le composant défini par cette recette. Vous ne pouvez spécifier que des dépendances directes.

`component_dependency_name`:artifacts:decompressedPath

Le chemin racine des artefacts d'archive décompressés pour le composant défini par cette recette ou pour un composant dont dépend ce composant.

Lors de l'installation d'un composant, il AWS IoT Greengrass décompresse les artefacts d'archive du composant dans le dossier exposé par cette variable. Vous pouvez utiliser cette variable pour identifier l'emplacement d'un script à exécuter dans le cycle de vie du composant, par exemple.

Chaque artefact est décompressé dans un dossier situé dans le chemin décompressé, dans lequel le dossier porte le même nom que l'artefact, sauf son extension. Par exemple, un artefact ZIP nommé `models.zip` décompressé dans le `{artifacts:decompressedPath}/models` dossier.

Le dossier situé sur ce chemin est en lecture seule. Pour modifier les fichiers d'artefacts, copiez-les vers un autre emplacement, tel que le répertoire de travail actuel (`$PWD` ou `.`). Modifiez ensuite les fichiers qui s'y trouvent.

Pour lire ou exécuter un artefact à partir d'une dépendance à un composant, l'artefact Read ou l'Execute autorisation de cet artefact doivent être. ALL Pour plus d'informations, consultez les [autorisations d'artefact](#) que vous définissez dans la recette du composant.

Cette variable de recette contient les entrées suivantes :

- `component_dependency_name`— (Facultatif) Nom de la dépendance du composant à interroger. Omettez ce segment pour interroger le composant défini par cette recette. Vous ne pouvez spécifier que des dépendances directes.

`component_dependency_name`:work:path

Cette fonctionnalité est disponible pour les versions 2.0.4 et ultérieures du composant [Greengrass](#) nucleus.

Le chemin de travail pour le composant défini par cette recette ou pour un composant dont dépend ce composant. La valeur de cette variable de recette est équivalente à la sortie de la variable d'\$PWDenvironnement et de la commande [pwd](#) lorsqu'elle est exécutée depuis le contexte du composant.

Vous pouvez utiliser cette variable de recette pour partager des fichiers entre un composant et une dépendance.

Le dossier situé sur ce chemin est lisible et inscriptible par le composant défini par cette recette et par d'autres composants exécutés sous le même nom d'utilisateur et de même groupe.

Cette variable de recette contient les entrées suivantes :

- `component_dependency_name`— (Facultatif) Nom de la dépendance du composant à interroger. Omettez ce segment pour interroger le composant défini par cette recette. Vous ne pouvez spécifier que des dépendances directes.

`kernel:rootPath`

Le chemin racine AWS IoT Greengrass principal.

`iot:thingName`

Cette fonctionnalité est disponible pour les versions 2.3.0 et ultérieures du composant [Greengrass nucleus](#).

Le nom de l'appareil AWS IoT principal.

Exemples de recettes

Vous pouvez consulter les exemples de recettes suivants pour vous aider à créer des recettes pour vos composants.

AWS IoT Greengrass organise un index des composants de Greengrass, appelé Greengrass Software Catalog. Ce catalogue suit les composants de Greengrass développés par la communauté Greengrass. À partir de ce catalogue, vous pouvez télécharger, modifier et déployer des composants pour créer vos applications Greengrass. Pour de plus amples informations, veuillez consulter [Composantes communautaires](#).

Rubriques

- [Recette du composant Hello World](#)

- [Exemple de composant d'exécution Python](#)
- [Recette de composant qui spécifie plusieurs champs](#)

Recette du composant Hello World

La recette suivante décrit un composant Hello World qui exécute un script Python. Ce composant prend en charge toutes les plateformes et accepte un Message paramètre AWS IoT Greengrass qui est transmis en tant qu'argument au script Python. Voici la recette du composant Hello World du [didacticiel de démarrage](#).

JSON

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

```
}
```

YAML

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

Exemple de composant d'exécution Python

La recette suivante décrit un composant qui installe Python. Ce composant prend en charge les appareils Linux 64 bits.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.PythonRuntime",
  "ComponentDescription": "Installs Python 3.7",
  "ComponentPublisher": "Amazon",
  "ComponentVersion": "3.7.0",
  "Manifests": [
    {
      "Platform": {
        "os": "linux",
```

```
    "architecture": "amd64"
  },
  "Lifecycle": {
    "install": "apt-get update\napt-get install python3.7"
  }
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.PythonRuntime
ComponentDescription: Installs Python 3.7
ComponentPublisher: Amazon
ComponentVersion: '3.7.0'
Manifests:
- Platform:
  os: linux
  architecture: amd64
Lifecycle:
  install: |
    apt-get update
    apt-get install python3.7
```

Recette de composant qui spécifie plusieurs champs

La recette de composant suivante utilise plusieurs champs de recette.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.FooService",
  "ComponentDescription": "Complete recipe for AWS IoT Greengrass components",
  "ComponentPublisher": "Amazon",
  "ComponentVersion": "1.0.0",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "TestParam": "TestValue"
    }
  }
}
```



```
},
"ComponentDependencies": {
  "BarService": {
    "VersionRequirement": "^1.1.0",
    "DependencyType": "SOFT"
  },
  "BazService": {
    "VersionRequirement": "^2.0.0"
  }
},
"Manifests": [
  {
    "Platform": {
      "os": "linux",
      "architecture": "amd64"
    },
    "Lifecycle": {
      "install": {
        "Skipif": "onpath git",
        "Script": "sudo apt-get install git"
      },
      "Setenv": {
        "environment_variable1": "variable_value1",
        "environment_variable2": "variable_value2"
      }
    },
    "Artifacts": [
      {
        "Uri": "s3://amzn-s3-demo-bucket/hello_world.zip",
        "Unarchive": "ZIP"
      },
      {
        "Uri": "s3://amzn-s3-demo-bucket/hello_world_linux.py"
      }
    ]
  },
  {
    "Lifecycle": {
      "install": {
        "Skipif": "onpath git",
        "Script": "sudo apt-get install git",
        "RequiresPrivilege": "true"
      }
    }
  }
},
```

```
    "Artifacts": [  
      {  
        "Uri": "s3://amzn-s3-demo-bucket/hello_world.py"  
      }  
    ]  
  }  
]
```

YAML

```
---  
RecipeFormatVersion: '2020-01-25'  
ComponentName: com.example.FooService  
ComponentDescription: Complete recipe for AWS IoT Greengrass components  
ComponentPublisher: Amazon  
ComponentVersion: 1.0.0  
ComponentConfiguration:  
  DefaultConfiguration:  
    TestParam: TestValue  
ComponentDependencies:  
  BarService:  
    VersionRequirement: ^1.1.0  
    DependencyType: SOFT  
  BazService:  
    VersionRequirement: ^2.0.0  
Manifests:  
  - Platform:  
    os: linux  
    architecture: amd64  
  Lifecycle:  
    install:  
      SkipIf: onpath git  
      Script: sudo apt-get install git  
    SetEnv:  
      environment_variable1: variable_value1  
      environment_variable2: variable_value2  
  Artifacts:  
    - Uri: 's3://amzn-s3-demo-bucket/hello_world.zip'  
      Unarchive: ZIP  
    - Uri: 's3://amzn-s3-demo-bucket/hello_world_linux.py'  
  - Lifecycle:  
    install:
```

```
SkipIf: onpath git
Script: sudo apt-get install git
RequiresPrivilege: 'true'
Artifacts:
- Uri: 's3://amzn-s3-demo-bucket/hello_world.py'
```

Référence à la variable d'environnement du composant

Le logiciel AWS IoT Greengrass Core définit les variables d'environnement lorsqu'il exécute des scripts de cycle de vie pour les composants. Vous pouvez obtenir ces variables d'environnement dans vos composants pour obtenir le nom de l'objet et la Région AWS version du noyau de Greengrass. Le logiciel définit également les variables d'environnement dont votre composant a besoin pour utiliser [le SDK de communication interprocessus](#) et pour [interagir avec AWS les services](#).

Vous pouvez également définir des variables d'environnement personnalisées pour les scripts de cycle de vie de vos composants. Pour plus d'informations, consultez [Setenv](#).

Le logiciel AWS IoT Greengrass Core définit les variables d'environnement suivantes :

AWS_IOT_THING_NAME

Le nom de l' AWS IoT objet qui représente cet appareil central de Greengrass.

AWS_REGION

L' Région AWS endroit où fonctionne cet appareil principal Greengrass.

AWS SDKs Utilisez cette variable d'environnement pour identifier la région par défaut à utiliser. Cette variable est équivalente àAWS_DEFAULT_REGION.

AWS_DEFAULT_REGION

L' Région AWS endroit où fonctionne cet appareil principal Greengrass.

AWS CLI Utilise cette variable d'environnement pour identifier la région par défaut à utiliser. Cette variable est équivalente àAWS_REGION.

GGC_VERSION

Version du [composant Greengrass noyau](#) qui s'exécute sur cet appareil principal de Greengrass.

GG_ROOT_CA_PATH

Cette fonctionnalité est disponible pour les versions 2.5.5 et ultérieures du composant [Greengrass](#) nucleus.

Le chemin d'accès au certificat de l'autorité de certification (CA) racine utilisé par le noyau Greengrass.

AWS_GG_NUCLEUS_DOMAIN_SOCKET_FILEPATH_FOR_COMPONENT

Le chemin d'accès au socket IPC que les composants utilisent pour communiquer avec le logiciel AWS IoT Greengrass Core. Pour de plus amples informations, veuillez consulter [Utilisez le Kit SDK des appareils AWS IoT pour communiquer avec le noyau de Greengrass, les autres composants et AWS IoT Core](#).

SVCUID

Le jeton secret que les composants utilisent pour se connecter au socket IPC et communiquer avec le logiciel AWS IoT Greengrass Core. Pour de plus amples informations, veuillez consulter [Utilisez le Kit SDK des appareils AWS IoT pour communiquer avec le noyau de Greengrass, les autres composants et AWS IoT Core](#).

AWS_CONTAINER_AUTHORIZATION_TOKEN

Le jeton secret que les composants utilisent pour récupérer les informations d'identification du [composant du service d'échange de jetons](#).

AWS_CONTAINER_CREDENTIALS_FULL_URI

L'URI demandé par les composants pour récupérer les informations d'identification auprès du [composant du service d'échange de jetons](#).

Déployer AWS IoT Greengrass des composants sur des appareils

Vous pouvez l'utiliser AWS IoT Greengrass pour déployer des composants sur des appareils ou des groupes d'appareils. Vous utilisez les déploiements pour définir les composants et les configurations qui sont envoyés aux appareils. AWS IoT Greengrass se déploie sur des cibles, des AWS IoT objets ou des groupes d'objets qui représentent les principaux appareils de Greengrass. AWS IoT Greengrass utilise des [AWS IoT Core tâches](#) pour les déployer sur vos appareils principaux. Vous pouvez configurer le mode de déploiement de la tâche sur vos appareils.

Déploiements d'appareils principaux

Chaque périphérique principal exécute les composants des déploiements pour cet appareil. Un nouveau déploiement vers la même cible remplace le déploiement précédent vers la cible. Lorsque vous créez un déploiement, vous définissez les composants et les configurations à appliquer au logiciel existant du périphérique principal.

Lorsque vous révisiez un déploiement pour une cible, vous remplacez les composants de la version précédente par ceux de la nouvelle révision. Par exemple, vous déployez les [Directeur des services secrets](#) composants [Gestionnaire de journaux](#) et dans le groupe d'objets `TestGroup`. Ensuite, vous créez un autre déploiement pour `TestGroup` lequel seul le composant du gestionnaire de secrets est spécifié. Par conséquent, les appareils principaux de ce groupe n'exécutent plus le gestionnaire de journaux.

Résolution des dépendances entre plateformes

Lorsqu'un périphérique principal reçoit un déploiement, il vérifie que les composants sont compatibles avec le périphérique principal. Par exemple, si vous déployez [Firehose](#) le sur une cible Windows, le déploiement échouera.

Résolution de dépendance des composants

Lors du déploiement d'un composant, le périphérique principal vérifie la compatibilité des dépendances de tous les composants et des exigences de version au sein d'un groupe d'objets. Cette vérification garantit que les contraintes de version sont satisfaites pour tous les composants et leurs dépendances avant de procéder au déploiement.

Le processus de résolution des dépendances commence par l'identification des composants cibles dont les recettes ne contiennent aucune dépendance. Le système construit ensuite un arbre de dépendances à l'aide de la méthode BFS (BFS) qui explore systématiquement chaque nœud cible et trouve d'abord ses dépendances avant de passer au nœud suivant. Chaque nœud inclut le composant cible comme clé et les exigences de version comme valeur.

Les exigences de version combinent trois ensembles de contraintes :

- Les exigences de version déjà définies dans le groupe d'objets existant.
- Version du composant requise par le déploiement. Vous devez sélectionner une version de composant lorsque vous effectuez ou mettez à jour un déploiement.

- Toutes les contraintes de version des composants définies dans la section de dépendance de la recette.

Résoudre les dépendances entre les composants

Lors d'un déploiement, le noyau Greengrass tente d'abord de trouver le candidat local actuellement en cours d'exécution sur l'appareil qui répond aux exigences. Si le composant en cours d'exécution répond aux exigences, le noyau obtient le chemin de recette stocké dans le dossier de recettes et trouve la dernière version locale dans le magasin local.

Pour les AWS Cloud déploiements, le noyau appellera ensuite l'[ResolveComponentCandidates API](#). Cette API démarrera avec la dernière version disponible et vérifiera si elle répond aux dépendances et aux exigences. Lorsque le noyau reçoit la réponse de l'API, il sélectionne la dernière version. Si aucune version répondant aux exigences n'est trouvée dans AWS Cloud le fichier, le déploiement échoue. Si l'appareil est hors ligne, il revient au candidat local d'origine trouvé. Si aucun candidat local répondant aux exigences n'est trouvé, le déploiement échoue.

Pour les déploiements locaux, le noyau utilise exclusivement des candidats locaux s'ils existent et s'ils répondent aux exigences sans avoir à AWS Cloud négocier avec eux. En l'absence d'un tel candidat, le déploiement échoue.

Note

Toutes les recettes résolues sont stockées localement pour référence future.

Pour plus d'informations, consultez la section sur la [résolution des dépendances](#) dans GitHub.

Si le noyau Greengrass parvient à résoudre tous les composants, le journal du noyau contiendra la ligne suivante.

```
resolve-all-group-dependencies-finish. Finish resolving all groups dependencies.
```

Exemple

Voici un exemple de la manière dont le noyau répondra aux exigences en matière de composants.

- Vous déployez ComponentA qui dépend des versions 1.0.0-1.9.0 de ComponentC.
- Vous déployez également le composant B qui dépend des versions 1.4.0-1.9.5 de ComponentC.

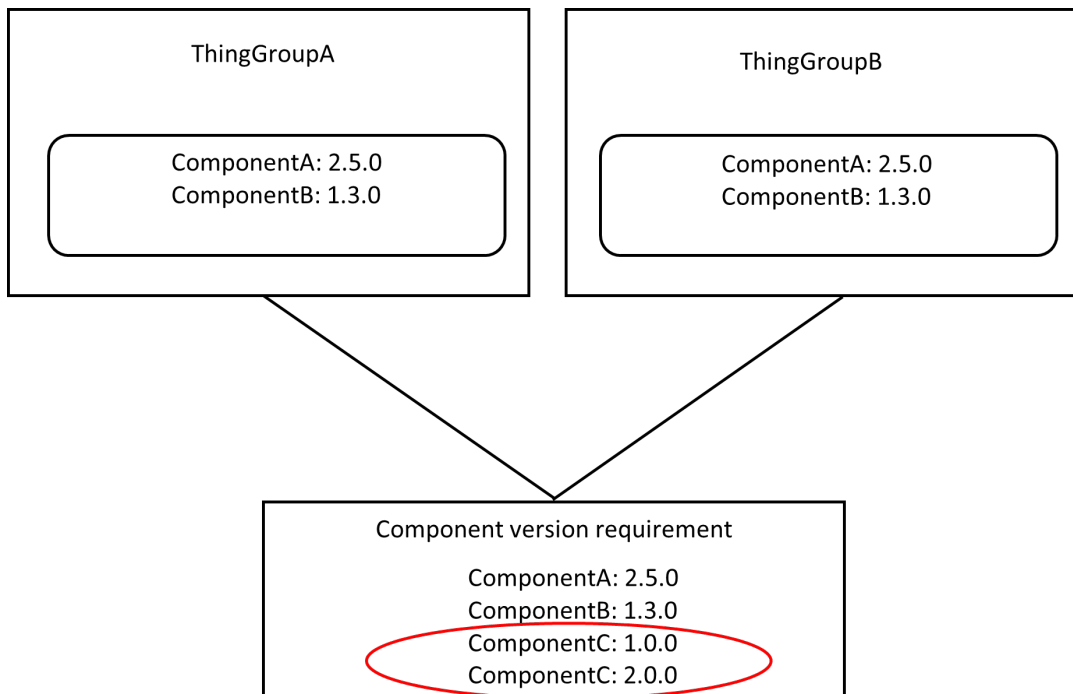
Avec la résolution des dépendances des composants, le noyau déploiera la dernière version de la version ComponentC pour répondre aux exigences des composants A et ComponentB. Cette dernière version de ComponentC est la version 1.9.0.

Défaillances courantes de résolution des dépendances entre les composants

La résolution des dépendances des composants peut échouer pour deux raisons principales : un conflit entre les exigences de la version cible ou le conflit des exigences de dépendance des composants.

Scénario 1 : conflit entre les exigences de la version cible

- Un objet existe dans un groupe d'objets et vous souhaitez également l'ajouter à un nouveau groupe d'objets. Le déploiement échouera si le nouveau groupe d'objets nécessite une version d'objet différente.
- Un déploiement peut également échouer si un objet appartient à un groupe d'objets et souhaite mettre à jour la version du composant par le biais d'un déploiement d'objets.



Exemple de journal des défaillances :

```
2025-04-11T06:16:03.315Z [ERROR] (pool-3-thread-27)
com.aws.greengrass.componentmanager.ComponentManager: Failed to negotiate version
```

```

with cloud and no local version to fall back to. {componentName=ComponentC,
versionRequirement={thing/ABC==2.0.0, thinggroup/ThingGroupA==1.0.0}}
2025-04-11T06:16:03.316Z [ERROR] (pool-3-thread-26)
com.aws.greengrass.deployment.DeploymentService: Error occurred while
processing deployment. {deploymentId=fbac24de-4ef9-44b0-a685-fdc63b0f02b8,
serviceName=DeploymentService, currentState=RUNNING}
java.util.concurrent.ExecutionException:
com.aws.greengrass.componentmanager.exceptions.NoAvailableComponentVersionException:
No local or cloud component version satisfies the requirements Check whether the
version constraints conflict and that the component exists in your AWS account with
a version that matches the version constraints. If the version constraints conflict,
revise deployments to resolve the conflict. Component ComponentC version constraints:
thing/ABC requires =2.0.0, thinggroup/ThingGroupA requires =1.0.0.
    at java.base/java.util.concurrent.FutureTask.report(FutureTask.java:122)
    at java.base/java.util.concurrent.FutureTask.get(FutureTask.java:191)
    at
com.aws.greengrass.deployment.DefaultDeploymentTask.call(DefaultDeploymentTask.java:127)
    at
com.aws.greengrass.deployment.DefaultDeploymentTask.call(DefaultDeploymentTask.java:50)
    at java.base/java.util.concurrent.FutureTask.run(FutureTask.java:264)
    at java.base/
java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1128)
    at java.base/java.util.concurrent.ThreadPoolExecutor
$Worker.run(ThreadPoolExecutor.java:628)
    at java.base/java.lang.Thread.run(Thread.java:829)
Caused by:
com.aws.greengrass.componentmanager.exceptions.NoAvailableComponentVersionException:
No local or cloud component version satisfies the requirements Check whether the
version constraints conflict and that the component exists in your AWS account with
a version that matches the version constraints. If the version constraints conflict,
revise deployments to resolve the conflict. Component ComponentC version constraints:
thing/ABC requires =2.0.0, thinggroup/ThingGroupA requires =1.0.0.
    at
com.aws.greengrass.componentmanager.ComponentManager.negotiateVersionWithCloud(ComponentManager
    at
com.aws.greengrass.componentmanager.ComponentManager.resolveComponentVersion(ComponentManager
    at com.aws.greengrass.componentmanager.DependencyResolver.lambda
$resolveDependencies$0(DependencyResolver.java:134)
    at
com.aws.greengrass.componentmanager.DependencyResolver.resolveComponentDependencies(Dependency
    at
com.aws.greengrass.componentmanager.DependencyResolver.resolveDependencies(DependencyResolver
    at com.aws.greengrass.deployment.DefaultDeploymentTask.lambda$call
$2(DefaultDeploymentTask.java:125)

```


... 4 more

Les journaux indiquent une erreur de conflit de version car le noyau ne trouve pas de version de composant répondant simultanément à deux exigences contradictoires.

Comment le résoudre

- Si vous souhaitez conserver le composant dans chaque groupe d'objets, sélectionnez la même version de ce composant dans chaque groupe d'objets.
- Sélectionnez une version de composant qui répond aux exigences de déploiement.
- Si vous souhaitez utiliser une version de composant qui ne répond pas aux exigences des deux groupes d'objets, sélectionnez la version du composant qui répond aux exigences de version du groupe d'objets et utilisez ce composant uniquement dans ce groupe d'objets.

Scénario 2 : conflit entre les exigences de version et de dépendance des composants

Si un composant est une dépendance de différents composants et que les composants nécessitent différentes versions ou différentes plages de versions de ce composant, il est possible qu'aucune version ne soit disponible pour satisfaire toutes les exigences de version. Dans ce scénario, le déploiement échouera.

Exemple

Déploiement de ComponentA (v2.5.0), ComponentB (v1.3.0) et ComponentC (v1.0.0)

- ComponentA nécessite une version de ComponentB supérieure ou égale à 1.0.0.

```
---
...
ComponentName: ComponentA
ComponentVersion: "2.5.0"
ComponentDependencies:
  ComponentB:
    VersionRequirement: ">=1.0.0"
    DependencyType: "HARD"
...
```

- ComponentC nécessite une version de ComponentA <2.0.0.

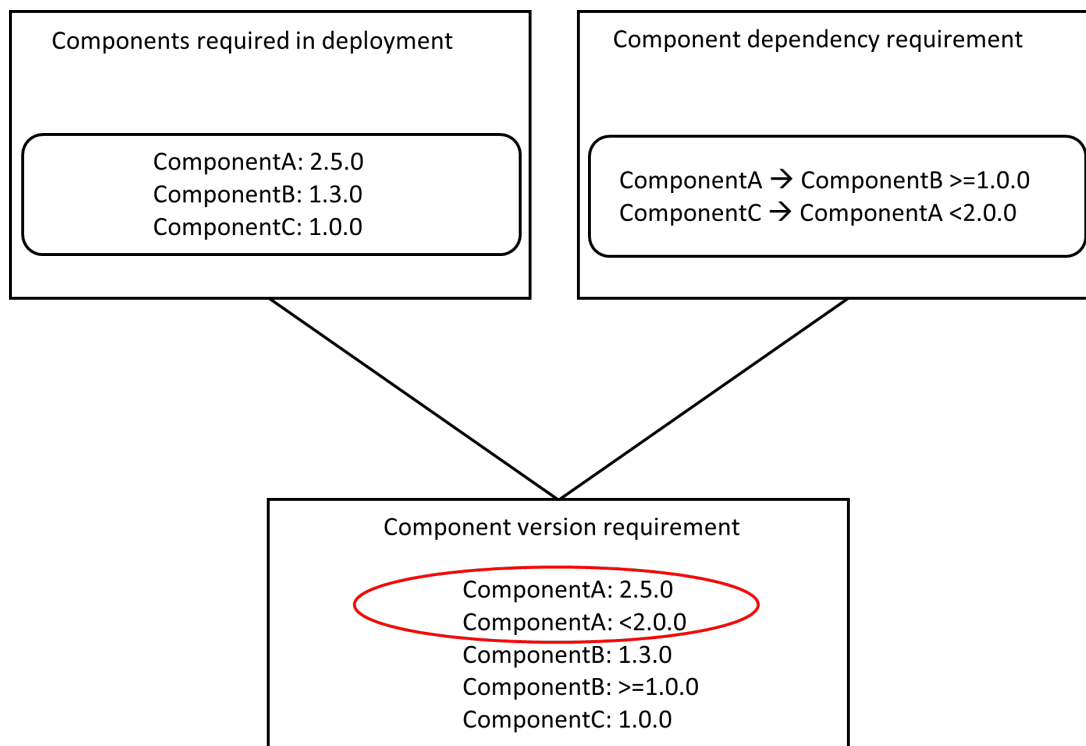
```
---
...
```

```
ComponentName: ComponentC
ComponentVersion: "1.0.0"
ComponentDependencies:
  ComponentA:
    VersionRequirement: "<2.0.0"
    DependencyType: "HARD"
...
```

Il existe un conflit de version entre deux exigences pour ComponentA :

- ComponentA nécessite la version 2.5.0 dans ce déploiement
- ComponentC nécessite des versions de ComponentA inférieures à 2.0.0

Ces deux exigences se contredisent, ce qui empêche le noyau de trouver une version de ComponentA qui réponde aux deux exigences. Par conséquent, la résolution des dépendances échoue.



Exemple de journal des défaillances :

```
2025-04-11T06:07:18.291Z [ERROR] (pool-3-thread-25)
com.aws.greengrass.componentmanager.ComponentManager: Failed to negotiate version
```

```

with cloud and no local version to fall back to. {componentName=ComponentA,
versionRequirement={ComponentC=<2.0.0, thinggroup/ThingGroupA==2.5.0}}
2025-04-11T06:07:18.292Z [ERROR] (pool-3-thread-24)
com.aws.greengrass.deployment.DeploymentService: Error occurred while
processing deployment. {deploymentId=2ffac4df-1ac9-405c-8c11-28494a1b4382,
serviceName=DeploymentService, currentState=RUNNING}
java.util.concurrent.ExecutionException:
com.aws.greengrass.componentmanager.exceptions.NoAvailableComponentVersionException:
No local or cloud component version satisfies the requirements Check whether the
version constraints conflict and that the component exists in your AWS account with
a version that matches the version constraints. If the version constraints conflict,
revise deployments to resolve the conflict. Component ComponentA version constraints:
ComponentC requires <2.0.0, thinggroup/ThingGroupA requires =2.5.0.
    at java.base/java.util.concurrent.FutureTask.report(FutureTask.java:122)
    at java.base/java.util.concurrent.FutureTask.get(FutureTask.java:191)
    at
com.aws.greengrass.deployment.DefaultDeploymentTask.call(DefaultDeploymentTask.java:127)
    at
com.aws.greengrass.deployment.DefaultDeploymentTask.call(DefaultDeploymentTask.java:50)
    at java.base/java.util.concurrent.FutureTask.run(FutureTask.java:264)
    at java.base/
java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1128)
    at java.base/java.util.concurrent.ThreadPoolExecutor
$Worker.run(ThreadPoolExecutor.java:628)
    at java.base/java.lang.Thread.run(Thread.java:829)
Caused by:
com.aws.greengrass.componentmanager.exceptions.NoAvailableComponentVersionException:
No local or cloud component version satisfies the requirements Check whether the
version constraints conflict and that the component exists in your AWS account with
a version that matches the version constraints. If the version constraints conflict,
revise deployments to resolve the conflict. Component ComponentA version constraints:
ComponentC requires <2.0.0, thinggroup/ThingGroupA requires =2.5.0.
    at
com.aws.greengrass.componentmanager.ComponentManager.negotiateVersionWithCloud(ComponentManager.
    at
com.aws.greengrass.componentmanager.ComponentManager.resolveComponentVersion(ComponentManager.
    at com.aws.greengrass.componentmanager.DependencyResolver.lambda
$resolveDependencies$0(DependencyResolver.java:134)
    at
com.aws.greengrass.componentmanager.DependencyResolver.resolveComponentDependencies(Dependency
    at
com.aws.greengrass.componentmanager.DependencyResolver.resolveDependencies(DependencyResolver.
    at com.aws.greengrass.deployment.DefaultDeploymentTask.lambda$call
$2(DefaultDeploymentTask.java:125)

```

... 4 more

Les journaux indiquent que le noyau ne trouve pas de version de ComponentA qui réponde aux exigences suivantes.

- La configuration requise pour ComponentA doit être exactement la version 2.5.0 (à partir de ThingGroup A).
- L'obligation de travailler avec les versions de ComponentC inférieures à 2.0.0.

Comment le résoudre

- Si vous souhaitez conserver le composant dans chaque groupe d'objets, sélectionnez la même version de ce composant dans chaque groupe d'objets.
- Sélectionnez une version de composant qui répond aux exigences de déploiement.
- Si vous souhaitez utiliser une version de composant qui ne répond pas aux exigences des deux groupes d'objets, sélectionnez la version du composant qui répond aux exigences de version du groupe d'objets et utilisez ce composant uniquement dans ce groupe d'objets.

Tip

Si cette erreur s'affiche sur l'un des composants AWS fournis, vous pouvez la résoudre en mettant à jour les composants en conflit avec la dernière version.

Supprimer un appareil d'un groupe d'objets

Lorsque vous supprimez un périphérique principal d'un groupe d'objets, le comportement de déploiement du composant dépend de la version du [noyau Greengrass](#) que le périphérique principal exécute.

2.5.1 and later

Lorsque vous supprimez un appareil principal d'un groupe d'objets, le comportement dépend de `greengrass:ListThingGroupsForCoreDevice` autorisation accordée ou non par la AWS IoT politique. Pour plus d'informations sur cette autorisation et AWS IoT les politiques relatives aux appareils principaux, consultez [Authentification et autorisation de l'appareil pour AWS IoT Greengrass](#).

- Si la AWS IoT politique accorde cette autorisation

Lorsque vous supprimez un appareil principal d'un groupe d'objets, AWS IoT Greengrass les composants du groupe d'objets seront supprimés lors du prochain déploiement sur l'appareil. Si un composant de l'appareil est inclus dans le déploiement suivant, ce composant n'est pas supprimé de l'appareil.

- Si la AWS IoT politique n'accorde pas cette autorisation

Lorsque vous supprimez un appareil principal d'un groupe d'objets, les composants de ce groupe d'objets AWS IoT Greengrass ne sont pas supprimés de l'appareil.

Pour supprimer un composant d'un appareil, utilisez la commande deployment [create](#) de la CLI Greengrass. Spécifiez le composant à supprimer avec l'- -removeargument, et spécifiez le groupe d'objets avec l'- -groupIdargument.

2.5.0

Lorsque vous supprimez un appareil principal d'un groupe d'objets, AWS IoT Greengrass les composants du groupe d'objets seront supprimés lors du prochain déploiement sur l'appareil. Si un composant de l'appareil est inclus dans le déploiement suivant, ce composant n'est pas supprimé de l'appareil.

Ce comportement nécessite que la AWS IoT politique du périphérique principal accorde l'greengrass:ListThingGroupsForCoreDeviceautorisation. Si un appareil principal ne dispose pas de cette autorisation, il ne parvient pas à appliquer les déploiements. Pour de plus amples informations, veuillez consulter [Authentification et autorisation de l'appareil pour AWS IoT Greengrass](#).

2.0.x - 2.4.x

Lorsque vous supprimez un appareil principal d'un groupe d'objets, les composants de ce groupe d'objets AWS IoT Greengrass ne sont pas supprimés de l'appareil.

Pour supprimer un composant d'un appareil, utilisez la commande deployment [create](#) de la CLI Greengrass. Spécifiez le composant à supprimer avec l'- -removeargument, et spécifiez le groupe d'objets avec l'- -groupIdargument.

Déploiements

Les déploiements sont continus. Lorsque vous créez un déploiement, AWS IoT Greengrass déployez-le sur les appareils cibles qui sont en ligne. Si une machine cible n'est pas en ligne, elle recevra le déploiement lors de sa prochaine connexion AWS IoT Greengrass. Lorsque vous ajoutez un appareil principal à un groupe d'objets cible, AWS IoT Greengrass envoie à l'appareil le dernier déploiement pour ce groupe d'objets.

Avant qu'un périphérique principal ne déploie un composant, il notifie par défaut chaque composant du périphérique. Les composants Greengrass peuvent répondre à la notification pour différer le déploiement. Vous souhaitez peut-être différer le déploiement si le niveau de batterie de l'appareil est faible ou s'il exécute un processus qui ne peut pas être interrompu. Pour de plus amples informations, veuillez consulter [Tutoriel : Développement d'un composant Greengrass qui reporte les mises à jour des composants](#). Lorsque vous créez un déploiement, vous pouvez le configurer pour qu'il soit déployé sans avertir les composants.

Chaque objet ou groupe d'objets cible peut être déployé un par un. Cela signifie que lorsque vous créez un déploiement pour une cible, la version précédente du déploiement de cette cible AWS IoT Greengrass ne sera plus déployée.

Options de déploiement

Les déploiements proposent plusieurs options qui vous permettent de contrôler quels appareils reçoivent une mise à jour et comment la mise à jour est déployée. Lorsque vous créez un déploiement, vous pouvez configurer les options suivantes :

- AWS IoT Greengrass composants

Définissez les composants à installer et à exécuter sur les équipements cibles. AWS IoT Greengrass les composants sont des modules logiciels que vous déployez et exécutez sur les appareils principaux de Greengrass. Les appareils reçoivent des composants uniquement s'ils sont compatibles avec la plate-forme de l'appareil. Cela vous permet de déployer sur des groupes d'appareils même si les appareils cibles fonctionnent sur plusieurs plateformes. Si un composant ne prend pas en charge la plate-forme de l'appareil, il n'est pas déployé sur l'appareil.

Vous pouvez déployer des composants personnalisés et des composants AWS fournis sur vos appareils. Lorsque vous déployez un composant, AWS IoT Greengrass identifie les dépendances des composants et déployez-les également. Pour plus d'informations, consultez [Développer des AWS IoT Greengrass composants](#) et [AWS-composants fournis](#).

Vous définissez la version et la mise à jour de configuration à déployer pour chaque composant. La mise à jour de configuration indique comment modifier la configuration existante du composant sur le périphérique principal, ou la configuration par défaut du composant si le composant n'existe pas sur le périphérique principal. Vous pouvez spécifier les valeurs de configuration à rétablir aux valeurs par défaut et les nouvelles valeurs de configuration à fusionner sur le périphérique principal. Lorsqu'un périphérique principal reçoit des déploiements pour différentes cibles et que chaque déploiement spécifie des versions de composants compatibles, le périphérique principal applique les mises à jour de configuration dans l'ordre en fonction de l'horodatage du moment où vous créez le déploiement. Pour de plus amples informations, veuillez consulter [Mettre à jour les configurations des composants](#).

 Important

Lorsque vous déployez un composant, AWS IoT Greengrass installe les dernières versions prises en charge de toutes les dépendances de ce composant. De ce fait, les nouvelles versions de correctif des composants publics AWS fournis peuvent être automatiquement déployées sur vos appareils principaux si vous ajoutez de nouveaux appareils à un groupe d'objets ou si vous mettez à jour le déploiement qui cible ces appareils. Certaines mises à jour automatiques, telles que la mise à jour du noyau, peuvent provoquer le redémarrage inattendu de vos appareils.

Pour éviter les mises à jour involontaires d'un composant en cours d'exécution sur votre appareil, nous vous recommandons d'inclure directement votre version préférée de ce composant lorsque vous [créez un déploiement](#). Pour plus d'informations sur le comportement de mise à jour du logiciel AWS IoT Greengrass Core, consultez [Mettre à jour le logiciel AWS IoT Greengrass principal \(OTA\)](#).

- Politiques de déploiement

Définissez à quel moment il est possible de déployer une configuration en toute sécurité et ce qu'il convient de faire en cas d'échec du déploiement. Vous pouvez indiquer s'il faut attendre ou non que les composants signalent qu'ils peuvent être mis à jour. Vous pouvez également spécifier s'il convient ou non de rétablir la configuration précédente des appareils s'ils appliquent un déploiement qui échoue.

- Arrêter la configuration

Définissez quand et comment arrêter un déploiement. Le déploiement s'arrête et échoue si les critères que vous définissez sont remplis. Par exemple, vous pouvez configurer un déploiement

pour qu'il s'arrête si un pourcentage d'appareils ne parvient pas à appliquer ce déploiement après qu'un nombre minimum d'appareils l'aient reçu.

- Configuration du déploiement

Définissez la vitesse à laquelle un déploiement est déployé sur les équipements cibles. Vous pouvez configurer une augmentation exponentielle du taux avec des limites de taux minimum et maximum.

- Configuration du délai d'expiration

Définissez le délai maximal dont dispose chaque appareil pour effectuer un déploiement. Si un appareil dépasse la durée que vous spécifiez, il ne parvient pas à appliquer le déploiement.

 Important

Les composants personnalisés peuvent définir des artefacts dans des compartiments S3. Lorsque le logiciel AWS IoT Greengrass Core déploie un composant, il télécharge les artefacts du composant depuis le AWS Cloud. Les rôles principaux des appareils n'autorisent pas l'accès aux compartiments S3 par défaut. Pour déployer des composants personnalisés qui définissent des artefacts dans un compartiment S3, le rôle principal du périphérique doit accorder des autorisations pour télécharger des artefacts depuis ce compartiment. Pour de plus amples informations, veuillez consulter [Autoriser l'accès aux compartiments S3 pour les artefacts des composants](#).

Rubriques

- [Créer des déploiements](#)
- [Création de sous-déploiements](#)
- [Réviser les déploiements](#)
- [Annuler les déploiements](#)
- [Vérifier l'état du déploiement](#)

Créer des déploiements

Vous pouvez créer un déploiement qui cible un objet ou un groupe d'objets.

Lorsque vous créez un déploiement, vous configurez les composants logiciels à déployer et la manière dont la tâche de déploiement est déployée sur les équipements cibles. Vous pouvez définir le déploiement dans le fichier JSON que vous fournissez au AWS CLI.

La cible de déploiement détermine les appareils sur lesquels vous souhaitez exécuter vos composants. Pour effectuer un déploiement sur un appareil principal, spécifiez un élément. Pour effectuer un déploiement sur plusieurs appareils principaux, spécifiez un groupe d'objets qui inclut ces appareils. Pour plus d'informations sur la configuration des groupes d'objets, consultez les sections [Groupes d'objets statiques](#) et [Groupes d'objets dynamiques](#) dans le Guide du AWS IoT développeur.

Suivez les étapes décrites dans cette section pour créer un déploiement vers une cible. Pour plus d'informations sur la façon de mettre à jour les composants logiciels sur une cible déployée, consultez [Réviser les déploiements](#).

 Warning

L'[CreateDeployment](#) opération peut désinstaller des composants des périphériques principaux. Si un composant est présent dans le déploiement précédent et non dans le nouveau déploiement, le périphérique principal désinstalle ce composant. Pour éviter de désinstaller des composants, utilisez d'abord l'[ListDeployments](#) opération pour vérifier si la cible du déploiement possède déjà un déploiement existant. Utilisez ensuite l'[GetDeployment](#) opération pour démarrer à partir de ce déploiement existant lorsque vous créez un nouveau déploiement.

Pour créer un déploiement (AWS CLI)

1. Créez un fichier appelé `deployment.json`, puis copiez l'objet JSON suivant dans le fichier. *targetArn* Remplacez-le par l'ARN de l' AWS IoT objet ou du groupe d'objets à cibler pour le déploiement. L'objet et le groupe ARNs d'objets ont le format suivant :
- Objet : `arn:aws:iot:region:account-id:thing/thingName`
 - Groupe d'objets : `arn:aws:iot:region:account-id:thinggroup/thingGroupName`

```
{
  "targetArn": "targetArn"
}
```

2. Vérifiez si la cible de déploiement comporte un déploiement existant que vous souhaitez réviser. Procédez comme suit :

- a. Exécutez la commande suivante pour répertorier les déploiements pour la cible de déploiement. Remplacez *targetArn* par l'ARN de l' AWS IoT objet ou du groupe d'objets cible.

```
aws greengrassv2 list-deployments --target-arn targetArn
```

La réponse contient une liste du dernier déploiement pour la cible. Si la réponse est vide, la cible n'a pas de déploiement existant et vous pouvez passer à [Step 3](#). Sinon, copiez le contenu `deploymentId` de la réponse pour l'utiliser à l'étape suivante.

 Note

Vous pouvez également modifier un déploiement autre que la dernière révision pour la cible. Spécifiez l' `--history-filter ALL` argument pour répertorier tous les déploiements pour la cible. Copiez ensuite l'ID du déploiement que vous souhaitez modifier.

- b. Exécutez la commande suivante pour obtenir les détails du déploiement. Ces détails incluent les métadonnées, les composants et la configuration des tâches. *deploymentId* Remplacez-le par l'ID de l'étape précédente.

```
aws greengrassv2 get-deployment --deployment-id deploymentId
```

La réponse contient les détails du déploiement.

- c. Copiez l'une des paires clé-valeur suivantes de la réponse de la commande précédente dans `deployment.json`. Vous pouvez modifier ces valeurs pour le nouveau déploiement.
- `deploymentName`— Le nom du déploiement.
 - `components`— Les composants du déploiement. Pour désinstaller un composant, supprimez-le de cet objet.
 - `deploymentPolicies`— Les politiques du déploiement.
 - `iotJobConfiguration`— La configuration des tâches du déploiement.
 - `tags`— Les tags du déploiement.

3. (Facultatif) Définissez un nom pour le déploiement. Remplacez *deploymentName* par le nom du déploiement.

```
{
  "targetArn": "targetArn",
  "deploymentName": "deploymentName"
}
```

4. Ajoutez chaque composant pour déployer les équipements cibles. Pour ce faire, ajoutez des paires clé-valeur à l'objet `components`, la clé étant le nom du composant et la valeur étant un objet contenant les détails de ce composant. Spécifiez les détails suivants pour chaque composant que vous ajoutez :
 - `version`— Version du composant à déployer.
 - `configurationUpdate`— La [mise à jour de configuration](#) à déployer. La mise à jour est une opération de correctif qui modifie la configuration existante du composant sur chaque équipement cible, ou la configuration par défaut du composant s'il n'existe pas sur le périphérique cible. Vous pouvez spécifier les mises à jour de configuration suivantes :
 - Reset updates (`reset`) — (Facultatif) Une liste de pointeurs JSON qui définissent les valeurs de configuration à rétablir à leurs valeurs par défaut sur l'appareil cible. Le logiciel AWS IoT Greengrass Core applique les mises à jour de réinitialisation avant d'appliquer les mises à jour de fusion. Pour de plus amples informations, veuillez consulter [Réinitialiser les mises à jour](#).
 - Merge updates (`merge`) — (Facultatif) Document JSON qui définit les valeurs de configuration à fusionner sur le périphérique cible. Vous devez sérialiser le document JSON sous forme de chaîne. Pour de plus amples informations, veuillez consulter [Fusion des mises à jour](#).
 - `runWith`— (Facultatif) Les options de processus système que le logiciel AWS IoT Greengrass Core utilise pour exécuter les processus de ce composant sur le périphérique principal. Si vous omettez un paramètre dans l'objet `runWith`, le logiciel AWS IoT Greengrass Core utilise les valeurs par défaut que vous configurez sur le composant [Greengrass nucleus](#).

Vous pouvez définir l'une des options suivantes :

- `posixUser`— L'utilisateur du système POSIX et, éventuellement, le groupe à utiliser pour exécuter ce composant sur les périphériques principaux de Linux. L'utilisateur et le groupe, s'ils sont spécifiés, doivent exister sur chaque périphérique principal Linux.

Spécifiez l'utilisateur et le groupe en les séparant par deux points (:) au format suivant : `user:group`. Le groupe est facultatif. Si vous ne spécifiez aucun groupe, le logiciel AWS IoT Greengrass Core utilise le groupe principal pour l'utilisateur. Pour de plus amples informations, veuillez consulter [Configurer l'utilisateur qui exécute les composants](#).

- `windowsUser`— L'utilisateur Windows à utiliser pour exécuter ce composant sur les appareils principaux de Windows. L'utilisateur doit exister sur chaque appareil principal de Windows, et son nom et son mot de passe doivent être stockés dans l'instance Credentials Manager du LocalSystem compte. Pour de plus amples informations, veuillez consulter [Configurer l'utilisateur qui exécute les composants](#).

Cette fonctionnalité est disponible pour les versions 2.5.0 et ultérieures du composant [Greengrass](#) nucleus.

- `systemResourceLimits`— Les limites de ressources système à appliquer aux processus de ce composant. Vous pouvez appliquer des limites de ressources système aux composants Lambda génériques et non conteneurisés. Pour de plus amples informations, veuillez consulter [Configuration des limites de ressources système pour les composants](#).

Vous pouvez définir l'une des options suivantes :

- `cpus`— Le temps processeur maximal que les processus de ce composant peuvent utiliser sur le périphérique principal. Le temps processeur total d'un appareil principal est équivalent au nombre de cœurs processeurs de l'appareil. Par exemple, sur un périphérique principal doté de 4 cœurs de processeur, vous pouvez définir cette valeur 2 pour limiter les processus de ce composant à 50 % d'utilisation de chaque cœur de processeur. Sur un appareil doté d'un cœur de processeur, vous pouvez définir cette valeur 0.25 pour limiter les processus de ce composant à 25 % d'utilisation du processeur. Si vous définissez cette valeur sur un nombre supérieur au nombre de cœurs de processeur, le logiciel AWS IoT Greengrass Core ne limite pas l'utilisation du processeur par le composant.
- `memory`— La quantité maximale de RAM (en kilo-octets) que les processus de ce composant peuvent utiliser sur le périphérique principal.

Cette fonctionnalité est disponible pour les versions 2.4.0 et ultérieures du composant [Greengrass](#) nucleus. AWS IoT Greengrass ne prend actuellement pas en charge cette fonctionnalité sur les appareils Windows principaux.

Exemple Exemple de mise à jour de configuration de base

L'exemple composants d'objet suivant indique de déployer un composant qui attend un paramètre de configuration nommé `pythonVersion`. `com.example.PythonRuntime`

```
{
  "targetArn": "targetArn",
  "deploymentName": "deploymentName",
  "components": {
    "com.example.PythonRuntime": {
      "componentVersion": "1.0.0",
      "configurationUpdate": {
        "merge": "{\"pythonVersion\":\"3.7\"}"
      }
    }
  }
}
```

Exemple Exemple de mise à jour de configuration avec mises à jour de réinitialisation et de fusion

Prenons un exemple de composant de tableau de bord industriel dont la configuration par défaut est la suivante. `com.example.IndustrialDashboard`

```
{
  "name": null,
  "mode": "REQUEST",
  "network": {
    "useHttps": true,
    "port": {
      "http": 80,
      "https": 443
    },
  },
  "tags": []
}
```

La mise à jour de configuration suivante spécifie les instructions suivantes :

1. Réinitialisez le paramètre HTTPS à sa valeur par défaut (`true`).

2. Réinitialisez la liste des tags industriels à une liste vide.
3. Fusionnez une liste d'étiquettes industrielles identifiant les flux de données de température et de pression pour deux chaudières.

```
{
  "reset": [
    "/network/useHttps",
    "/tags"
  ],
  "merge": {
    "tags": [
      "/boiler/1/temperature",
      "/boiler/1/pressure",
      "/boiler/2/temperature",
      "/boiler/2/pressure"
    ]
  }
}
```

L'exemple d'objet suivant indique de déployer ce composant de tableau de bord industriel et de mettre à jour la configuration.

```
{
  "targetArn": "targetArn",
  "deploymentName": "deploymentName",
  "components": {
    "com.example.IndustrialDashboard": {
      "componentVersion": "1.0.0",
      "configurationUpdate": {
        "reset": [
          "/network/useHttps",
          "/tags"
        ],
        "merge": "{\\"tags\\":[\\\"/boiler/1/temperature\\\",\\\"/boiler/1/pressure\\\",\\\"/boiler/2/temperature\\\",\\\"/boiler/2/pressure\\\"]}"
      }
    }
  }
}
```

5. (Facultatif) Définissez les politiques de déploiement pour le déploiement. Vous pouvez configurer à quel moment les périphériques principaux peuvent effectuer un déploiement en toute sécurité ou ce qu'il convient de faire si un périphérique principal ne parvient pas à effectuer le déploiement. Pour ce faire, ajoutez un `deploymentPolicies` objet à `deployment.json`, puis effectuez l'une des opérations suivantes :

1. (Facultatif) Spécifiez la politique de mise à jour des composants (`componentUpdatePolicy`). Cette politique définit si le déploiement permet ou non aux composants de différer une mise à jour jusqu'à ce qu'ils soient prêts à être mis à jour. Par exemple, les composants peuvent avoir besoin de nettoyer les ressources ou de terminer des actions critiques avant de pouvoir redémarrer pour appliquer une mise à jour. Cette politique définit également le délai dont disposent les composants pour répondre à une notification de mise à jour.

Cette politique est un objet doté des paramètres suivants :

- `action`— (Facultatif) S'il faut ou non informer les composants et attendre qu'ils signalent lorsqu'ils sont prêts à être mis à jour. Sélectionnez parmi les options suivantes :
 - `NOTIFY_COMPONENTS` : le déploiement avertit chaque composant avant qu'il ne s'arrête et met à jour ce composant. Les composants peuvent utiliser l'opération [SubscribeToComponentUpdates](#) IPC pour recevoir ces notifications.
 - `SKIP_NOTIFY_COMPONENTS` : le déploiement ne notifie pas les composants et n'attend pas qu'ils soient mis à jour en toute sécurité.

La valeur par défaut est `NOTIFY_COMPONENTS`.

- `timeoutInSeconds` Durée en secondes pendant laquelle chaque composant doit répondre à une notification de mise à jour avec l'opération [DeferComponentUpdate](#) IPC. Si le composant ne répond pas dans ce délai, le déploiement se poursuit sur le périphérique principal.

La valeur par défaut est de 60 secondes.

2. (Facultatif) Spécifiez la politique de validation de configuration (`configurationValidationPolicy`). Cette politique définit le temps dont dispose chaque composant pour valider une mise à jour de configuration à partir d'un déploiement. Les composants peuvent utiliser l'opération [SubscribeToValidateConfigurationUpdates](#) IPC pour s'abonner aux notifications relatives à leurs propres mises à jour de configuration. Les composants peuvent ensuite utiliser l'opération [SendConfigurationValidityReport](#) IPC pour

indiquer au logiciel AWS IoT Greengrass principal si la mise à jour de configuration est valide. Si la mise à jour de configuration n'est pas valide, le déploiement échoue.

Cette politique est un objet dont le paramètre est le suivant :

- `timeoutInSeconds`(Facultatif) Durée en secondes dont dispose chaque composant pour valider une mise à jour de configuration. Si le composant ne répond pas dans ce délai, le déploiement se poursuit sur le périphérique principal.

La valeur par défaut est de 30 secondes.

3. (Facultatif) Spécifiez la politique de gestion des défaillances (`failureHandlingPolicy`). Cette politique est une chaîne qui définit s'il faut ou non annuler les appareils en cas d'échec du déploiement. Sélectionnez parmi les options suivantes :

- `ROLLBACK`— Si le déploiement échoue sur un périphérique principal, le logiciel AWS IoT Greengrass Core rétablit la configuration précédente de ce périphérique principal.
- `DO_NOTHING`— Si le déploiement échoue sur un appareil principal, le logiciel AWS IoT Greengrass Core conserve la nouvelle configuration. Cela peut entraîner des composants cassés si la nouvelle configuration n'est pas valide.

La valeur par défaut est `ROLLBACK`.

Votre déploiement dans `deployment.json` peut ressembler à l'exemple suivant :

```
{
  "targetArn": "targetArn",
  "deploymentName": "deploymentName",
  "components": {
    "com.example.IndustrialDashboard": {
      "componentVersion": "1.0.0",
      "configurationUpdate": {
        "reset": [
          "/network/useHttps",
          "/tags"
        ],
        "merge": "{\\"tags\\":[\\\"/boiler/1/temperature\\\",\\\"/boiler/1/pressure\\\",\\\"/boiler/2/temperature\\\",\\\"/boiler/2/pressure\\\"]}"
      }
    }
  },
  "deploymentPolicies": {
```



```

    "componentUpdatePolicy": {
      "action": "NOTIFY_COMPONENTS",
      "timeoutInSeconds": 30
    },
    "configurationValidationPolicy": {
      "timeoutInSeconds": 60
    },
    "failureHandlingPolicy": "ROLLBACK"
  }
}

```

6. (Facultatif) Définissez le mode d'arrêt, de déploiement ou d'expiration du déploiement. AWS IoT Greengrass utilise des AWS IoT Core tâches pour envoyer des déploiements vers les appareils principaux. Ces options sont donc identiques aux options de configuration des AWS IoT Core tâches. Pour plus d'informations, consultez la section [Déploiement de Job et configuration d'abandon](#) dans le Guide du AWS IoT développeur.

Pour définir les options de la tâche, ajoutez un `iotJobConfiguration` objet à `deployment.json`. Définissez ensuite les options à configurer.

Votre déploiement dans `deployment.json` peut ressembler à l'exemple suivant :

```

{
  "targetArn": "targetArn",
  "deploymentName": "deploymentName",
  "components": {
    "com.example.IndustrialDashboard": {
      "componentVersion": "1.0.0",
      "configurationUpdate": {
        "reset": [
          "/network/useHttps",
          "/tags"
        ],
        "merge": "{\"tags\": [\"/boiler/1/temperature\", \"/boiler/1/pressure\", \"/boiler/2/temperature\", \"/boiler/2/pressure\"]}"
      }
    }
  },
  "deploymentPolicies": {
    "componentUpdatePolicy": {
      "action": "NOTIFY_COMPONENTS",
      "timeoutInSeconds": 30
    },

```

```
{
  "configurationValidationPolicy": {
    "timeoutInSeconds": 60
  },
  "failureHandlingPolicy": "ROLLBACK"
},
"iotJobConfiguration": {
  "abortConfig": {
    "criteriaList": [
      {
        "action": "CANCEL",
        "failureType": "ALL",
        "minNumberOfExecutedThings": 100,
        "thresholdPercentage": 5
      }
    ]
  },
  "jobExecutionsRolloutConfig": {
    "exponentialRate": {
      "baseRatePerMinute": 5,
      "incrementFactor": 2,
      "rateIncreaseCriteria": {
        "numberOfNotifiedThings": 10,
        "numberOfSucceededThings": 5
      }
    },
    "maximumPerMinute": 50
  },
  "timeoutConfig": {
    "inProgressTimeoutInMinutes": 5
  }
}
}
```

7. (Facultatif) Ajoutez des balises (tags) pour le déploiement. Pour de plus amples informations, veuillez consulter [Marquez vos AWS IoT Greengrass Version 2 ressources](#).
8. Exécutez la commande suivante pour créer le déploiement à partir `deployment.json`.

```
aws greengrassv2 create-deployment --cli-input-json file://deployment.json
```

La réponse inclut un `deploymentId` identifiant ce déploiement. Vous pouvez utiliser l'ID de déploiement pour vérifier l'état du déploiement. Pour de plus amples informations, veuillez consulter [Vérifier l'état du déploiement](#).

Mettre à jour les configurations des composants

Les configurations de composants sont des objets JSON qui définissent les paramètres de chaque composant. La recette de chaque composant définit sa configuration par défaut, que vous modifiez lorsque vous déployez des composants sur des appareils principaux.

Lorsque vous créez un déploiement, vous pouvez spécifier la mise à jour de configuration à appliquer pour chaque composant. Les mises à jour de configuration sont des opérations de correctif, ce qui signifie que la mise à jour modifie la configuration des composants qui existe sur le périphérique principal. Si le périphérique principal ne possède pas le composant, la mise à jour de configuration modifie et applique la configuration par défaut pour ce déploiement.

La mise à jour de configuration définit les mises à jour réinitialisées et les mises à jour de fusion. Les mises à jour de réinitialisation définissent les valeurs de configuration à rétablir par défaut ou à supprimer. Les mises à jour de fusion définissent les nouvelles valeurs de configuration à définir pour le composant. Lorsque vous déployez une mise à jour de configuration, le logiciel AWS IoT Greengrass Core exécute la mise à jour de réinitialisation avant la mise à jour de fusion.

Les composants peuvent valider les mises à jour de configuration que vous déployez. Le composant s'abonne pour recevoir une notification lorsqu'un déploiement change de configuration, et il peut rejeter une configuration qu'il ne prend pas en charge. Pour de plus amples informations, veuillez consulter [Interagir avec la configuration des composants](#).

Rubriques

- [Réinitialiser les mises à](#)
- [Fusion des mises à jour](#)
- [Exemples](#)

Réinitialiser les mises à

Les mises à jour de réinitialisation définissent les valeurs de configuration à rétablir à leurs valeurs par défaut sur le périphérique principal. Si aucune valeur de configuration n'a de valeur par défaut, la mise à jour de réinitialisation supprime cette valeur de la configuration du composant. Cela peut vous aider à réparer un composant qui se brise à la suite d'une configuration non valide.

Utilisez une liste de pointeurs JSON pour définir les valeurs de configuration à réinitialiser. Les pointeurs JSON commencent par une barre oblique/. Pour identifier une valeur dans une

configuration de composants imbriqués, utilisez des barres obliques (/) pour séparer les clés de chaque niveau de la configuration. Pour plus d'informations, consultez la [spécification du pointeur JSON](#).

 Note

Vous ne pouvez rétablir les valeurs par défaut que pour une liste entière. Vous ne pouvez pas utiliser les mises à jour de réinitialisation pour réinitialiser un élément individuel d'une liste.

Pour rétablir les valeurs par défaut de l'ensemble de la configuration d'un composant, spécifiez une seule chaîne vide comme mise à jour de réinitialisation.

```
"reset": [""]
```

Fusion des mises à jour

Les mises à jour de fusion définissent les valeurs de configuration à insérer dans la configuration des composants sur le noyau. La mise à jour de fusion est un objet JSON que le logiciel AWS IoT Greengrass Core fusionne après avoir réinitialisé les valeurs dans les chemins que vous spécifiez dans la mise à jour de réinitialisation. Lorsque vous utilisez le AWS CLI ou AWS SDKs, vous devez sérialiser cet objet JSON sous forme de chaîne.

Vous pouvez fusionner une paire clé-valeur qui n'existe pas dans la configuration par défaut du composant. Vous pouvez également fusionner une paire clé-valeur dont le type est différent de celui de la valeur associée à la même clé. La nouvelle valeur remplace l'ancienne valeur. Cela signifie que vous pouvez modifier la structure de l'objet de configuration.

Vous pouvez fusionner des valeurs nulles avec des chaînes, des listes et des objets vides.

 Note

Vous ne pouvez pas utiliser les mises à jour par fusion dans le but d'insérer ou d'ajouter un élément à une liste. Vous pouvez remplacer une liste complète ou définir un objet dans lequel chaque élément possède une clé unique.

AWS IoT Greengrass utilise JSON pour les valeurs de configuration. Le JSON spécifie un type de nombre mais ne fait pas la différence entre les nombres entiers et les nombres flottants. Par conséquent, les valeurs de configuration peuvent être converties en valeurs

flottantes. AWS IoT Greengrass Pour garantir que votre composant utilise le type de données approprié, nous vous recommandons de définir les valeurs de configuration numériques sous forme de chaînes. Ensuite, demandez à votre composant de les analyser sous forme de nombres entiers ou de nombres flottants. Cela garantit que vos valeurs de configuration sont du même type dans la configuration et sur votre appareil principal.

Utiliser des variables de recette dans les mises à jour de fusion

Cette fonctionnalité est disponible pour les versions 2.6.0 et ultérieures du composant [Greengrass](#) nucleus.

Si vous définissez l'option de [interpolateComponentConfiguration](#) configuration du noyau de Greengrass sur `true`, vous pouvez utiliser des variables de recette, autres que la variable de recette, dans les mises à `component_dependency_name:configuration:json_pointer` jour de fusion. Par exemple, vous pouvez utiliser la variable de `{iot:thingName}` recette dans une mise à jour de fusion pour inclure le nom de l' AWS IoT objet du périphérique principal dans une valeur de configuration du composant, telle qu'une politique d'[autorisation de communication interprocessus \(IPC\)](#).

Exemples

L'exemple suivant illustre les mises à jour de configuration pour un composant de tableau de bord dont la configuration par défaut est la suivante. Cet exemple de composant affiche des informations sur les équipements industriels.

```
{
  "name": null,
  "mode": "REQUEST",
  "network": {
    "useHttps": true,
    "port": {
      "http": 80,
      "https": 443
    },
  },
  "tags": []
}
```

Recette de composants de tableau de bord industriel

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.IndustrialDashboard",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "Displays information about industrial equipment.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "name": null,
      "mode": "REQUEST",
      "network": {
        "useHttps": true,
        "port": {
          "http": 80,
          "https": 443
        }
      },
      "tags": []
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "Run": "python3 -u {artifacts:path}/industrial_dashboard.py"
      }
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "Run": "py -3 -u {artifacts:path}/industrial_dashboard.py"
      }
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.IndustrialDashboard
ComponentVersion: '1.0.0'
ComponentDescription: Displays information about industrial equipment.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    name: null
    mode: REQUEST
    network:
      useHttps: true
      port:
        http: 80
        https: 443
    tags: []
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      Run: |
        python3 -u {artifacts:path}/industrial_dashboard.py
  - Platform:
      os: windows
    Lifecycle:
      Run: |
        py -3 -u {artifacts:path}/industrial_dashboard.py
```

Exemple Exemple 1 : mise à jour par fusion

Vous créez un déploiement qui applique la mise à jour de configuration suivante, qui spécifie une mise à jour par fusion mais pas une mise à jour de réinitialisation. Cette mise à jour de configuration indique au composant d'afficher le tableau de bord sur le port HTTP 8080 avec les données de deux chaudières.

Console

Configuration à fusionner

```
{
```

```
{
  "name": "Factory 2A",
  "network": {
    "useHttps": false,
    "port": {
      "http": 8080
    }
  },
  "tags": [
    "/boiler/1/temperature",
    "/boiler/1/pressure",
    "/boiler/2/temperature",
    "/boiler/2/pressure"
  ]
}
```

AWS CLI

La commande suivante crée un déploiement sur un périphérique principal.

```
aws greengrassv2 create-deployment --cli-input-json file://dashboard-deployment.json
```

Le `dashboard-deployment.json` fichier contient le document JSON suivant.

```
{
  "targetArn": "arn:aws:iot:us-west-2:123456789012:thing/MyGreengrassCore",
  "deploymentName": "Deployment for MyGreengrassCore",
  "components": {
    "com.example.IndustrialDashboard": {
      "componentVersion": "1.0.0",
      "configurationUpdate": {
        "merge": "{\"name\":\"Factory 2A\",\"network\":{\"useHttps\":false,\"port\":{\"http\":8080}},\"tags\":[\"/boiler/1/temperature\",\"/boiler/1/pressure\",\"/boiler/2/temperature\",\"/boiler/2/pressure\"]}"
      }
    }
  }
}
```

Greengrass CLI

La commande [Greengrass CLI](#) suivante crée un déploiement local sur un périphérique principal.


```
sudo greengrass-cli deployment create \  
  --recipeDir recipes \  
  --artifactDir artifacts \  
  --merge "com.example.IndustrialDashboard=1.0.0" \  
  --update-config dashboard-configuration.json
```

Le `dashboard-configuration.json` fichier contient le document JSON suivant.

```
{  
  "com.example.IndustrialDashboard": {  
    "MERGE": {  
      "name": "Factory 2A",  
      "network": {  
        "useHttps": false,  
        "port": {  
          "http": 8080  
        }  
      },  
      "tags": [  
        "/boiler/1/temperature",  
        "/boiler/1/pressure",  
        "/boiler/2/temperature",  
        "/boiler/2/pressure"  
      ]  
    }  
  }  
}
```

Après cette mise à jour, le composant du tableau de bord possède la configuration suivante.

```
{  
  "name": "Factory 2A",  
  "mode": "REQUEST",  
  "network": {  
    "useHttps": false,  
    "port": {  
      "http": 8080,  
      "https": 443  
    }  
  },  
  "tags": [  

```

```
"/boiler/1/temperature",
"/boiler/1/pressure",
"/boiler/2/temperature",
"/boiler/2/pressure"
]
}
```

Exemple Exemple 2 : réinitialiser et fusionner les mises à jour

Vous créez ensuite un déploiement qui applique la mise à jour de configuration suivante, qui spécifie une mise à jour de réinitialisation et une mise à jour de fusion. Ces mises à jour indiquent d'afficher le tableau de bord sur le port HTTPS par défaut avec les données des différentes chaudières. Ces mises à jour modifient la configuration qui résulte des mises à jour de configuration de l'exemple précédent.

Console

Réinitialiser les chemins

```
[
  "/network/useHttps",
  "/tags"
]
```

Configuration à fusionner

```
{
  "tags": [
    "/boiler/3/temperature",
    "/boiler/3/pressure",
    "/boiler/4/temperature",
    "/boiler/4/pressure"
  ]
}
```

AWS CLI

La commande suivante crée un déploiement sur un périphérique principal.

```
aws greengrassv2 create-deployment --cli-input-json file://dashboard-
deployment2.json
```

Le dashboard-deployment2.json fichier contient le document JSON suivant.

```
{
  "targetArn": "arn:aws:iot:us-west-2:123456789012:thing/MyGreengrassCore",
  "deploymentName": "Deployment for MyGreengrassCore",
  "components": {
    "com.example.IndustrialDashboard": {
      "componentVersion": "1.0.0",
      "configurationUpdate": {
        "reset": [
          "/network/useHttps",
          "/tags"
        ],
        "merge": "{\"tags\": [\"/boiler/3/temperature\", \"/boiler/3/pressure\", \"/boiler/4/temperature\", \"/boiler/4/pressure\"]}"
      }
    }
  }
}
```

Greengrass CLI

La commande [Greengrass CLI](#) suivante crée un déploiement local sur un périphérique principal.

```
sudo greengrass-cli deployment create \
  --recipeDir recipes \
  --artifactDir artifacts \
  --merge "com.example.IndustrialDashboard=1.0.0" \
  --update-config dashboard-configuration2.json
```

Le dashboard-configuration2.json fichier contient le document JSON suivant.

```
{
  "com.example.IndustrialDashboard": {
    "RESET": [
      "/network/useHttps",
      "/tags"
    ],
    "MERGE": {
      "tags": [
        "/boiler/3/temperature",
        "/boiler/3/pressure",
        "/boiler/4/temperature",
```

```
        "/boiler/4/pressure"  
      ]  
    }  
  }  
}
```

Après cette mise à jour, le composant du tableau de bord possède la configuration suivante.

```
{  
  "name": "Factory 2A",  
  "mode": "REQUEST",  
  "network": {  
    "useHttps": true,  
    "port": {  
      "http": 8080,  
      "https": 443  
    }  
  },  
  "tags": [  
    "/boiler/3/temperature",  
    "/boiler/3/pressure",  
    "/boiler/4/temperature",  
    "/boiler/4/pressure",  
  ]  
}
```

Création de sous-déploiements

Note

La fonctionnalité de sous-déploiement est disponible sur Greengrass nucleus version 2.9.0 et versions ultérieures. Il n'est pas possible de déployer une configuration dans un sous-déploiement comportant des versions antérieures des composants du noyau Greengrass.

Un sous-déploiement est un déploiement qui cible un plus petit sous-ensemble d'appareils au sein d'un déploiement parent. Vous pouvez utiliser des sous-déploiements pour déployer une configuration sur un plus petit sous-ensemble d'appareils. Vous pouvez également créer des sous-déploiements pour réessayer un déploiement parent infructueux lorsqu'un ou plusieurs appareils de ce déploiement parent échouent. Grâce à cette fonctionnalité, vous pouvez sélectionner les

appareils qui ont échoué lors de ce déploiement parent et créer un sous-déploiement pour tester les configurations jusqu'à ce que le sous-déploiement soit réussi. Une fois le sous-déploiement réussi, vous pouvez redéploier cette configuration vers le déploiement parent.

Suivez les étapes décrites dans cette section pour créer un sous-déploiement et vérifier son statut. Pour plus d'informations sur la création de déploiements, consultez la section [Créer des déploiements](#).

Pour créer un sous-déploiement (AWS CLI)

1. Exécutez la commande suivante pour récupérer les derniers déploiements d'un groupe d'objets. Remplacez l'ARN de la commande par l'ARN du groupe d'objets à interroger. Configurez **LATEST_ONLY** sur `--history-filter` pour voir le dernier déploiement de ce groupe d'objets.

```
aws greengrassv2 list-deployments --target-arn arn:aws:iot:region:account-id:thinggroup/thingGroupName --history-filter LATEST_ONLY
```

2. Copiez le contenu `deploymentId` de la réponse à la `list-deployments` commande à utiliser à l'étape suivante.
3. Exécutez la commande suivante pour récupérer l'état d'un déploiement. Remplacez *deploymentId* par l'ID du déploiement à interroger.

```
aws greengrassv2 get-deployment --deployment-id deploymentId
```

4. Copiez le contenu `iotJobId` de la réponse à la `get-deployment` commande à utiliser à l'étape suivante.
5. Exécutez la commande suivante pour récupérer la liste des exécutions de tâches pour la tâche spécifiée. Remplacez *jobID* par celui `iotJobId` de l'étape précédente. Remplacez *status* par le statut pour lequel vous souhaitez filtrer. Vous pouvez filtrer les résultats avec les statuts suivants :

- QUEUED
- IN_PROGRESS
- SUCCEEDED
- FAILED
- TIMED_OUT
- REJECTED

- REMOVED
- CANCELED

```
aws iot list-job-executions-for-job --job-id jobID --status status
```

6. Créez un nouveau AWS IoT groupe d'objets ou utilisez un groupe d'objets existant pour votre sous-déploiement. Ajoutez ensuite un AWS IoT objet à ce groupe d'objets. Vous utilisez des groupes d'objets pour gérer des flottes d'appareils principaux de Greengrass. Lorsque vous déployez des composants logiciels sur vos appareils, vous pouvez cibler des appareils individuels ou des groupes d'appareils. Vous pouvez ajouter un appareil à un groupe d'objets avec un déploiement Greengrass actif. Une fois ajoutés, vous pouvez déployer les composants logiciels de ce groupe d'objets sur cet appareil.

Pour créer un nouveau groupe d'objets et y ajouter vos appareils, procédez comme suit :

- a. Créez un groupe AWS IoT d'objets. Remplacez *MyGreengrassCoreGroup* par le nom du nouveau groupe d'objets. Vous ne pouvez pas utiliser de deux-points (:) dans le nom d'un groupe d'objets.

 Note

Si un groupe d'objets pour un sous-déploiement est utilisé avec un `autreparentTargetArn`, il ne peut pas être réutilisé avec un autre parc parent. Si un groupe d'objets a déjà été utilisé pour créer un sous-déploiement pour une autre flotte, l'API renvoie une erreur.

```
aws iot create-thing-group --thing-group-name MyGreengrassCoreGroup
```

Si la demande aboutit, la réponse ressemble à l'exemple suivant :

```
{
  "thingGroupName": "MyGreengrassCoreGroup",
  "thingGroupArn": "arn:aws:iot:us-
west-2:123456789012:thinggroup/MyGreengrassCoreGroup",
  "thingGroupId": "4df721e1-ff9f-4f97-92dd-02db4e3f03aa"
}
```

- b. Ajoutez un noyau Greengrass provisionné à votre groupe d'objets. Exécutez la commande suivante avec les paramètres suivants :

- *MyGreengrassCore* Remplacez-le par le nom de votre noyau Greengrass provisionné.
- *MyGreengrassCoreGroup* Remplacez-le par le nom de votre groupe d'objets.

```
aws iot add-thing-to-thing-group --thing-name MyGreengrassCore --thing-group-name MyGreengrassCoreGroup
```

La commande n'a aucune sortie si la demande aboutit.

7. Créez un fichier appelé `deployment.json`, puis copiez l'objet JSON suivant dans le fichier. *targetArn* Remplacez-le par l'ARN du AWS IoT groupe d'objets à cibler pour le sous-déploiement. Une cible de sous-déploiement ne peut être qu'un groupe d'objets. Les groupes ARNs d'objets ont le format suivant :

- Groupe d'objets — `arn:aws:iot:region:account-id:thinggroup/thingGroupName`

```
{  
  "targetArn": "targetArn"  
}
```

8. Exécutez à nouveau la commande suivante pour obtenir les détails du déploiement d'origine. Ces détails incluent les métadonnées, les composants et la configuration des tâches. Remplacez *deploymentId* par l'identifiant de [Step 1](#). Vous pouvez utiliser cette configuration de déploiement pour configurer votre sous-déploiement et apporter les modifications nécessaires.

```
aws greengrassv2 get-deployment --deployment-id deploymentId
```

La réponse contient les détails du déploiement. Copiez l'une des paires clé-valeur suivantes depuis la réponse de la `get-deployment` commande dans `deployment.json`. Vous pouvez modifier ces valeurs pour le sous-déploiement. Pour plus d'informations sur les détails de cette commande, consultez [GetDeployment](#).

- `components`— Les composants du déploiement. Pour désinstaller un composant, supprimez-le de cet objet.
- `deploymentName`— Le nom du déploiement.

- `deploymentPolicies`— Les politiques du déploiement.
 - `iotJobConfiguration`— La configuration des tâches du déploiement.
 - `parentTargetArn`— La cible du déploiement parent.
 - `tags`— Les tags du déploiement.
9. Exécutez la commande suivante pour créer le sous-déploiement à partir de `deployment.json`. *subdeploymentName* Remplacez-le par un nom pour le sous-déploiement.

```
aws greengrassv2 create-deployment --deployment-name subdeploymentName --cli-input-json file://deployment.json
```

La réponse inclut un identifiant `deploymentId` ce sous-déploiement. Vous pouvez utiliser l'ID de déploiement pour vérifier l'état du déploiement. Pour plus d'informations, voir [Vérifier l'état du déploiement](#).

10. Si le sous-déploiement est réussi, vous pouvez utiliser sa configuration pour modifier le déploiement parent. Copiez celui `deployment.json` que vous avez utilisé à l'étape précédente. Remplacez `targetArn` le fichier JSON par l'ARN du déploiement parent et exécutez la commande suivante pour créer le déploiement parent à l'aide de cette nouvelle configuration.

 Note

Si vous créez une nouvelle révision de déploiement du parc parent, elle remplace toutes les révisions et sous-déploiements de ce déploiement parent. Pour plus d'informations, consultez la section [Réviser les déploiements](#).

```
aws greengrassv2 create-deployment --cli-input-json file://deployment.json
```

La réponse inclut un `deploymentId` identifiant ce déploiement. Vous pouvez utiliser l'ID de déploiement pour vérifier l'état du déploiement. Pour de plus amples informations, veuillez consulter [Vérifier l'état du déploiement](#).

Réviser les déploiements

Chaque objet ou groupe d'objets cible peut faire l'objet d'un déploiement actif à la fois. Lorsque vous créez un déploiement pour une cible déjà déployée, les composants logiciels du nouveau

déploiement remplacent ceux du déploiement précédent. Si le nouveau déploiement ne définit aucun composant défini par le déploiement précédent, le logiciel AWS IoT Greengrass Core supprime ce composant des équipements principaux cibles. Vous pouvez modifier un déploiement existant afin de ne pas supprimer les composants exécutés sur les appareils principaux d'un déploiement précédent vers une cible.

Pour réviser un déploiement, vous créez un déploiement qui part des mêmes composants et configurations que ceux d'un déploiement précédent. Vous utilisez l'[CreateDeployment](#) opération, qui est la même que celle que vous utilisez pour [créer des déploiements](#).

Pour réviser un déploiement (AWS CLI)

1. Exécutez la commande suivante pour répertorier les déploiements pour la cible de déploiement. Remplacez *targetArn* par l'ARN de l' AWS IoT objet ou du groupe d'objets cible.

```
aws greengrassv2 list-deployments --target-arn targetArn
```

La réponse contient une liste du dernier déploiement pour la cible. Copiez le `deploymentId` contenu de la réponse pour l'utiliser à l'étape suivante.

 Note

Vous pouvez également modifier un déploiement autre que la dernière révision pour la cible. Spécifiez l'`--history-filter ALL` argument pour répertorier tous les déploiements pour la cible. Copiez ensuite l'ID du déploiement que vous souhaitez modifier.

2. Exécutez la commande suivante pour obtenir les détails du déploiement. Ces détails incluent les métadonnées, les composants et la configuration des tâches. *deploymentId* Remplacez-le par l'ID de l'étape précédente.

```
aws greengrassv2 get-deployment --deployment-id deploymentId
```

La réponse contient les détails du déploiement.

3. Créez un fichier appelé `deployment.json` et copiez la réponse de la commande précédente dans le fichier.
4. Supprimez les paires clé-valeur suivantes de l'objet JSON dans `deployment.json` :

- deploymentId
- revisionId
- iotJobId
- iotJobArn
- creationTimestamp
- isLatestForTarget
- deploymentStatus

L'[CreateDeployment](#) opération attend une charge utile dont la structure est la suivante.

```
{
  "targetArn": "String",
  "components": Map of components,
  "deploymentPolicies": DeploymentPolicies,
  "iotJobConfiguration": DeploymentIoTJobConfiguration,
  "tags": Map of tags
}
```

5. Dans `deployment.json`, effectuez l'une des actions suivantes :
 - Modifiez le nom du déploiement (`deploymentName`).
 - Modifiez les composants du déploiement (`components`).
 - Modifiez les politiques du déploiement (`deploymentPolicies`).
 - Modifiez la configuration des tâches du déploiement (`iotJobConfiguration`).
 - Modifiez les balises du déploiement (`tags`).

Pour plus d'informations sur la définition de ces détails de déploiement, consultez [Créer des déploiements](#).

6. Exécutez la commande suivante pour créer le déploiement à partir de `deployment.json`.

```
aws greengrassv2 create-deployment --cli-input-json file://deployment.json
```

La réponse inclut un `deploymentId` identifiant ce déploiement. Vous pouvez utiliser l'ID de déploiement pour vérifier l'état du déploiement. Pour de plus amples informations, veuillez consulter [Vérifier l'état du déploiement](#).

Annuler les déploiements

Vous pouvez annuler un déploiement actif pour empêcher l'installation de ses composants logiciels sur les appareils AWS IoT Greengrass principaux. Si vous annulez un déploiement qui cible un groupe d'objets, les principaux appareils que vous ajoutez au groupe ne bénéficieront pas de ce déploiement continu. Si un appareil principal exécute déjà le déploiement, vous ne modifierez pas les composants de cet appareil lorsque vous annulez le déploiement. Vous devez [créer un nouveau déploiement](#) ou [réviser le déploiement](#) pour modifier les composants qui s'exécutent sur les appareils principaux ayant reçu le déploiement annulé.

Pour annuler un déploiement (AWS CLI)

1. Exécutez la commande suivante pour trouver l'ID de la dernière révision de déploiement pour une cible. La dernière révision est le seul déploiement qui peut être actif pour une cible, car les déploiements précédents sont annulés lorsque vous créez une nouvelle révision. Remplacez *targetArn* par l'ARN de l' AWS IoT objet ou du groupe d'objets cible.

```
aws greengrassv2 list-deployments --target-arn targetArn
```

La réponse contient une liste du dernier déploiement pour la cible. Copiez le `deploymentId` contenu de la réponse pour l'utiliser à l'étape suivante.

2. Exécutez la commande suivante pour annuler le déploiement. *deploymentId* Remplacez-le par l'ID de l'étape précédente.

```
aws greengrassv2 cancel-deployment --deployment-id deploymentId
```

Si l'opération réussit, l'état du déploiement passe à `CANCELED`.

Vérifier l'état du déploiement

Vous pouvez vérifier l'état d'un déploiement que vous créez dans AWS IoT Greengrass. Vous pouvez également vérifier l'état des AWS IoT tâches qui déploient le déploiement sur chaque appareil principal. Lorsqu'un déploiement est actif, le statut de la AWS IoT tâche est le suivant `IN_PROGRESS`. Une fois que vous avez créé une nouvelle révision d'un déploiement, le statut de la AWS IoT tâche de la révision précédente passe à `CANCELLED`.

Rubriques

- [Vérifier l'état du déploiement](#)
- [Vérifier l'état de déploiement de l'appareil](#)

Vérifier l'état du déploiement

Vous pouvez vérifier l'état d'un déploiement que vous identifiez par sa cible ou son ID.

Pour vérifier l'état du déploiement par cible (AWS CLI)

- Exécutez la commande suivante pour récupérer l'état du dernier déploiement pour une cible. *targetArn* Remplacez-le par le Amazon Resource Name (ARN) de l' AWS IoT objet ou du groupe d'objets ciblé par le déploiement.

```
aws greengrassv2 list-deployments --target-arn targetArn
```

La réponse contient une liste du dernier déploiement pour la cible. Cet objet de déploiement inclut l'état du déploiement.

Pour vérifier l'état du déploiement par ID (AWS CLI)

- Exécutez la commande suivante pour récupérer l'état d'un déploiement. Remplacez *deploymentId* par l'ID du déploiement à interroger.

```
aws greengrassv2 get-deployment --deployment-id deploymentId
```

La réponse contient l'état du déploiement.

Vérifier l'état de déploiement de l'appareil

Vous pouvez vérifier l'état d'une tâche de déploiement qui s'applique à un périphérique principal individuel. Vous pouvez également vérifier l'état d'une tâche de déploiement pour le déploiement d'un groupe d'objets.

Pour vérifier l'état des tâches de déploiement pour un périphérique principal ()AWS CLI

- Exécutez la commande suivante pour récupérer l'état de toutes les tâches de déploiement pour un périphérique principal. Remplacez *coreDeviceName* par le nom du périphérique principal à interroger.

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

La réponse contient la liste des tâches de déploiement pour le périphérique principal. Vous pouvez identifier la tâche à déployer à l'aide de la fonction `deploymentId` ou `targetArn`. Chaque tâche de déploiement contient l'état de la tâche sur le périphérique principal.

Pour vérifier les statuts de déploiement d'un groupe d'objets ()AWS CLI

1. Exécutez la commande suivante pour récupérer l'ID d'un déploiement existant. Remplacez *targetArn* par l'ARN du groupe d'objets cible.

```
aws greengrassv2 list-deployments --target-arn targetArn
```

La réponse contient une liste du dernier déploiement pour la cible. Copiez le `deploymentId` contenu de la réponse pour l'utiliser à l'étape suivante.

 Note

Vous pouvez également répertorier un déploiement autre que le dernier déploiement pour la cible. Spécifiez l'argument `--history-filter ALL` pour répertorier tous les déploiements pour la cible. Copiez ensuite l'ID du déploiement dont vous souhaitez vérifier l'état.

2. Exécutez la commande suivante pour obtenir les détails du déploiement. Remplacez-le par l'ID de l'étape précédente.

```
aws greengrassv2 get-deployment --deployment-id deploymentId
```

La réponse contient des informations sur le déploiement. Copiez le `iotJobId` contenu de la réponse à utiliser à l'étape suivante.

3. Exécutez la commande suivante pour décrire l'exécution de la tâche d'un périphérique principal pour le déploiement. Remplacez *iotJobId* et *coreDeviceThingName* par l'ID de tâche de l'étape précédente et le périphérique principal dont vous souhaitez vérifier l'état.

```
aws iot describe-job-execution --job-id iotJobId --thing-name coreDeviceThingName
```

La réponse contient l'état de l'exécution de la tâche de déploiement du périphérique principal et des détails sur cet état. `detailsMap` contient les informations suivantes :

- `detailed-deployment-status`— L'état du résultat du déploiement, qui peut prendre l'une des valeurs suivantes :
 - `SUCCESSFUL`— Le déploiement a réussi.
 - `FAILED_NO_STATE_CHANGE`— Le déploiement a échoué alors que le périphérique principal se préparait à effectuer le déploiement.
 - `FAILED_ROLLBACK_NOT_REQUESTED`— Le déploiement a échoué et le déploiement n'a pas spécifié de revenir à une configuration de travail précédente. Le périphérique principal ne fonctionne donc peut-être pas correctement.
 - `FAILED_ROLLBACK_COMPLETE`— Le déploiement a échoué et le périphérique principal a réussi à revenir à une configuration de travail précédente.
 - `FAILED_UNABLE_TO_ROLLBACK`— Le déploiement a échoué et le périphérique principal n'a pas pu revenir à une configuration fonctionnelle précédente. Il se peut donc que le périphérique principal ne fonctionne pas correctement.

Si le déploiement a échoué, vérifiez la `deployment-failure-cause` valeur et les fichiers journaux du périphérique principal pour identifier le problème. Pour plus d'informations sur la façon d'accéder aux fichiers journaux du périphérique principal, consultez [AWS IoT Greengrass Journaux de surveillance](#).

- `deployment-failure-cause`— Message d'erreur fournissant des informations supplémentaires sur les raisons pour lesquelles l'exécution de la tâche a échoué.

La réponse ressemble à l'exemple suivant.

```
{
  "execution": {
    "jobId": "2cc2698a-5175-48bb-adf2-1dd345606ebd",
    "status": "FAILED",
    "statusDetails": {
      "detailsMap": {
        "deployment-failure-cause": "No local or cloud component version satisfies the requirements. Check whether the version constraints conflict and that the component exists in your Compte AWS with a version that matches the version constraints. If the version constraints conflict, revise deployments to resolve the conflict. Component com.example.HelloWorld version constraints:
```

```
LOCAL_DEPLOYMENT requires =1.0.0, thinggroup/MyGreengrassCoreGroup requires
=1.0.1.",
    "detailed-deployment-status": "FAILED_NO_STATE_CHANGE"
  },
  "thingArn": "arn:aws:iot:us-west-2:123456789012:thing/MyGreengrassCore",
  "queuedAt": "2022-02-15T14:45:53.098000-08:00",
  "startedAt": "2022-02-15T14:46:05.670000-08:00",
  "lastUpdatedAt": "2022-02-15T14:46:20.892000-08:00",
  "executionNumber": 1,
  "versionNumber": 3
}
```

Connexion et surveillance AWS IoT Greengrass

La surveillance joue un rôle important dans le maintien de la fiabilité, de la disponibilité AWS IoT Greengrass et des performances de vos AWS solutions. Vous devez collecter des données de surveillance provenant de toutes les parties de votre AWS solution afin de pouvoir corriger plus facilement une défaillance multipoint, le cas échéant. Avant de commencer la surveillance AWS IoT Greengrass, vous devez créer un plan de surveillance qui inclut 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 ?

Rubriques

- [Outils de surveillance](#)
- [AWS IoT Greengrass Journaux de surveillance](#)
- [Enregistrez les appels AWS IoT Greengrass V2 d'API avec AWS CloudTrail](#)
- [Collectez les données de télémétrie relatives à l'état du système à partir des principaux appareils AWS IoT Greengrass](#)
- [Recevez des notifications relatives au déploiement et à l'état de santé des composants](#)
- [Vérifiez l'état de l'appareil Greengrass Core](#)

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 suivants pour surveiller AWS IoT Greengrass et signaler les problèmes :

- Amazon CloudWatch Logs — Surveillez, stockez et accédez à vos fichiers journaux depuis AWS CloudTrail ou d'autres sources. Pour plus d'informations, consultez la section [Surveillance des fichiers journaux](#) dans le guide de CloudWatch l'utilisateur Amazon.
- AWS CloudTrail Surveillance des journaux : partagez les fichiers journaux entre les comptes, surveillez les fichiers CloudTrail CloudWatch journaux en temps réel en les envoyant à Logs, 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 plus d'informations, consultez la section [Utilisation des fichiers CloudTrail journaux](#) dans le Guide de AWS CloudTrail l'utilisateur.
- Télémétrie de santé du système Greengrass — Abonnez-vous pour recevoir les données de télémétrie envoyées depuis le cœur de Greengrass. Pour de plus amples informations, veuillez consulter [the section called “Collectez les données de télémétrie relatives à l'état du système”](#).
- Notifications relatives à l'état de l'appareil Créez des événements EventBridge à l'aide d'Amazon pour recevoir des mises à jour sur l'état des déploiements et des composants. Pour de plus amples informations, veuillez consulter [Recevez des notifications relatives au déploiement et à l'état de santé des composants](#).
- Service d'état du parc : utilisez les opérations de l'API d'état du parc pour vérifier l'état des principaux appareils et de leurs composants Greengrass. Vous pouvez également consulter les informations sur l'état du parc dans la AWS IoT Greengrass console. Pour de plus amples informations, veuillez consulter [Vérifiez l'état de l'appareil Greengrass Core](#).

AWS IoT Greengrass Journaux de surveillance

AWS IoT Greengrass comprend le service cloud et le logiciel AWS IoT Greengrass de base. Le logiciel AWS IoT Greengrass Core peut écrire des CloudWatch journaux sur Amazon Logs et dans le système de fichiers local de l'appareil principal. Les composants Greengrass qui s'exécutent sur le périphérique principal 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 du AWS IoT Greengrass journal incluent un horodatage, un niveau de journalisation et des informations sur l'événement.

Par défaut, le logiciel AWS IoT Greengrass Core écrit les journaux uniquement dans le système de fichiers local. Vous pouvez consulter les journaux du système de fichiers en temps réel afin de pouvoir déboguer les composants Greengrass que vous développez et déployez. Vous pouvez également configurer un périphérique principal pour écrire des journaux dans CloudWatch des journaux, afin de pouvoir dépanner le périphérique principal sans accéder au système de fichiers

local. Pour de plus amples informations, veuillez consulter [Activer la journalisation dans CloudWatch Logs](#).

Rubriques

- [Accéder aux journaux du système de fichiers](#)
- [CloudWatch Journaux d'accès](#)
- [Accédez aux journaux de service du système](#)
- [Activer la journalisation dans CloudWatch Logs](#)
- [Configurer la journalisation pour AWS IoT Greengrass](#)
- [AWS CloudTrail journaux](#)

Accéder aux journaux du système de fichiers

Le logiciel AWS IoT Greengrass Core stocke les journaux dans le `/greengrass/v2/logs` dossier d'un périphérique principal, où se `/greengrass/v2` trouve le chemin d'accès au dossier AWS IoT Greengrass racine. La structure du dossier logs est la suivante.

```
/greengrass/v2
### logs
  ### greengrass.log
  ### greengrass_2021_09_14_15_0.log
  ### ComponentName.log
  ### ComponentName_2021_09_14_15_0.log
  ### main.log
```

- `greengrass.log`— Le fichier journal du logiciel AWS IoT Greengrass Core. Utilisez ce fichier journal pour afficher des informations en temps réel sur les composants et les déploiements. Ce fichier journal inclut les journaux du noyau Greengrass, qui est le cœur du logiciel AWS IoT Greengrass Core, et les composants du plugin, tels que le gestionnaire de [journaux et le gestionnaire](#) de [secrets](#).
- `ComponentName.log`— Fichiers journaux des composants Greengrass. Utilisez les fichiers journaux des composants pour afficher des informations en temps réel sur un composant Greengrass qui s'exécute sur le périphérique principal. Les composants génériques et les composants Lambda écrivent une sortie standard (stdout) et une erreur standard (stderr) dans ces fichiers journaux.

- `main.log`— Le fichier journal du `main` service qui gère le cycle de vie des composants. Ce fichier journal sera toujours vide.

Pour plus d'informations sur les différences entre les composants du plugin, les composants génériques et les composants Lambda, consultez. [Types de composants](#)

Les considérations suivantes s'appliquent lorsque vous utilisez les journaux du système de fichiers :

- Autorisations de l'utilisateur root

Vous devez disposer des autorisations root pour lire AWS IoT Greengrass les journaux du système de fichiers.

- Rotation du fichier journal

Le logiciel AWS IoT Greengrass Core fait pivoter les fichiers journaux toutes les heures ou lorsqu'ils dépassent une limite de taille de fichier. Les fichiers journaux pivotés contiennent un horodatage dans leur nom de fichier. Par exemple, un fichier journal du logiciel AWS IoT Greengrass Core pivoté peut être nommé `greengrass_2021_09_14_15_0.log`. La limite de taille de fichier par défaut est de 1 024 Ko (1 Mo). Vous pouvez configurer la limite de taille de fichier sur le composant [Greengrass nucleus](#).

- Suppression du fichier journal

Le logiciel AWS IoT Greengrass Core nettoie les fichiers journaux antérieurs lorsque la taille des fichiers journaux du logiciel AWS IoT Greengrass Core ou des fichiers journaux des composants Greengrass, y compris les fichiers journaux pivotés, dépasse une limite d'espace disque. La limite d'espace disque par défaut pour le journal du logiciel AWS IoT Greengrass Core et pour chaque journal des composants est de 10 240 Ko (10 Mo). Vous pouvez configurer la limite d'espace disque des journaux du logiciel AWS IoT Greengrass Core sur le composant [Greengrass nucleus](#) ou sur le composant du [gestionnaire de journaux](#). Vous pouvez configurer la limite d'espace disque de journal de chaque composant sur le [composant du gestionnaire de journaux](#).

Pour consulter le fichier journal du logiciel AWS IoT Greengrass Core

- Exécutez la commande suivante pour afficher le fichier journal en temps réel. Remplacez `/greengrass/v2` par le chemin d'accès au dossier AWS IoT Greengrass racine.

Linux or Unix

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

Windows Command Prompt (CMD)

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

La type commande écrit le contenu du fichier sur le terminal. Exécutez cette commande plusieurs fois pour observer les modifications apportées au fichier.

PowerShell

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

Pour consulter le fichier journal d'un composant

- Exécutez la commande suivante pour afficher le fichier journal en temps réel. Remplacez */greengrass/v2* ou *C:\greengrass\v2* par le chemin d'accès au dossier AWS IoT Greengrass racine, puis remplacez *com.example.HelloWorld* par le nom du composant.

Linux or Unix

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

PowerShell

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

Vous pouvez également utiliser la logs commande de la [CLI Greengrass](#) pour analyser les journaux Greengrass sur un périphérique principal. Pour utiliser la logs commande, vous devez configurer le [noyau Greengrass](#) pour générer des fichiers journaux au format JSON. Pour plus d'informations, consultez [Interface de ligne de commande Greengrass](#) et [journaux](#).

CloudWatch Journaux d'accès

Vous pouvez déployer le [composant du gestionnaire de journaux](#) pour configurer le périphérique principal afin qu'il écrive dans CloudWatch Logs. Pour de plus amples informations, veuillez consulter [Activer la journalisation dans CloudWatch Logs](#). Vous pouvez ensuite consulter les journaux sur la page Logs de la CloudWatch console Amazon ou à l'aide de l'API CloudWatch Logs.

Nom du groupe de journaux

```
/aws/greengrass/componentType/region/componentName
```

Le nom du groupe de journaux utilise les variables suivantes :

- **componentType**— Le type du composant, qui peut être l'un des suivants :
 - **GreengrassSystemComponent**— Ce groupe de journaux inclut les journaux des composants du noyau et du plugin, qui s'exécutent dans la même JVM que le noyau Greengrass. Le composant fait partie du noyau de [Greengrass](#).
 - **UserComponent**— Ce groupe de journaux inclut les journaux des composants génériques, des composants Lambda et d'autres applications de l'appareil. Le composant ne fait pas partie du noyau de Greengrass.

Pour de plus amples informations, veuillez consulter [Types de composants](#).

- **region**— La AWS région utilisée par l'appareil principal.
- **componentName**— Le nom du composant. Pour les journaux système, cette valeur est `System`.

Nom du flux de log

```
/date/thing/thingName
```

Le nom du flux de journal utilise les variables suivantes :

- **date**— La date du journal, telle que `2020/12/15`. Le composant du gestionnaire de journaux utilise le `yyyy/MM/dd` format.
- **thingName**— Le nom de l'appareil principal.

 Note

Si le nom d'un objet contient deux points (:), le gestionnaire de journaux remplace les deux points par un signe plus (+).

Les considérations suivantes s'appliquent lorsque vous utilisez le composant du gestionnaire de CloudWatch journaux pour écrire dans Logs :

- Retards de journalisation

 Note

Nous vous recommandons de passer à la version 2.3.0 du gestionnaire de journaux, qui réduit les délais de journalisation pour les fichiers journaux actifs et pivotés. Lorsque vous passez à Log Manager 2.3.0, nous vous recommandons également de passer à Greengrass nucleus 2.9.1.

Le composant du gestionnaire de journaux version 2.2.8 (et versions antérieures) traite et télécharge les journaux uniquement à partir de fichiers journaux ayant fait l'objet d'une rotation. Par défaut, le logiciel AWS IoT Greengrass Core fait pivoter les fichiers journaux toutes les heures ou une fois qu'ils atteignent 1 024 Ko. Par conséquent, le composant du gestionnaire de journaux ne télécharge les journaux qu'une fois que le logiciel AWS IoT Greengrass Core ou un composant Greengrass a écrit plus de 1 024 Ko de journaux. Vous pouvez configurer une limite de taille de fichier journal inférieure afin de provoquer une rotation plus fréquente des fichiers journaux. Le composant du gestionnaire de journaux télécharge donc les CloudWatch journaux dans Logs plus fréquemment.

Le composant du gestionnaire de journaux version 2.3.0 (et versions ultérieures) traite et télécharge tous les journaux. Lorsque vous rédigez un nouveau journal, la version 2.3.0 (et versions ultérieures) du gestionnaire de journaux traite et télécharge directement le fichier journal actif au lieu d'attendre qu'il fasse l'objet d'une rotation. Cela signifie que vous pouvez consulter le nouveau journal en 5 minutes ou moins.

Le composant du gestionnaire de journaux télécharge régulièrement de nouveaux journaux. Par défaut, le composant du gestionnaire de journaux télécharge de nouveaux journaux toutes les 5 minutes. Vous pouvez configurer un intervalle de téléchargement plus court, afin que le composant

du gestionnaire de journaux télécharge les CloudWatch journaux vers Logs plus fréquemment en configurant `leperiodicUploadIntervalSec`. Pour plus d'informations sur la configuration de cet intervalle périodique, consultez [la section Configuration](#).

Les journaux peuvent être téléchargés en temps quasi réel à partir du même système de fichiers Greengrass. Si vous devez observer les journaux en temps réel, pensez à utiliser les [journaux du système de fichiers](#).

Note

Si vous utilisez différents systèmes de fichiers pour y écrire des journaux, le gestionnaire de journaux revient au comportement des composants du gestionnaire de journaux dans les versions 2.2.8 et antérieures. Pour plus d'informations sur l'accès aux journaux du système de fichiers, voir [Accès aux journaux du système de fichiers](#).

- Horloge oblique

Le composant du gestionnaire de journaux utilise le processus de signature standard de Signature version 4 pour créer des demandes d'API destinées à CloudWatch Logs. Si l'heure système d'un appareil principal est désynchronisée de plus de 15 minutes, CloudWatch Logs rejette les demandes. Pour plus d'informations, consultez [Processus de signature Signature Version 4](#) dans le Références générales AWS.

Accédez aux journaux de service du système

Si vous [configurez le logiciel AWS IoT Greengrass Core en tant que service système](#), vous pouvez consulter les journaux des services système pour résoudre des problèmes, tels que l'échec du démarrage du logiciel.

Pour afficher les journaux de service du système (CLI)

1. Exécutez la commande suivante pour afficher les journaux de service du système logiciel AWS IoT Greengrass Core.

Linux or Unix (systemd)

```
sudo journalctl -u greengrass.service
```

Windows Command Prompt (CMD)

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

PowerShell

```
gc C:\greengrass\v2\logs\greengrass.wrapper.log
```

2. Sur les appareils Windows, le logiciel AWS IoT Greengrass Core crée un fichier journal distinct pour les erreurs de service système. Exécutez la commande suivante pour afficher les journaux d'erreurs du service système.

Windows Command Prompt (CMD)

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

PowerShell

```
gc C:\greengrass\v2\logs\greengrass.err.log
```

Sur les appareils Windows, vous pouvez également utiliser l'application Event Viewer pour consulter les journaux des services du système.

Pour consulter les journaux des services Windows (Observateur d'événements)

1. Ouvrez l'application Event Viewer.
2. Sélectionnez Windows Logs pour le développer.
3. Choisissez Application pour afficher les journaux des services des applications.
4. Recherchez et ouvrez les journaux d'événements dont la source est greengrass.

Activer la journalisation dans CloudWatch Logs

Vous pouvez déployer le [composant du gestionnaire de journaux](#) pour configurer un périphérique principal afin d'écrire des CloudWatch journaux dans Logs. Vous pouvez activer CloudWatch les journaux pour les journaux du logiciel AWS IoT Greengrass Core, et vous pouvez activer CloudWatch les journaux pour des composants spécifiques de Greengrass.

Note

Le rôle d'échange de jetons du périphérique principal Greengrass doit permettre au périphérique principal d'écrire dans CloudWatch Logs, comme illustré dans l'exemple de politique IAM suivant. Si vous avez [installé le logiciel AWS IoT Greengrass Core avec provisionnement automatique des ressources](#), votre appareil principal dispose de ces autorisations.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:logs:*:*:*"
    }
  ]
}
```

Pour configurer un périphérique principal afin d'écrire les journaux du logiciel AWS IoT Greengrass Core dans CloudWatch les journaux, [créez un déploiement](#) qui spécifie une mise à jour de configuration true définie pour le `aws.greengrass.LogManager` composant. `uploadToCloudWatch` AWS IoT Greengrass Les journaux du logiciel de base incluent les journaux du [noyau Greengrass](#) et des composants du [plugin](#).

```
{
  "logsUploaderConfiguration": {
    "systemLogsConfiguration": {
      "uploadToCloudWatch": "true"
    }
  }
}
```

Pour configurer un périphérique principal afin d'écrire les journaux d'un composant Greengrass dans Logs, [créez un déploiement qui spécifie une](#) mise à jour de configuration ajoutant le composant à la liste des configurations de journalisation des composants. CloudWatch Lorsque vous ajoutez un composant à cette liste, le composant du gestionnaire de journaux écrit ses journaux dans CloudWatch Logs. Les journaux des composants incluent les journaux des [composants génériques](#) et des [composants Lambda](#).

```
{
  "logsUploaderConfiguration": {
    "componentLogsConfigurationMap": {
      "com.example.HelloWorld": {

      }
    }
  }
}
```

Lorsque vous déployez le composant du gestionnaire de journaux, vous pouvez également configurer les limites d'espace disque et indiquer si le périphérique principal supprime les fichiers CloudWatch journaux après les avoir écrits dans Logs. Pour de plus amples informations, veuillez consulter [Configurer la journalisation pour AWS IoT Greengrass](#).

Configurer la journalisation pour AWS IoT Greengrass

Vous pouvez configurer les options suivantes pour personnaliser la journalisation pour les appareils principaux de Greengrass. Pour configurer ces options, [créez un déploiement qui spécifie une](#) mise à jour de configuration du noyau Greengrass ou des composants du gestionnaire de journaux.

- Écrire des journaux dans des CloudWatch journaux

Pour dépanner à distance les périphériques principaux, vous pouvez configurer les périphériques principaux pour écrire les journaux des logiciels et des composants AWS IoT Greengrass principaux dans les CloudWatch journaux. Pour ce faire, déployez et configurez le [composant du gestionnaire de journaux](#). Pour de plus amples informations, veuillez consulter [Activer la journalisation dans CloudWatch Logs](#).

- Suppression des fichiers journaux téléchargés

Pour réduire l'utilisation de l'espace disque, vous pouvez configurer les périphériques principaux pour qu'ils suppriment les fichiers journaux après les avoir écrits dans CloudWatch Logs. Pour

plus d'informations, consultez le `deleteLogFileAfterCloudUpload` paramètre du composant du gestionnaire de journaux, que vous pouvez spécifier pour les journaux du [logiciel AWS IoT Greengrass Core et les journaux des composants](#).

- Limites d'espace disque pour les journaux

Pour limiter l'utilisation de l'espace disque, vous pouvez configurer l'espace disque maximal pour chaque journal, y compris ses fichiers journaux pivotés, sur un périphérique principal. Par exemple, vous pouvez configurer l'espace disque combiné maximal pour les `greengrass.log` fichiers faisant l'objet d'une rotation. Pour plus d'informations, consultez le paramètre du composant `Greengrass nucleus` et le `logging.totalLogsSizeKB` paramètre du composant du gestionnaire de `diskSpaceLimit` journaux, que vous pouvez spécifier pour les journaux du [logiciel AWS IoT Greengrass Core et les journaux des composants](#).

- Limite de taille du fichier journal

Vous pouvez configurer la taille de fichier maximale pour chaque fichier journal. Lorsqu'un fichier journal dépasse cette limite de taille, le logiciel AWS IoT Greengrass Core crée un nouveau fichier journal. Le [composant du gestionnaire de journaux](#) version 2.28 (et versions antérieures) écrit uniquement les fichiers CloudWatch journaux pivotés dans Logs. Vous pouvez donc spécifier une limite de taille de fichier inférieure pour écrire des journaux dans Logs plus CloudWatch fréquemment. Le composant du gestionnaire de journaux version 2.3.0 (et versions ultérieures) traite et télécharge tous les journaux au lieu d'attendre leur rotation. Pour plus d'informations, consultez le [paramètre de limite de taille du fichier journal](#) (`logging.fileSizeKB`) du composant `Greengrass nucleus`.

- Niveaux de journalisation minimaux

Vous pouvez configurer le niveau de journalisation minimal que le composant `Greengrass nucleus` écrit dans les journaux du système de fichiers. Par exemple, vous pouvez spécifier des journaux de `DEBUG` niveau pour faciliter le dépannage, ou vous pouvez spécifier des journaux de `ERROR` niveau pour réduire le nombre de journaux créés par un périphérique principal. Pour plus d'informations, consultez le [paramètre de niveau logarithmique](#) (`logging.level`) du composant `Greengrass nucleus`.

Vous pouvez également configurer le niveau de journalisation minimal que le composant du gestionnaire de CloudWatch journaux écrit dans Logs. Par exemple, vous pouvez spécifier un niveau de journalisation plus élevé pour réduire les [coûts de journalisation](#). Pour plus d'informations, consultez le `minimumLogLevel` paramètre du composant du gestionnaire de

journaux, que vous pouvez spécifier pour les journaux du [logiciel AWS IoT Greengrass Core et les journaux des composants](#).

- Intervalle pour vérifier les journaux à écrire dans CloudWatch Logs

Pour augmenter ou diminuer la fréquence à laquelle le composant du gestionnaire de journaux écrit des CloudWatch journaux dans Logs, vous pouvez configurer l'intervalle entre lequel il vérifie la présence de nouveaux fichiers journaux à écrire. Par exemple, vous pouvez spécifier un intervalle plus court pour afficher les CloudWatch journaux dans Logs plus tôt que vous ne le feriez avec l'intervalle de 5 minutes par défaut. Vous pouvez spécifier un intervalle plus élevé pour réduire les coûts, car le composant du gestionnaire de journaux regroupe les fichiers journaux en moins de demandes. Pour plus d'informations, consultez le [paramètre d'intervalle de téléchargement](#) (`periodicUploadIntervalSec`) du composant du gestionnaire de journaux.

- Format du journal

Vous pouvez choisir si le logiciel AWS IoT Greengrass Core écrit les journaux au format texte ou JSON. Choisissez le format texte si vous lisez les journaux, ou choisissez le format JSON si vous utilisez une application pour lire ou analyser les journaux. Pour plus d'informations, consultez le [paramètre de format de log](#) (`logging.format`) du composant Greengrass nucleus.

- Dossier des journaux du système de fichiers local

Vous pouvez remplacer le dossier des `/greengrass/v2/logs` journaux par un autre dossier sur le périphérique principal. Pour plus d'informations, consultez le [paramètre de répertoire de sortie](#) (`logging.outputDirectory`) du composant Greengrass nucleus.

AWS CloudTrail journaux

AWS IoT Greengrass s'intègre AWS CloudTrail à un service qui fournit un enregistrement des actions entreprises par un utilisateur, un rôle ou Service AWS dans AWS IoT Greengrass. Pour de plus amples informations, veuillez consulter [Enregistrez les appels AWS IoT Greengrass V2 d'API avec AWS CloudTrail](#).

Enregistrez les appels AWS IoT Greengrass V2 d'API avec AWS CloudTrail

AWS IoT Greengrass V2 est intégré à AWS CloudTrail un service qui fournit un enregistrement des actions entreprises par un utilisateur, un rôle ou un AWS service dans AWS IoT Greengrass Version

2. CloudTrail capture tous les appels d'API AWS IoT Greengrass sous forme d'événements. Les appels capturés incluent les appels provenant de la AWS IoT Greengrass console et les appels de code vers les opérations de l' AWS IoT Greengrass API.

Si vous créez un suivi, vous pouvez activer la diffusion continue des CloudTrail événements vers un compartiment S3, y compris les événements pour AWS 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 faite AWS 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 plus d'informations CloudTrail, consultez le [guide de AWS CloudTrail l'utilisateur](#).

Rubriques

- [AWS IoT Greengrass V2 informations dans CloudTrail](#)
- [AWS IoT Greengrass événements de données dans CloudTrail](#)
- [AWS IoT Greengrass événements de gestion dans CloudTrail](#)
- [Comprendre les entrées du fichier AWS IoT Greengrass V2 journal](#)

AWS IoT Greengrass V2 informations dans CloudTrail

CloudTrail est activé sur votre compte Compte AWS lorsque vous créez le compte. Lorsqu'une activité se produit dans AWS 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 à l'aide de l'historique des CloudTrail événements](#).

Pour un enregistrement continu des événements de votre région Compte AWS, y compris des événements pour AWS IoT Greengrass, créez un parcours. Un journal permet CloudTrail de fournir des fichiers journaux à un compartiment S3. Par défaut, lorsque vous créez un parcours dans la console, celui-ci s'applique à tous les Région AWS s. Le journal enregistre les événements de toutes les régions de la AWS partition et transmet les fichiers journaux au compartiment S3 que vous spécifiez. 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 plus d'informations, 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 V2 actions sont enregistrées CloudTrail et documentées dans la [référence de l'AWS IoT Greengrass V2 API](#). Par exemple, les appels `auCreateComponentVersion`, `CreateDeployment` et les `CancelDeployment` actions 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 faite avec les informations d'identification de l'utilisateur root ou Gestion des identités et des accès AWS (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 faite par un autre AWS service.

Pour de plus amples informations, veuillez consulter l'[élément userIdentity CloudTrail](#).

AWS IoT Greengrass événements de données dans CloudTrail

[Les événements de données](#) fournissent des informations sur les opérations de ressources effectuées sur ou dans une ressource (par exemple, l'obtention d'une version de composant ou la configuration d'un déploiement). Ils sont également connus sous le nom opérations de plans de données. Les événements de données sont souvent des activités dont le volume est élevé. Par défaut, CloudTrail n'enregistre pas les événements liés aux données. L'historique des CloudTrail événements n'enregistre pas les événements liés aux données.

Des frais supplémentaires s'appliquent pour les événements de données. Pour plus d'informations sur la CloudTrail tarification, consultez la section [AWS CloudTrail Tarification](#).

Vous pouvez enregistrer les événements de données pour les types de AWS IoT Greengrass ressources à l'aide de la CloudTrail console ou AWS CLI des opérations de CloudTrail l'API. Le [tableau](#) de cette section indique les types de ressources disponibles pour AWS IoT Greengrass.

- Pour enregistrer les événements de données à l'aide de la CloudTrail console, créez un [magasin de données de suivi ou d'événement](#) pour enregistrer les événements, ou [mettez à jour un magasin de données de suivi ou d'événement existant](#) pour enregistrer les événements de données.
 1. Choisissez Data events pour enregistrer les événements liés aux données.
 2. Dans la liste des types d'événements de données, choisissez le type de ressource pour lequel vous souhaitez enregistrer les événements de données.
 3. Choisissez le modèle de sélecteur de journal que vous souhaitez utiliser. Vous pouvez enregistrer tous les événements de données pour le type de ressource, consigner tous les `readOnly` événements, consigner tous les `writeOnly` événements ou créer un modèle de sélecteur de journal personnalisé pour filtrer les `resources.ARN` champs `readOnlyeventName`, et.
- Pour enregistrer des événements de données à l'aide de AWS CLI, configurez le `--advanced-event-selectors` paramètre pour définir le `eventCategory` champ égal à la valeur du type de ressource Data et le `resources.type` champ égal à la valeur du type de ressource (voir le [tableau](#)). Vous pouvez ajouter des conditions pour filtrer les valeurs des `resources.ARN` champs `readOnlyeventName`, et.
- Pour configurer un journal afin de consigner les événements liés aux données, exécutez le [put-event-selectors](#) commande. Pour plus d'informations, consultez la section [Enregistrement des événements de données pour les sentiers avec le AWS CLI](#).
- Pour configurer un magasin de données d'événements afin de consigner les événements de données, exécutez [create-event-data-store](#) commande pour créer un nouveau magasin de données d'événements afin de consigner les événements de données, ou pour exécuter le [update-event-data-store](#) commande pour mettre à jour un magasin de données d'événements existant. Pour plus d'informations, consultez la section [Enregistrement des événements de données pour les magasins de données d'événements avec le AWS CLI](#).

Le tableau suivant répertorie les types de AWS IoT Greengrass ressources. La colonne Type d'événement de données (console) indique la valeur à choisir dans la liste des types d'événements de données de la CloudTrail console. La colonne de valeur `resources.type` indique la **`resources.type`** valeur que vous devez spécifier lors de la configuration de sélecteurs d'événements avancés à l'aide du ou. AWS CLI CloudTrail APIs La CloudTrail colonne Données APIs enregistrées indique les appels d'API enregistrés CloudTrail pour le type de ressource.

Type d'événement de données (console)	valeur <code>resources.type</code>	Données APIs enregistrées sur CloudTrail
Certificat IoT	<code>AWS::IoT::Certificate</code>	• <code>VerifyClientDeviceIdentity</code> • <code>VerifyClientDeviceIoTCertificateAssociation</code>
Version du composant IoT Greengrass	<code>AWS::GreengrassV2::ComponentVersion</code>	• ResolveComponentCandidates
Déploiement de Greengrass pour l'IoT	<code>AWS::GreengrassV2::Deployment</code>	• <code>GetDeploymentConfiguration</code>
Un truc lié à l'IoT	<code>AWS::IoT::Thing</code>	• <code>ListThingGroupsForCoreDevices</code> • <code>PutCertificateAuthorities</code> • <code>VerifyClientDeviceIoTCertificateAssociation</code>

Vous pouvez configurer des sélecteurs d'événements avancés pour filtrer les champs `eventName`, `readOnly` et `resources.ARN` afin de ne journaliser que les événements importants pour vous.

Ajoutez un filtre `eventName` pour inclure ou exclure des données spécifiques APIs.

Pour plus d'informations sur ces champs, voir [AdvancedFieldSelector](#).

Les exemples suivants montrent comment configurer des sélecteurs avancés à l'aide du AWS CLI. Remplacez *TrailName* et *region* par vos propres informations.

Exemple— Enregistrez les événements liés aux données pour les objets IoT

```
aws cloudtrail put-event-selectors --trail-name TrailName --region region \
--advanced-event-selectors \
'[
  {
    "Name": "Log all thing data events",
    "FieldSelectors": [
      { "Field": "eventCategory", "Equals": ["Data"] },
      { "Field": "resources.type", "Equals": ["AWS::IoT::Thing"] }
```



```

    ]
  }
]'
```

Exemple— Filtrer sur une API spécifique à un objet IoT

```
aws cloudtrail put-event-selectors --trail-name TrailName --region region \
--advanced-event-selectors \
'[
  {
    "Name": "Log IoT Greengrass PutCertificateAuthorities API calls",
    "FieldSelectors": [
      { "Field": "eventCategory", "Equals": ["Data"] },
      { "Field": "resources.type", "Equals": ["AWS::IoT::Thing"] },
      { "Field": "eventName", "Equals": ["PutCertificateAuthorities"] }
    ]
  }
]'
```

Exemple— Enregistrez tous les événements liés aux données de Greengrass

```
aws cloudtrail put-event-selectors --trail-name TrailName --region region \
--advanced-event-selectors \
'[
  {
    "Name": "Log all certificate data events",
    "FieldSelectors": [
      {
        "Field": "eventCategory",
        "Equals": [
          "Data"
        ]
      },
      {
        "Field": "resources.type",
        "Equals": [
          "AWS::IoT::Certificate"
        ]
      }
    ]
  },
  {
    "Name": "Log all component version data events",
```

```

    "FieldSelectors": [
      {
        "Field": "eventCategory",
        "Equals": [
          "Data"
        ]
      },
      {
        "Field": "resources.type",
        "Equals": [
          "AWS::GreengrassV2::ComponentVersion"
        ]
      }
    ]
  },
  {
    "Name": "Log all deployment version",
    "FieldSelectors": [
      {
        "Field": "eventCategory",
        "Equals": [
          "Data"
        ]
      },
      {
        "Field": "resources.type",
        "Equals": [
          "AWS::GreengrassV2::Deployment"
        ]
      }
    ]
  },
  {
    "Name": "Log all thing data events",
    "FieldSelectors": [
      {
        "Field": "eventCategory",
        "Equals": [
          "Data"
        ]
      },
      {
        "Field": "resources.type",
        "Equals": [

```

```
    ]',
    },
  ],
},
]
"::AWS::IoT::Thing"
```

AWS IoT Greengrass événements de gestion dans CloudTrail

[Les événements de gestion](#) fournissent des informations sur les opérations de gestion effectuées sur les ressources de votre AWS compte. Ils sont également connus sous le nom opérations de plan de contrôle. Par défaut, CloudTrail enregistre les événements de gestion.

AWS IoT Greengrass enregistre toutes les opérations AWS IoT Greengrass du plan de contrôle en tant qu'événements de gestion. Pour obtenir la liste des opérations du plan de AWS IoT Greengrass contrôle auxquelles AWS IoT Greengrass se connecte CloudTrail, consultez la [référence de l'AWS IoT Greengrass API, version 2](#).

Comprendre les entrées du fichier AWS IoT Greengrass V2 journal

Un suivi est une configuration qui permet de transmettre des événements sous forme de fichiers journaux à un compartiment S3 que vous spécifiez. CloudTrail les fichiers journaux contiennent une ou plusieurs entrées de journal. Un événement représente une demande individuelle d'une source quelconque. Il 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'CreateDeployment action.

```
{
  "eventVersion": "1.08",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "AIDACKCEVSQ6C2EXAMPLE",
    "arn": "arn:aws:iam::123456789012:user/Administrator",
    "accountId": "123456789012",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "userName": "Administrator"
  },
  "eventTime": "2021-01-06T02:38:05Z",
  "eventSource": "greengrass.amazonaws.com",
}
```

```

    "eventName": "CreateDeployment",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "203.0.113.0",
    "userAgent": "aws-cli/2.1.9 Python/3.7.9 Windows/10 exe/AMD64 prompt/off command/
greengrassv2.create-deployment",
    "requestParameters": {
        "deploymentPolicies": {
            "failureHandlingPolicy": "DO_NOTHING",
            "componentUpdatePolicy": {
                "timeoutInSeconds": 60,
                "action": "NOTIFY_COMPONENTS"
            },
            "configurationValidationPolicy": {
                "timeoutInSeconds": 60
            }
        },
        "deploymentName": "Deployment for MyGreengrassCoreGroup",
        "components": {
            "aws.greengrass.Cli": {
                "componentVersion": "2.0.3"
            }
        },
        "iotJobConfiguration": {},
        "targetArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/
MyGreengrassCoreGroup"
    },
    "responseElements": {
        "iotJobArn": "arn:aws:iot:us-west-2:123456789012:job/fdfeba1d-ac6d-44ef-
ab28-54f684ea578d",
        "iotJobId": "fdfeba1d-ac6d-44ef-ab28-54f684ea578d",
        "deploymentId": "4196dddc-0a21-4c54-a985-66a525f6946e"
    },
    "requestID": "311b9529-4aad-42ac-8408-c06c6fec79a9",
    "eventID": "c0f3aa2c-af22-48c1-8161-bad4a2ab1841",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "eventCategory": "Management",
    "recipientAccountId": "123456789012"
}

```

Collectez les données de télémétrie relatives à l'état du système à partir des principaux appareils AWS IoT Greengrass

Les données de télémétrie relatives à l'état du système sont des données de diagnostic qui peuvent vous aider à surveiller les performances des opérations critiques sur vos principaux appareils Greengrass. Vous pouvez créer des projets et des applications pour récupérer, analyser, transformer et rapporter des données de télémétrie à partir de vos appareils périphériques. Les experts du domaine, tels que les ingénieurs de processus, peuvent utiliser ces applications pour obtenir des informations sur l'état de la flotte.

Vous pouvez utiliser les méthodes suivantes pour collecter des données de télémétrie à partir de vos principaux appareils Greengrass :

- Composant émetteur de télémétrie Nucleus : le composant émetteur de [télémétrie Nucleus](#) () d'un appareil central Greengrass publie les données de télémétrie `aws.greengrass.telemetry.NucleusEmitter` sur le sujet par défaut. `$local/greengrass/telemetry` Vous pouvez utiliser les données publiées dans cette rubrique pour agir localement sur votre appareil principal, même lorsque la connectivité de celui-ci au cloud est limitée. Facultativement, vous pouvez également configurer le composant pour publier des données de télémétrie sur un sujet AWS IoT Core MQTT de votre choix.

Vous devez déployer le composant Nucleus Emitter sur un périphérique principal pour publier les données de télémétrie. La publication de données de télémétrie sur le sujet local n'entraîne aucun coût. Toutefois, l'utilisation d'une rubrique MQTT pour publier des données sur le AWS Cloud est soumise à une [AWS IoT Core tarification](#).

AWS IoT Greengrass fournit plusieurs [composants communautaires](#) pour vous aider à analyser et à visualiser les données de télémétrie localement sur votre appareil principal à l'aide d'InfluxDB et Grafana. Ces composants utilisent les données de télémétrie du composant émetteur du noyau. Pour plus d'informations, consultez le fichier README du composant éditeur [InfluxDB](#).

- Agent de télémétrie : l'agent de télémétrie installé sur les appareils principaux de Greengrass collecte des données de télémétrie locales et les publie sur Amazon sans aucune interaction avec le client. EventBridge 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.

La fonction d'agent de télémétrie est activée par défaut pour tous les appareils principaux de Greengrass. Vous commencez automatiquement à recevoir des données dès que vous configurez

un appareil principal Greengrass. Outre les frais de liaison de données, le transfert de données de l'appareil principal vers le périphérique AWS IoT Core est gratuit. Cela est dû au fait que l'agent publie sur un sujet AWS réservé. Toutefois, en fonction de votre cas d'utilisation, vous pouvez encourir des frais lorsque vous recevez ou traitez les données.

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. Pour plus d'informations, consultez [Qu'est-ce qu'Amazon EventBridge ?](#) dans le guide de EventBridge l'utilisateur Amazon.

Pour garantir le bon fonctionnement du logiciel de AWS IoT Greengrass base, AWS IoT Greengrass utilise les données à des fins de développement et d'amélioration de la qualité. Cette fonctionnalité contribue également à définir de nouvelles fonctionnalités de pointe améliorées. AWS IoT Greengrass conserve les données de télémétrie pendant sept jours au maximum.

Cette section décrit comment configurer et utiliser l'agent de télémétrie. Pour plus d'informations sur la configuration du composant émetteur de télémétrie Nucleus, consultez [Émetteur de télémétrie Nucleus](#)

Rubriques

- [Métriques de télémétrie](#)
- [Configuration des paramètres de l'agent de télémétrie](#)
- [Abonnez-vous aux données de télémétrie dans EventBridge](#)

Métriques de télémétrie

Le tableau suivant décrit les métriques publiées par l'agent de télémétrie.

Name (Nom)	Description	
Système		
SystemMemUsage	La quantité de mémoire actuellement utilisée par	

Name (Nom)	Description	
	toutes les applications du périphérique principal de Greengrass, y compris le système d'exploitation.	
CpuUsage	La quantité de processeur actuellement utilisée par toutes les applications du périphérique principal de Greengrass, y compris le système d'exploitation.	
TotalNumberOfFDs	Nombre de descripteurs de fichiers stockés par le système d'exploitation du périphérique principal Greengrass. Un descripteur de fichier identifie de manière unique un fichier ouvert.	
Noyau de Greengrass		
NumberOfComponentsRunning	Le nombre de composants qui s'exécutent sur le périphérique principal de Greengrass.	
NumberOfComponentsErrored	Nombre de composants présentant un état d'erreur sur le périphérique principal de Greengrass.	
NumberOfComponentsInstalled	Le nombre de composants installés sur le périphérique principal de Greengrass.	

Name (Nom)	Description	
NumberOfComponentsStarting	Le nombre de composants qui démarrent sur le périphérique principal de Greengrass.	
NumberOfComponentsNew	Le nombre de composants nouveaux sur le périphérique principal de Greengrass.	
NumberOfComponentsStopping	Le nombre de composants qui s'arrêtent sur le périphérique principal de Greengrass.	
NumberOfComponentsFinished	Le nombre de composants terminés sur le périphérique principal Greengrass.	
NumberOfComponentsBroken	Le nombre de composants cassés sur le périphérique principal de Greengrass.	
NumberOfComponentsStateless	Le nombre de composants qui sont apatrides sur le périphérique principal de Greengrass.	
Authentification du périphérique client : cette fonctionnalité nécessite la version 2.4.0 ou ultérieure du composant d'authentification du périphérique client.		
VerifyClientDeviceIdentity.Success	Nombre de fois que l'identité de l'appareil client a été vérifiée avec succès.	

Name (Nom)	Description	
VerifyClientDeviceIdentity.Failure	Nombre de fois où la vérification de l'identité de l'appareil client a échoué.	
AuthorizeClientDeviceActions.Success	Le nombre de fois où le dispositif client est autorisé à effectuer les actions demandées.	
AuthorizeClientDeviceActions.Failure	Le nombre de fois où l'appareil client n'est pas autorisé à effectuer les actions demandées.	
GetClientDeviceAuthToken.Success	Le nombre de fois que l'appareil client est authentifié avec succès.	
GetClientDeviceAuthToken.Failure	Le nombre de fois où l'appareil client ne peut pas être authentifié.	
SubscribeToCertificateUpdates.Success	Le nombre d'abonnements réussis aux mises à jour de certificats.	
SubscribeToCertificateUpdates.Failure	Nombre de tentatives infructueuses de souscription aux mises à jour des certificats.	
ServiceError	Nombre d'erreurs internes non gérées sur l'authentification de l'appareil client.	

Name (Nom)	Description	
Gestionnaire de flux	Cette fonctionnalité nécessite la version 2.7.0 ou ultérieure du composant Greengrass Nucleus.	
BytesAppended	Nombre d'octets de données ajoutés au gestionnaire de flux.	
BytesUploadedToIoTAnalytics	Le nombre d'octets de données que le gestionnaire de flux exporte vers les canaux vers lesquels AWS IoT Analytics.	
BytesUploadedToKinesis	Le nombre d'octets de données que le gestionnaire de flux exporte vers des flux dans Amazon Kinesis Data Streams.	
BytesUploadedToIoTSiteWise	Le nombre d'octets de données que le gestionnaire de flux exporte vers les propriétés des actifs dans AWS IoT SiteWise.	
BytesUploadedToS3	Le nombre d'octets de données que le gestionnaire de flux exporte vers des objets dans Amazon S3.	

Name (Nom)	Description	
System Metrics — Cette fonctionnalité nécessite la version 2.15.0 ou ultérieure du composant Greengrass Nucleus.		
CPUArchitecture	Architecture de l'unité centrale de l'appareil.	
Family	Famille de systèmes d'exploitation de l'appareil (Windows uniquement).	
KernelVersion	Version du noyau de l'appareil (Unix uniquement).	
KnowledgeBaseArticles	Articles de la base de connaissances installés sur un appareil (Windows uniquement).	
OSBuildMajor	Numéro de version principal de la version du système d'exploitation (Windows uniquement).	
OSBuildMinor	Numéro de version mineur de la version du système d'exploitation (Windows uniquement).	
OSName	Nom du système d'exploitation de l'appareil.	
OSVersion	Version marketing du système d'exploitation de l'appareil.	

Configuration des paramètres de l'agent de télémétrie

L'agent de télémétrie utilise les paramètres par défaut 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.

L'agent de télémétrie publie les données à l'aide du protocole MQTT avec un niveau de qualité de service (QoS) de 0, ce qui signifie qu'il ne confirme pas la livraison ni ne réessaie aucune tentative de publication. Les messages de télémétrie partagent une connexion MQTT avec d'autres messages pour les abonnements destinés à AWS IoT Core.

Outre les coûts de liaison de données, le transfert de données du cœur vers le AWS IoT Core réseau est gratuit. Cela est dû au fait que l'agent publie sur un sujet AWS réservé. Toutefois, en fonction de votre cas d'utilisation, vous pouvez encourir des frais lorsque vous recevez ou traitez les données.

Vous pouvez activer ou désactiver la fonction d'agent de télémétrie pour chaque appareil principal de Greengrass. Vous pouvez également configurer les intervalles pendant lesquels le périphérique principal agrège et publie les données. [Pour configurer la télémétrie, personnalisez le paramètre de configuration de télémétrie lorsque vous déployez le composant Greengrass nucleus.](#)

Abonnez-vous aux données de télémétrie dans EventBridge

Vous pouvez créer des règles dans Amazon EventBridge qui définissent le mode de traitement des données de télémétrie publiées par l'agent de télémétrie sur l'appareil principal de Greengrass. Lorsqu'il EventBridge reçoit les données, il invoque les actions cibles définies dans vos règles. Par exemple, vous pouvez créer des règles d'événements qui envoient des notifications, stockent des informations sur les événements, prennent des mesures correctives ou invoquent d'autres événements.

Événements de télémétrie

Les événements de télémétrie utilisent le format suivant.

```
{
  "version": "0",
  "id": "a09d303e-2f6e-3d3c-a693-8e33f4fe3955",
  "detail-type": "Greengrass Telemetry Data",
  "source": "aws.greengrass",
  "account": "123456789012",
```

```
"time": "2020-11-30T20:45:53Z",
"region": "us-east-1",
"resources": [],
"detail": {
  "ThingName": "MyGreengrassCore",
  "Schema": "2020-07-30",
  "ADP": [
    {
      "TS": 1602186483234,
      "NS": "SystemMetrics",
      "M": [
        {
          "N": "TotalNumberOfFDs",
          "Sum": 6447.0,
          "U": "Count"
        },
        {
          "N": "CpuUsage",
          "Sum": 15.458333333333332,
          "U": "Percent"
        },
        {
          "N": "SystemMemUsage",
          "Sum": 10201.0,
          "U": "Megabytes"
        }
      ]
    },
    {
      "TS": 1602186483234,
      "NS": "GreengrassComponents",
      "M": [
        {
          "N": "NumberOfComponentsStopping",
          "Sum": 0.0,
          "U": "Count"
        },
        {
          "N": "NumberOfComponentsStarting",
          "Sum": 0.0,
          "U": "Count"
        },
        {
          "N": "NumberOfComponentsBroken",
```

```
        "Sum": 0.0,
        "U": "Count"
    },
    {
        "N": "NumberOfComponentsFinished",
        "Sum": 1.0,
        "U": "Count"
    },
    {
        "N": "NumberOfComponentsInstalled",
        "Sum": 0.0,
        "U": "Count"
    },
    {
        "N": "NumberOfComponentsRunning",
        "Sum": 7.0,
        "U": "Count"
    },
    {
        "N": "NumberOfComponentsNew",
        "Sum": 0.0,
        "U": "Count"
    },
    {
        "N": "NumberOfComponentsErrored",
        "Sum": 0.0,
        "U": "Count"
    },
    {
        "N": "NumberOfComponentsStateless",
        "Sum": 0.0,
        "U": "Count"
    }
]
},
{
    "TS": 1602186483234,
    "NS": "aws.greengrass.ClientDeviceAuth",
    "M": [
        {
            "N": "VerifyClientDeviceIdentity.Success",
            "Sum": 3.0,
            "U": "Count"
        }
    ],
}
```

```
{
  "N": "VerifyClientDeviceIdentity.Failure",
  "Sum": 1.0,
  "U": "Count"
},
{
  "N": "AuthorizeClientDeviceActions.Success",
  "Sum": 20.0,
  "U": "Count"
},
{
  "N": "AuthorizeClientDeviceActions.Failure",
  "Sum": 5.0,
  "U": "Count"
},
{
  "N": "GetClientDeviceAuthToken.Success",
  "Sum": 5.0,
  "U": "Count"
},
{
  "N": "GetClientDeviceAuthToken.Failure",
  "Sum": 2.0,
  "U": "Count"
},
{
  "N": "SubscribeToCertificateUpdates.Success",
  "Sum": 10.0,
  "U": "Count"
},
{
  "N": "SubscribeToCertificateUpdates.Failure",
  "Sum": 1.0,
  "U": "Count"
},
{
  "N": "ServiceError",
  "Sum": 3.0,
  "U": "Count"
}
],
{
  "TS": 1602186483234,
```

```
"NS": "aws.greengrass.StreamManager",
"M": [
  {
    "N": "BytesAppended",
    "Sum": 157745524.0,
    "U": "Bytes"
  },
  {
    "N": "BytesUploadedToIoTAnalytics",
    "Sum": 149012.0,
    "U": "Bytes"
  },
  {
    "N": "BytesUploadedToKinesis",
    "Sum": 12192.0,
    "U": "Bytes"
  },
  {
    "N": "BytesUploadedToIoTSiteWise",
    "Sum": 13321.0,
    "U": "Bytes"
  },
  {
    "N": "BytesUploadedToS3",
    "Sum": 12213.0,
    "U": "Bytes"
  }
]
}
{
  "TS": 1750104449426,
  "NS": "SystemMetrics",
  "M": [
    {
      "N": "KernelVersion",
      "Sum": 1,
      "U": "6.1.140-154.222.amzn2023.x86_64"
    },
    {
      "N": "OSVersion",
      "Sum": 1,
      "U": "2023.7.20250609"
    },
    {
```



```
        "N": "OSName",
        "Sum": 1,
        "U": "Amazon Linux"
    },
    {
        "N": "CPUArchitecture",
        "Sum": 1,
        "U": "Broadwell (Server)"
    },
]
}
```

Le ADP tableau contient une liste de points de données agrégés possédant les propriétés suivantes :

TS

Horodatage de la collecte des données.

NS

L'espace de noms de la métrique.

M

La liste des métriques. Une métrique contient les propriétés suivantes :

N

Le nom de la métrique.

Sum

Somme des valeurs de la métrique dans cet événement de télémétrie.

U

Unité de la valeur métrique.

Pour plus d'informations sur chaque métrique, consultez [Métriques de télémétrie](#).

Conditions préalables à la création de règles EventBridge

Avant de créer une EventBridge règle pour AWS IoT Greengrass, vous devez effectuer les opérations suivantes :

- Familiarisez-vous avec les événements, les règles et les cibles dans EventBridge.
- Créez et configurez les [cibles](#) invoquées par vos EventBridge règles. Les règles peuvent invoquer de nombreux types de cibles, telles que les flux Amazon Kinesis, les AWS Lambda fonctions, les rubriques Amazon SNS et les files d'attente Amazon SQS.

Votre EventBridge règle et les cibles associées doivent se trouver dans l' Région AWS endroit où vous avez créé vos ressources Greengrass. Pour plus d'informations, consultez la section [Points de terminaison et quotas du service](#) dans le Références générales AWS.

Pour plus d'informations, consultez [Qu'est-ce qu'Amazon EventBridge ?](#) et [Getting started with 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 ci-dessous pour AWS Management Console créer une EventBridge règle qui reçoit les données de télémétrie publiées par le périphérique principal de 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 déclenchant un événement à partir d'une AWS ressource](#) dans le guide de EventBridge l'utilisateur Amazon.

1. Ouvrez la [EventBridgeconsole Amazon](#), puis choisissez Create rule.
2. Sous Name and description (Nom et description), entrez un nom et une description pour la règle.
3. Sous Define pattern (Définir un modèle), configurez le modèle de règle.
 - a. Choisissez Event pattern.
 - b. Choisissez Pré-defined pattern by service (Modèle prédéfini par service).
 - c. Pour Service Provider (Fournisseur de service), sélectionnez AWS.
 - d. Dans Service name (Nom du service), choisissez Greengrass.
 - e. Pour Type d'événement, sélectionnez Greengrass Telemetry Data.
4. Sous Select event bus (Sélectionner un bus d'événements), conservez les options de bus d'événements par défaut.

5. Sous **Select targets** (Sélectionner les cibles), configurez votre cible. L'exemple suivant utilise une file d'attente Amazon SQS, mais vous pouvez configurer d'autres types de cibles.
 - a. Pour **Target**, choisissez la file d'attente SQS.
 - b. Pour **Queue***, choisissez votre file d'attente cible.
6. Sous **Tags - optional** (Balises - facultatif), définissez les balises pour la règle ou laissez les champs vides.
7. Choisissez **Créer**.

Création d'une règle d'événement pour obtenir des données de télémétrie (CLI)

Suivez les étapes ci-dessous pour AWS CLI créer une EventBridge règle qui reçoit les données de télémétrie publiées par les appareils principaux de 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 .
 - *thing-name* Remplacez-le par le nom de l'objet du périphérique principal.

Linux or Unix

```
aws events put-rule \
  --name MyGreengrassTelemetryEventRule \
  --event-pattern "{\"source\": [\"aws.greengrass\"], \"detail\": {\"ThingName\": [\"thing-name\"]}}"
```

Windows Command Prompt (CMD)

```
aws events put-rule ^
  --name MyGreengrassTelemetryEventRule ^
  --event-pattern "{\"source\": [\"aws.greengrass\"], \"detail\": {\"ThingName\": [\"thing-name\"]}}"
```

PowerShell

```
aws events put-rule `
  --name MyGreengrassTelemetryEventRule `
```

```
--event-pattern "{\"source\": [\"aws.greengrass\"], \"detail\": {\"ThingName\": [\"thing-name\"]}}"
```

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.

- *queue-arn* Remplacez-le par l'ARN de votre file d'attente Amazon SQS.

Linux or Unix

```
aws events put-targets \  
  --rule MyGreengrassTelemetryEventRule \  
  --targets "Id"="1", "Arn"="queue-arn"
```

Windows Command Prompt (CMD)

```
aws events put-targets ^  
  --rule MyGreengrassTelemetryEventRule ^  
  --targets "Id"="1", "Arn"="queue-arn"
```

PowerShell

```
aws events put-targets `  
  --rule MyGreengrassTelemetryEventRule `  
  --targets "Id"="1", "Arn"="queue-arn"
```

Note

Pour permettre EventBridge à Amazon d'appeler votre file d'attente cible, vous devez ajouter une politique basée sur les ressources à votre sujet. 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.

Recevez des notifications relatives au déploiement et à l'état de santé des composants

Les règles relatives aux EventBridge événements d'Amazon vous fournissent des notifications concernant les changements d'état de vos déploiements Greengrass reçus par vos appareils et des composants installés sur votre appareil. EventBridge fournit un flux d'événements système en temps quasi réel qui décrit l'évolution des AWS ressources. AWS IoT Greengrass envoie ces événements dans EventBridge la mesure du possible. Cela signifie que AWS IoT Greengrass tous les événements sont envoyés à EventBridge mais, dans de rares cas, un événement peut ne pas être diffusé. En outre, AWS IoT Greengrass cela peut envoyer plusieurs copies d'un événement donné, ce qui signifie que les auditeurs de vos événements risquent de ne pas recevoir les événements dans l'ordre dans lequel ils se sont produits.

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 et de composants. Pour plus d'informations, consultez [Qu'est-ce qu'Amazon EventBridge ?](#) dans le guide de EventBridge l'utilisateur Amazon.

Rubriques

- [Événement de modification de l'état du déploiement](#)
- [Événement de changement d'état du composant](#)
- [Conditions préalables à la création de règles EventBridge](#)
- [Configuration des notifications relatives à l'état de santé de l'appareil \(console\)](#)
- [Configuration des notifications d'état de l'appareil \(CLI\)](#)
- [Configurer les notifications relatives à l'état de l'appareil \(CloudFormation\)](#)
- [Consultez aussi](#)

Événement de modification de l'état du déploiement

AWS IoT Greengrass émet un événement lorsqu'un déploiement entre dans les états suivants : `FAILED`, `SUCCEEDED`, `COMPLETED`, `REJECTED`, et `CANCELED`. Vous pouvez créer une EventBridge règle qui s'applique à toutes les transitions d'état ou aux transitions vers les états que vous spécifiez. Lorsqu'un déploiement entre dans un état qui initie une règle, il EventBridge invoque 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 :

- Lancez des opérations après le déploiement, telles que le téléchargement de ressources et la notification du personnel.
- Envoyer des notifications en cas de réussite ou de l'échec d'un déploiement.
- Publier des métriques personnalisées sur les événements de déploiement.

L' [événement](#) pour un changement d'état de déploiement utilise le format suivant :

```
{
  "version": "0",
  "id": "cd4d811e-ab12-322b-8255-EXAMPLEb1bc8",
  "detail-type": "Greengrass V2 Effective Deployment Status Change",
  "source": "aws.greengrass",
  "account": "123456789012",
  "region": "us-west-2",
  "time": "2018-03-22T00:38:11Z",
  "resources": ["arn:aws:greengrass:us-east-1:123456789012:coreDevices:MyGreengrassCore"],
  "detail": {
    "deploymentId": "4f38f1a7-3dd0-42a1-af48-EXAMPLE09681",
    "coreDeviceExecutionStatus": "FAILED|SUCCEEDED|COMPLETED|REJECTED|CANCELED",
    "statusDetails": {
      "errorStack": ["DEPLOYMENT_FAILURE", "ARTIFACT_DOWNLOAD_ERROR", "S3_ERROR", "S3_ACCESS_DENIED", "S3_HEAD_OBJECT_ACCESS_DENIED"],
      "errorTypes": ["DEPENDENCY_ERROR", "PERMISSION_ERROR"],
    },
    "reason": "S3_HEAD_OBJECT_ACCESS_DENIED: FAILED_NO_STATE_CHANGE: Failed to download artifact name: 's3://pentest27/nucleus/281/aws.greengrass.nucleus.zip' for component aws.greengrass.Nucleus-2.8.1, reason: S3 HeadObject returns 403 Access Denied. Ensure the IAM role associated with the core device has a policy granting
```

```
s3:GetObject. null (Service: S3, Status Code: 403, Request ID: HR94ZNT2161DAR58,
Extended Request ID: wTX4DDI+qigQt3uzwl9rlnQiYlBgwvPm/KJFWeFAn9t1mnGXTms/
1uLCYANGq08RIH+x2H+hEKc=)"
}
```

Vous pouvez créer des règles et des événements qui vous informeront de l'état d'un déploiement. Un événement est déclenché lorsqu'un déploiement se termine sous la forme `FAILED`, `SUCCEEDED`, `COMPLETED`, `REJECTED`, ou `CANCELED`. Si le déploiement a échoué sur le périphérique principal, vous recevrez une réponse détaillée expliquant pourquoi le déploiement a échoué. Pour plus d'informations sur les codes d'erreur de déploiement, consultez [Codes d'erreur de déploiement détaillés](#).

États de déploiement

- `FAILED`. Le déploiement a échoué.
- `SUCCEEDED`. Le déploiement ciblé sur un groupe d'objets s'est terminé avec succès.
- `COMPLETED`. Le déploiement visait un objet terminé avec succès.
- `REJECTED`. Le déploiement a été refusé. Pour plus d'informations, consultez le `statusDetails` champ.
- `CANCELED`. Le déploiement a été annulé par l'utilisateur.

Il est possible que les événements soient dupliqués ou hors service. Pour déterminer l'ordre des événements, utilisez la propriété `time`.

Pour obtenir la liste complète des codes d'erreur contenus dans `errorStacks` et `errorTypes`, reportez-vous [Codes d'erreur de déploiement détaillés](#) aux sections et [Codes d'état détaillés des composants](#).

Événement de changement d'état du composant

Pour AWS IoT Greengrass les versions 2.12.2 et antérieures, Greengrass émet un événement lorsqu'un composant entre dans les états suivants : `et`, `ERROR`, `BROKEN`. Pour les versions 2.12.3 et ultérieures de Greengrass nucleus, Greengrass émet un événement lorsqu'un composant entre dans les états suivants : `,`, `et`, `ERROR`, `BROKEN`, `RUNNING`, `FINISHED`. Greengrass émettra également un événement lorsqu'un déploiement sera terminé. Vous pouvez créer une EventBridge règle qui s'applique à toutes les transitions d'état ou aux transitions vers les états que vous spécifiez. Lorsqu'un composant installé entre dans un état qui initie une règle, il EventBridge invoque 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.

L'[événement](#) de modification de l'état d'un composant utilise les formats suivants :

Greengrass nucleus v2.12.2 and earlier

<title>État du composant : ERRORED ou BROKEN</title>

```
{
  "version": "0",
  "id": " cd4d811e-ab12-322b-8255-EXAMPLEb1bc8",
  "detail-type": "Greengrass V2 Installed Component Status Change",
  "source": "aws.greengrass",
  "account": "123456789012",
  "region": "us-west-2",
  "time": "2018-03-22T00:38:11Z",
  "resources": ["arn:aws:greengrass:us-east-1:123456789012:coreDevices:MyGreengrassCore"],
  "detail": {
    "components": [
      {
        "componentName": "MyComponent",
        "componentVersion": "1.0.0",
        "root": true,
        "lifecycleState": "ERRORED|BROKEN",
        "lifecycleStatusCodes": ["STARTUP_ERROR"],
        "lifecycleStateDetails": "An error occurred during startup. The startup script exited with code 1."
      }
    ]
  }
}
```

Greengrass nucleus v2.12.3 and later

<title>État du composant : ERRORED ou BROKEN</title>

```
{
  "version": "0",
  "id": " cd4d811e-ab12-322b-8255-EXAMPLEb1bc8",
  "detail-type": "Greengrass V2 Installed Component Status Change",
```



```

    "source": "aws.greengrass",
    "account": "123456789012",
    "region": "us-west-2",
    "time": "2018-03-22T00:38:11Z",
    "resources": ["arn:aws:greengrass:us-east-1:123456789012:coreDevices:MyGreengrassCore"],
    "detail": {
      "components": [
        {
          "componentName": "MyComponent",
          "componentVersion": "1.0.0",
          "root": true,
          "lifecycleState": "ERRORED|BROKEN",
          "lifecycleStatusCodes": ["STARTUP_ERROR"],
          "lifecycleStateDetails": "An error occurred during startup. The startup script exited with code 1."
        }
      ]
    }
  }
}

```

<title>État du composant : RUNNING ou FINISHED</title>

```

{
  "version": "0",
  "id": " cd4d811e-ab12-322b-8255-EXAMPLEb1bc8",
  "detail-type": "Greengrass V2 Installed Component Status Change",
  "source": "aws.greengrass",
  "account": "123456789012",
  "region": "us-west-2",
  "time": "2018-03-22T00:38:11Z",
  "resources": ["arn:aws:greengrass:us-east-1:123456789012:coreDevices:MyGreengrassCore"],
  "detail": {
    "components": [
      {
        "componentName": "MyComponent",
        "componentVersion": "1.0.0",
        "root": true,
        "lifecycleState": "RUNNING|FINISHED",
        "lifecycleStateDetails": null
      }
    ]
  }
}

```

```
}  
}
```

Vous pouvez créer des règles et des événements qui vous informeront de l'état d'un composant installé. Un événement est déclenché lorsqu'un composant change d'état sur le périphérique. Vous recevrez une réponse détaillée expliquant pourquoi un composant est défectueux ou défectueux. Vous recevrez également un code d'état qui indiquera la raison de l'échec. Pour plus d'informations sur les codes d'état des composants, consultez [Codes d'état détaillés des composants](#).

Conditions préalables à la création de règles EventBridge

Avant de créer une EventBridge règle pour AWS IoT Greengrass, procédez comme suit :

- Familiarisez-vous avec les événements, les règles et les cibles dans EventBridge.
- Créez et configurez les cibles invoquées par vos EventBridge règles. Les règles peuvent appeler de nombreux types de cibles, notamment :
 - Amazon Simple Notification Service (Amazon SNS)
 - AWS Lambda fonctions
 - Amazon Kinesis Video Streams
 - Files d'attente Amazon Simple Queue Service (Amazon SQS)

Pour plus d'informations, consultez [Qu'est-ce qu'Amazon EventBridge ?](#) et [Getting started with Amazon EventBridge](#) dans le guide de EventBridge l'utilisateur Amazon.

Configuration des notifications relatives à l'état de santé de l'appareil (console)

Suivez les étapes ci-dessous pour créer une EventBridge règle qui publie une rubrique Amazon SNS lorsque l'état de déploiement d'un groupe change. 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 déclenchant un événement à partir d'une AWS ressource](#) dans le guide de EventBridge l'utilisateur Amazon.

1. Ouvrez la [EventBridgeconsole Amazon](#).
2. Dans le volet de navigation, choisissez Règles.
3. Choisissez Créer une règle.

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, sélectionnez Bus d'événements par défaut AWS . Lorsqu'un AWS service de votre compte émet un événement, celui-ci est toujours redirigé vers le bus d'événements par défaut de votre compte.

6. Pour Type de règle, choisissez Règle avec un modèle d'événement.

7. Choisissez Suivant.

8. Pour Event source (Source de l'événement), choisissez AWS events (Événements).

9. Dans Modèle d'événement, sélectionnez AWS services.

10. Pour le AWS service, choisissez Greengrass.

11. Pour le type d'événement, choisissez l'une des options suivantes :

- Pour les événements de déploiement, choisissez Greengrass V2 Effective Deployment Status Change.
- Pour les événements relatifs aux composants, choisissez Greengrass V2 Installed Component Status Change.

12. Choisissez Suivant.

13. Pour Types de cibles, choisissez service AWS .

14. Pour Sélectionner une cible, configurez votre cible. Cet exemple utilise une rubrique Amazon SNS, mais vous pouvez configurer d'autres types de cibles pour envoyer des notifications.

- a. Pour Target (Cible), choisissez SNS topic (Rubrique SNS).
- b. Pour Topic (Rubrique), choisissez votre rubrique cible.
- c. Choisissez Suivant.

15. Choisissez Suivant.

16. Consultez les détails de la règle et choisissez Create rule (Créer une règle).

Configuration des notifications d'état de l'appareil (CLI)

Suivez les étapes ci-dessous pour créer une EventBridge règle qui publie une rubrique Amazon SNS en cas d'événement de changement de statut de 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 .

- Pour les événements de modification de l'état du déploiement.

```
aws events put-rule \  
  --name TestRule \  
  --event-pattern "{\"source\": [\"aws.greengrass\"], \"detail-type\":  
  [\"Greengrass V2 Effective Deployment Status Change\"]}"
```

- Pour les événements de modification de l'état des composants.

```
aws events put-rule \  
  --name TestRule \  
  --event-pattern "{\"source\": [\"aws.greengrass\"], \"detail-type\":  
  [\"Greengrass V2 Installed Component Status Change\"]}"
```

Les propriétés qui sont omises dans le modèle sont ignorées.

2. Ajoutez la rubrique en tant que cible de règle.

- *topic-arn* Remplacez-le par l'ARN de votre rubrique Amazon SNS.

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

Note

Pour autoriser Amazon EventBridge à appeler votre sujet cible, vous devez ajouter une politique basée sur les ressources à votre sujet. Pour plus d'informations, consultez les [autorisations Amazon SNS](#) 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.

Configurer les notifications relatives à l'état de l'appareil (CloudFormation)

Utilisez des CloudFormation modèles pour créer des EventBridge règles qui envoient des notifications concernant les changements d'état pour les déploiements de votre groupe Greengrass. Pour plus d'informations, consultez la [référence aux types de EventBridge ressources Amazon](#) dans le guide de AWS CloudFormation l'utilisateur.

Consultez aussi

- [Vérifier l'état de déploiement de l'appareil](#)
- [Qu'est-ce qu'Amazon EventBridge ?](#) dans le guide de EventBridge l'utilisateur Amazon

Vérifiez l'état de l'appareil Greengrass Core

Les appareils Greengrass Core signalent l'état de leurs composants logiciels à AWS IoT Greengrass. Vous pouvez consulter le résumé de l'état de santé de chaque appareil et vous pouvez vérifier l'état de chaque composant sur chaque appareil.

L'état de santé des appareils principaux est le suivant :

- **HEALTHY**— Le logiciel AWS IoT Greengrass principal et tous les composants s'exécutent sans problème sur le périphérique principal.
- **UNHEALTHY**— Le logiciel AWS IoT Greengrass principal ou un composant présente un état d'erreur sur le périphérique principal.

Note

AWS IoT Greengrass s'appuie sur des appareils individuels pour envoyer des mises à jour de statut au AWS Cloud. Si le logiciel AWS IoT Greengrass Core n'est pas en cours d'exécution sur l'appareil, ou si l'appareil n'est pas connecté au AWS Cloud, l'état indiqué de cet appareil peut ne pas refléter son état actuel. L'horodatage d'état indique la date de dernière mise à jour de l'état de l'appareil.

Les appareils principaux envoient des mises à jour de statut aux heures suivantes :

- Quand le logiciel AWS IoT Greengrass Core démarre

- Lorsque le dispositif principal reçoit un déploiement du AWS Cloud
- Pour Greengrass nucleus 2.12.2 et versions antérieures, le périphérique principal envoie des mises à jour d'état lorsque l'état d'un composant du périphérique principal devient `ERROR` ou `BROKEN`
- Pour Greengrass nucleus 2.12.3 et versions ultérieures, le périphérique principal envoie des mises à jour d'état lorsque l'état de l'un des composants du périphérique principal devient `ERROR`, ou `BROKEN`, `RUNNING` ou `FINISHED`
- À un [intervalle régulier que vous pouvez configurer](#), qui est par défaut de 24 heures

Pour AWS IoT Greengrass Core v2.7.0 et versions ultérieures, le périphérique principal envoie des mises à jour d'état lors du déploiement local et du déploiement dans le cloud

Rubriques

- [Vérifier l'état de santé d'un appareil principal](#)
- [Vérifier l'état d'un groupe d'appareils principal](#)
- [Vérifier l'état des composants de l'appareil principal](#)

Vérifier l'état de santé d'un appareil principal

Vous pouvez vérifier l'état de chaque appareil principal.

Pour vérifier l'état d'un périphérique principal (AWS CLI)

- Exécutez la commande suivante pour récupérer l'état d'un appareil. Remplacez *coreDeviceName* par le nom du périphérique principal à interroger.

```
aws greengrassv2 get-core-device --core-device-thing-name coreDeviceName
```

La réponse contient des informations sur le périphérique principal, notamment son état.

Vérifier l'état d'un groupe d'appareils principal

Vous pouvez vérifier l'état d'un groupe d'appareils principaux (un groupe d'objets).

Pour vérifier l'état d'un groupe d'appareils (AWS CLI)

- Exécutez la commande suivante pour récupérer l'état de plusieurs périphériques principaux. Remplacez l'ARN de la commande par l'ARN du groupe d'objets à interroger.

```
aws greengrassv2 list-core-devices --thing-group-arn "arn:aws:iot:region:account-id:thinggroup/thingGroupName"
```

La réponse contient la liste des principaux appareils du groupe d'objets. Chaque entrée de la liste contient l'état du périphérique principal.

Vérifier l'état des composants de l'appareil principal

Vous pouvez vérifier l'état, tel que l'état du cycle de vie, des composants logiciels d'un périphérique principal. Pour plus d'informations sur l'état du cycle de vie des composants, consultez [Développer des AWS IoT Greengrass composants](#).

Pour vérifier l'état des composants d'un périphérique principal (AWS CLI)

- Exécutez la commande suivante pour récupérer l'état des composants sur un périphérique principal. Remplacez *coreDeviceName* par le nom du périphérique principal à interroger.

```
aws greengrassv2 list-installed-components --core-device-thing-name coreDeviceName
```

La réponse contient la liste des composants qui s'exécutent sur le périphérique principal. Chaque entrée de la liste contient l'état du cycle de vie du composant, y compris l'état actuel des données et la date à laquelle le périphérique principal Greengrass a envoyé pour la dernière fois un message contenant un certain composant au cloud. La réponse inclura également la source de déploiement la plus récente qui a intégré le composant au périphérique principal de Greengrass.

Note

Cette commande récupère une liste paginée des composants exécutés par un périphérique principal de Greengrass. Par défaut, cette liste n'inclut pas les composants déployés en tant que dépendances d'autres composants. Vous pouvez inclure les dépendances dans la réponse en définissant le `topologyFilter` paramètre sur `ALL`.

Exécuter AWS Lambda des fonctions

Note

AWS IoT Greengrass ne prend actuellement pas en charge cette fonctionnalité sur les appareils Windows principaux.

Vous pouvez importer des AWS Lambda fonctions sous forme de composants qui s'exécutent sur des appareils AWS IoT Greengrass principaux. Vous souhaitez peut-être procéder ainsi dans les cas suivants :

- Vous avez du code d'application dans les fonctions Lambda que vous souhaitez déployer sur les appareils principaux.
- Vous avez des applications AWS IoT Greengrass V1 que vous souhaitez exécuter sur des appareils AWS IoT Greengrass V2 principaux. Pour de plus amples informations, veuillez consulter [Étape 2 : créer et déployer des AWS IoT Greengrass V2 composants pour migrer AWS IoT Greengrass V1 des applications.](#)

Les fonctions Lambda incluent des dépendances sur les composants suivants. Il n'est pas nécessaire de définir ces composants comme des dépendances lorsque vous importez la fonction. Lorsque vous déployez le composant de fonction Lambda, le déploiement inclut ces dépendances du composant Lambda.

- Le [composant Lambda Launcher](#) (`aws.greengrass.LambdaLauncher`) gère les processus et la configuration de l'environnement.
- Le [composant Lambda Manager](#) (`aws.greengrass.LambdaManager`) gère la communication entre les processus et le dimensionnement.
- Le [composant Lambda runtimes](#) (`aws.greengrass.LambdaRuntimes`) fournit des artefacts pour chaque environnement d'exécution Lambda pris en charge.

Rubriques

- [Prérequis](#)
- [Configurer le cycle de vie des fonctions Lambda](#)
- [Configuration de la conteneurisation des fonctions Lambda](#)

- [Importer une fonction Lambda en tant que composant \(console\)](#)
- [Importer une fonction Lambda en tant que composant \(\)AWS CLI](#)

Prérequis

Vos appareils principaux et vos fonctions Lambda doivent répondre aux exigences suivantes pour que vous puissiez exécuter les fonctions sur le logiciel AWS IoT Greengrass principal :

- Votre appareil principal doit répondre aux exigences pour exécuter les fonctions Lambda. Si vous souhaitez que le périphérique principal exécute des fonctions Lambda conteneurisées, le périphérique doit répondre aux exigences requises. Pour de plus amples informations, veuillez consulter [Exigences relatives à la fonction Lambda](#).
- Vous devez installer les langages de programmation utilisés par la fonction Lambda sur vos appareils principaux.

Tip

Vous pouvez créer un composant qui installe le langage de programmation, puis spécifier ce composant en tant que dépendance de votre composant de fonction Lambda. Greengrass prend en charge toutes les versions compatibles avec Lambda des environnements d'exécution Python, Node.js et Java. Greengrass n'applique aucune restriction supplémentaire aux versions d'exécution Lambda obsolètes. Vous pouvez exécuter des fonctions Lambda qui utilisent ces environnements d'exécution obsolètes AWS IoT Greengrass, mais vous ne pouvez pas les créer dans. AWS Lambda Pour plus d'informations sur la AWS IoT Greengrass prise en charge des environnements d'exécution Lambda, consultez. [Exécuter AWS Lambda des fonctions](#)

Configurer le cycle de vie des fonctions 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 la manière dont le logiciel AWS IoT Greengrass Core conserve les variables et la logique de prétraitement qui se situent en dehors du gestionnaire de fonctions.

AWS IoT Greengrass prend en charge les cycles de vie à la demande (par défaut) et de longue durée :

- Les fonctions à la demande démarrent lorsqu'elles sont appelées et s'arrêtent lorsqu'il ne reste plus aucune tâche à exécuter. Chaque appel de la fonction crée un conteneur distinct, également appelé sandbox, pour traiter les invocations, sauf si un conteneur existant est disponible pour être réutilisé. Tous les conteneurs peuvent traiter les données que vous envoyez à la fonction.

Plusieurs invocations d'une fonction à la demande peuvent être exécutées simultanément.

Les variables et la logique de prétraitement que vous définissez en dehors du gestionnaire de fonctions ne sont pas conservées lors de la création de nouveaux conteneurs.

- Les fonctions de longue durée (ou épinglées) démarrent lorsque le logiciel AWS IoT Greengrass Core démarre et s'exécutent dans un conteneur unique. Le même conteneur traite toutes les données que vous envoyez à la fonction.

Les appels multiples sont mis en file d'attente jusqu'à ce que le logiciel AWS IoT Greengrass Core exécute les appels précédents.

Les variables et la logique de prétraitement que vous définissez en dehors du gestionnaire de fonctions sont conservées à chaque appel du gestionnaire.

Utilisez des fonctions Lambda de longue durée lorsque vous devez commencer à travailler sans aucune intervention initiale. Par exemple, une fonction de longue durée peut charger et commencer à traiter un modèle d'apprentissage automatique afin qu'il soit prêt lorsque la fonction reçoit les données de l'appareil.

Note

Les fonctions de longue durée ont des délais d'expiration associés à chaque appel de leur gestionnaire. Si vous souhaitez invoquer du code qui s'exécute indéfiniment, vous devez le démarrer en dehors du gestionnaire. Assurez-vous qu'aucun code de blocage en dehors du gestionnaire ne puisse empêcher l'initialisation de la fonction.

Ces fonctions s'exécutent sauf si le logiciel AWS IoT Greengrass Core s'arrête, par exemple lors d'un déploiement ou d'un redémarrage. Ces fonctions ne s'exécuteront pas si la fonction rencontre une exception non détectée, dépasse ses limites de mémoire ou entre dans un état d'erreur, tel qu'un délai d'expiration du gestionnaire.

Pour plus d'informations sur la réutilisation des conteneurs, consultez [la section Comprendre la réutilisation des conteneurs AWS Lambda dans](#) le blog AWS Compute.

Configuration de la conteneurisation des fonctions Lambda

Par défaut, les fonctions Lambda s'exécutent à l'intérieur d'un AWS IoT Greengrass conteneur. Les conteneurs Greengrass isolent vos fonctions de l'hôte. Cette isolation augmente la sécurité de l'hôte et des fonctions du 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 manière dont vous limitez l'accès aux ressources.

Vous pouvez exécuter une fonction Lambda sans conteneurisation dans les cas suivants :

- Vous souhaitez exécuter AWS IoT Greengrass sur un appareil qui ne prend pas en charge le mode conteneur. Par exemple, si vous souhaitez utiliser une distribution Linux spéciale ou si vous avez une version antérieure du noyau qui n'est plus à jour.
- 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 accéder à des ressources locales dont les chemins ne peuvent pas être déterminés au moment du déploiement ou dont les chemins peuvent changer après le déploiement. Un exemple de cette ressource serait un périphérique enfichable.
- Vous avez une application antérieure qui a été écrite en tant que processus, et vous rencontrez des problèmes lorsque vous l'exécutez dans un conteneur Greengrass.

Différences de conteneurisation

Conteneurisation	Remarques
Conteneur Greengrass	• Toutes les AWS IoT Greengrass fonctionnalités 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

Conteneurisation	Remarques
	groupe de systèmes. En d'autres termes, vos fonctions Lambda s'exécutent de manière plus isolée les unes des autres. • Étant donné que le logiciel AWS IoT Greengrass Core exécute tous les processus enfants dans le même conteneur que la fonction Lambda, les processus enfants s'arrêtent lorsque la fonction Lambda s'arrête.
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 locaux. Vous devez accéder à ces ressources en utilisant leurs chemins de fichiers sur le périphérique principal plutôt qu'en tant que ressources de fonction Lambda. • Si votre fonction Lambda non conteneurisée accède à une ressource d'apprentissage automatique, vous devez identifier le propriétaire de la ressource et définir des autorisations d'accès sur la ressource, et non sur la fonction Lambda. • Les fonctions Lambda non conteneurisées ont un accès en lecture seule au code déployé des autres fonctions Lambda exécutées avec le même groupe de systèmes.

Si vous modifiez la conteneurisation d'une fonction Lambda lorsque vous la déployez, la fonction risque de ne pas fonctionner comme prévu. Si la fonction Lambda utilise des ressources locales qui ne sont plus disponibles avec le nouveau paramètre 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 toutes les ressources associées avant de déployer la fonction Lambda.
- 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 limite de mémoire par défaut de 16 Mo. Vous pouvez configurer ces paramètres pour chaque fonction Lambda lorsque vous la déployez.

Pour modifier les paramètres de conteneurisation d'un composant de fonction Lambda, définissez la valeur du paramètre de `containerMode` configuration sur l'une des options suivantes lorsque vous déployez le composant.

- `NoContainer`— Le composant ne s'exécute pas dans un environnement d'exécution isolé.
- `GreengrassContainer`— Le composant s'exécute dans un environnement d'exécution isolé à l'intérieur du AWS IoT Greengrass conteneur.

Pour plus d'informations sur le déploiement et la configuration des composants, reportez-vous aux sections [Déployer AWS IoT Greengrass des composants sur des appareils](#) et [Mettre à jour les configurations des composants](#).

Importer une fonction Lambda en tant que composant (console)

Lorsque vous utilisez la [AWS IoT Greengrass console](#) pour créer un composant de fonction Lambda, vous importez une AWS Lambda fonction existante, puis vous la configurez pour créer un composant qui s'exécute sur votre appareil Greengrass.

Avant de commencer, passez en revue les [conditions requises](#) pour exécuter les fonctions Lambda sur les appareils Greengrass.

Tâches

- [Étape 1 : Choisissez une fonction Lambda à importer](#)

- [Étape 2 : Configuration des paramètres de la fonction Lambda](#)
- [Étape 3 : \(Facultatif\) Spécifiez les plateformes prises en charge pour la fonction Lambda](#)
- [Étape 4 : \(Facultatif\) Spécifiez les dépendances des composants pour la fonction Lambda](#)
- [Étape 5 : \(Facultatif\) Exécuter la fonction Lambda dans un conteneur](#)
- [Étape 6 : Création du composant de fonction Lambda](#)

Étape 1 : Choisissez une fonction Lambda à importer

1. Dans le menu de navigation de la [AWS IoT Greengrass console](#), sélectionnez Composants.
2. Sur la page Composants, choisissez Créer un composant.
3. Sur la page Créer un composant, sous Informations sur le composant, choisissez Importer la fonction Lambda.
4. Dans Fonction Lambda, recherchez et choisissez la fonction Lambda que vous souhaitez importer.

AWS IoT Greengrass crée le composant avec le nom de la fonction Lambda.

5. Dans la version de la fonction Lambda, choisissez la version à importer. Vous ne pouvez pas choisir des alias Lambda comme. \$LATEST

AWS IoT Greengrass crée le composant avec la version de la fonction Lambda en tant que version sémantique valide. Par exemple, si la version de votre fonction équivaut à 3, la version du composant devient 3.0.0.

Étape 2 : Configuration des paramètres de la fonction Lambda

Sur la page Créer un composant, sous Configuration de la fonction Lambda, configurez les paramètres suivants à utiliser pour exécuter la fonction Lambda.

1. (Facultatif) Ajoutez la liste des sources d'événements auxquelles la fonction Lambda est abonnés pour les messages professionnels. Vous pouvez spécifier des sources d'événements pour abonner cette fonction aux messages de publication/d'abonnement locaux et aux messages AWS IoT Core MQTT. La fonction Lambda est appelée lorsqu'elle reçoit un message d'une source d'événement.

 Note

Pour abonner cette fonction à des messages provenant d'autres fonctions ou composants Lambda, déployez l'[ancien composant routeur d'abonnement lorsque vous déployez ce composant](#) de fonction Lambda. Lorsque vous déployez l'ancien composant routeur d'abonnement, spécifiez les abonnements utilisés par la fonction Lambda.

Sous Sources d'événements, procédez comme suit pour ajouter une source d'événement :

a. Pour chaque source d'événement que vous ajoutez, spécifiez les options suivantes :

- Sujet : sujet auquel s'abonner pour recevoir des messages.
- Type : type de source d'événement. Sélectionnez parmi les options suivantes :
 - Publication/abonnement locaux — Abonnez-vous aux messages de publication/d'abonnement locaux.

Si vous utilisez [Greengrass nucleus](#) v2.6.0 ou version ultérieure et [Lambda Manager](#) v2.2.5 ou version ultérieure, vous pouvez utiliser des caractères génériques de sujet MQTT (+et#) dans le sujet lorsque vous spécifiez ce type.

- AWS IoT Core MQTT — Abonnez-vous aux messages AWS IoT Core MQTT.

Vous pouvez utiliser des caractères génériques de rubrique MQTT (+et#) dans la rubrique lorsque vous spécifiez ce type.

b. Pour ajouter une autre source d'événement, choisissez Ajouter une source d'événement et répétez l'étape précédente. Pour supprimer une source d'événement, choisissez Supprimer à côté de la source d'événement que vous souhaitez supprimer.

2. Pour Timeout (secondes), entrez la durée maximale en secondes pendant laquelle une fonction Lambda non épinglée peut s'exécuter avant son expiration. La durée par défaut est de 3 secondes.
3. Pour Épinglé, indiquez si le composant de la fonction Lambda est épinglé. La valeur par défaut est True.
 - Une fonction Lambda épinglée (ou à longue durée de vie) démarre au démarrage et continue de s' AWS IoT Greengrass exécuter dans son propre conteneur.

- Une fonction Lambda non épinglée (ou à la demande) démarre uniquement lorsqu'elle reçoit un élément de travail et s'arrête une fois qu'elle est restée inactive pendant une durée d'inactivité maximale spécifiée. Si la fonction comporte plusieurs éléments de travail, le logiciel AWS IoT Greengrass Core crée plusieurs instances de la fonction.
4. (Facultatif) Sous Paramètres supplémentaires, définissez les paramètres de la fonction Lambda suivants.
- Délai d'expiration du statut (secondes) : intervalle en secondes pendant lequel le composant de fonction Lambda envoie des mises à jour de statut au composant Lambda Manager. Ce paramètre s'applique uniquement aux fonctions épinglées. Le durée par défaut est 60 secondes.
 - Taille maximale de la file d'attente : taille maximale de la file de messages pour le composant de la fonction Lambda. Le logiciel AWS IoT Greengrass Core stocke les messages dans une file d'attente FIFO (premier entré, premier sorti) jusqu'à ce qu'il puisse exécuter la fonction Lambda pour consommer chaque message. La valeur par défaut est de 1 000 messages.
 - Nombre maximum d'instances : nombre maximal d'instances qu'une fonction Lambda non épinglée peut exécuter simultanément. La valeur par défaut est de 100 instances.
 - Durée d'inactivité maximale (secondes) : durée maximale en secondes pendant laquelle une fonction Lambda non épinglée peut être inactive avant que le logiciel AWS IoT Greengrass Core n'arrête son processus. Le durée par défaut est 60 secondes.
 - Type de codage : type de charge utile pris en charge par la fonction Lambda. Sélectionnez parmi les options suivantes :
 - JSON
 - Binaire
- La valeur par défaut est JSON.
5. (Facultatif) Spécifiez la liste des arguments de ligne de commande à transmettre à la fonction Lambda lors de son exécution.
- a. Sous Paramètres supplémentaires, Arguments de traitement, choisissez Ajouter un argument.
 - b. Pour chaque argument que vous ajoutez, entrez l'argument que vous souhaitez transmettre à la fonction.
 - c. Pour supprimer un argument, choisissez Supprimer à côté de l'argument que vous souhaitez supprimer.

6. (Facultatif) Spécifiez les variables d'environnement disponibles pour la fonction Lambda lors de son exécution. Les variables d'environnement vous permettent de stocker et de mettre à jour les paramètres de configuration sans avoir à modifier le code de fonction.
 - a. Sous Paramètres supplémentaires, Variables d'environnement, choisissez Ajouter une variable d'environnement.
 - b. Pour chaque variable d'environnement que vous ajoutez, spécifiez les options suivantes :
 - Clé : nom de la variable.
 - Valeur — La valeur par défaut de cette variable.
 - c. Pour supprimer une variable d'environnement, choisissez Supprimer à côté de la variable d'environnement que vous souhaitez supprimer.

Étape 3 : (Facultatif) Spécifiez les plateformes prises en charge pour la fonction Lambda

Tous les appareils principaux possèdent des attributs relatifs au système d'exploitation et à l'architecture. Lorsque vous déployez le composant de fonction Lambda, le logiciel AWS IoT Greengrass principal compare les valeurs de plate-forme que vous spécifiez avec les attributs de plate-forme du périphérique principal afin de déterminer si la fonction Lambda est prise en charge sur ce périphérique.

Note

Vous pouvez également spécifier des attributs de plate-forme personnalisés lorsque vous déployez le composant Greengrass nucleus sur un appareil principal. Pour plus d'informations, consultez le [paramètre Platform overrides](#) du composant [Greengrass nucleus](#).

Sous Configuration de la fonction Lambda, Paramètres supplémentaires, Plateformes, procédez comme suit pour spécifier les plates-formes prises en charge par cette fonction Lambda.

1. Pour chaque plateforme, spécifiez les options suivantes :
 - Système d'exploitation : nom du système d'exploitation de la plate-forme. À l'heure actuelle, la seule valeur prise en charge est `linux`.
 - Architecture : architecture du processeur de la plate-forme. Les valeurs prises en charge sont :

- amd64
 - arm
 - aarch64
 - x86
2. Pour ajouter une autre plateforme, choisissez Ajouter une plateforme et répétez l'étape précédente. Pour supprimer une plate-forme prise en charge, choisissez Supprimer à côté de la plate-forme que vous souhaitez supprimer.

Étape 4 : (Facultatif) Spécifiez les dépendances des composants pour la fonction Lambda

Les dépendances des composants identifient les composants supplémentaires AWS fournis ou les composants personnalisés utilisés par votre fonction. Lorsque vous déployez le composant de fonction Lambda, le déploiement inclut ces dépendances pour l'exécution de votre fonction.

Important

Pour importer une fonction Lambda que vous avez créée pour être exécutée sur la version AWS IoT Greengrass 1, vous devez définir les dépendances des composants individuels pour les fonctionnalités utilisées par votre fonction, telles que les secrets, les ombres locales et le gestionnaire de flux. Définissez ces composants comme des [dépendances fixes](#) afin que le composant de votre fonction Lambda redémarre si la dépendance change d'état. Pour de plus amples informations, veuillez consulter [Importer les fonctions Lambda de la V1](#).

Sous Configuration de la fonction Lambda, Paramètres supplémentaires, Dépendances des composants, procédez comme suit pour spécifier les dépendances des composants pour votre fonction Lambda.

1. Choisissez Ajouter une dépendance.
2. Pour chaque dépendance de composant que vous ajoutez, spécifiez les options suivantes :
 - Nom du composant : nom du composant. Par exemple, entrez **aws.greengrass.StreamManager** pour inclure le [composant du gestionnaire de flux](#).
 - Exigence de version — La contrainte de version sémantique de style npm qui identifie les versions compatibles de cette dépendance de composant. Vous pouvez spécifier une version

unique ou une série de versions. Par exemple, entrez **^1.0.0** pour spécifier que cette fonction Lambda dépend de n'importe quelle version de la première version majeure du composant du gestionnaire de flux. Pour plus d'informations sur les contraintes de version sémantiques, consultez le calculateur [npm semver](#).

- Type : type de dépendance. Sélectionnez parmi les options suivantes :
 - Difficile — Le composant de la fonction Lambda redémarre si la dépendance change d'état. Il s'agit de la sélection par défaut.
 - Soft — Le composant de la fonction Lambda ne redémarre pas si la dépendance change d'état.
3. Pour supprimer une dépendance de composant, choisissez Supprimer à côté de la dépendance de composant

Étape 5 : (Facultatif) Exécuter la fonction Lambda dans un conteneur

Par défaut, les fonctions Lambda s'exécutent dans un environnement d'exécution isolé au sein du logiciel AWS IoT Greengrass Core. Vous pouvez également choisir d'exécuter la fonction Lambda en tant que processus sans aucune isolation (c'est-à-dire en mode sans conteneur).

Dans Configuration du processus Linux, pour le mode isolation, choisissez l'une des options suivantes pour sélectionner la conteneurisation de votre fonction Lambda :

- Conteneur Greengrass — La fonction Lambda s'exécute dans un conteneur. Il s'agit de la sélection par défaut.
- Pas de conteneur : la fonction Lambda s'exécute comme un processus sans aucune isolation.

Si vous exécutez la fonction Lambda dans un conteneur, procédez comme suit pour configurer la configuration du processus pour la fonction Lambda.

1. Configurez la quantité de mémoire et les ressources système, telles que les volumes et les périphériques, à mettre à la disposition du conteneur.

Sous Paramètres du conteneur, procédez comme suit.

- a. Pour Taille de la mémoire, entrez la taille de mémoire que vous souhaitez allouer au conteneur. Vous pouvez spécifier la taille de la mémoire en Mo ou en kB.
- b. Pour le dossier sys en lecture seule, indiquez si le conteneur peut ou non lire les informations du dossier du /sys périphérique. La valeur par défaut est False.

2. (Facultatif) Configurez les volumes locaux auxquels la fonction Lambda conteneurisée peut accéder. Lorsque vous définissez un volume, le logiciel AWS IoT Greengrass Core monte les fichiers source vers la destination à l'intérieur du conteneur.
 - a. Sous Volumes, choisissez Ajouter un volume.
 - b. Pour chaque volume que vous ajoutez, spécifiez les options suivantes :
 - Volume physique : chemin d'accès au dossier source sur le périphérique principal.
 - Volume logique : chemin d'accès au dossier de destination dans le conteneur.
 - Autorisation — (Facultatif) L'autorisation d'accéder au dossier source depuis le conteneur. Sélectionnez parmi les options suivantes :
 - Lecture seule — La fonction Lambda dispose d'un accès en lecture seule au dossier source. Il s'agit de la sélection par défaut.
 - Lecture-écriture — La fonction Lambda dispose d'un accès en lecture/écriture au dossier source.
 - Ajouter le propriétaire du groupe — (Facultatif) Indique s'il faut ou non ajouter le groupe système qui exécute le composant de fonction Lambda en tant que propriétaire du dossier source. La valeur par défaut est False.
 - c. Pour supprimer un volume, choisissez Supprimer à côté du volume que vous souhaitez supprimer.
3. (Facultatif) Configurez les périphériques du système local auxquels la fonction Lambda conteneurisée peut accéder.
 - a. Sous Appareils, choisissez Ajouter un appareil.
 - b. Pour chaque appareil que vous ajoutez, définissez les options suivantes :
 - Chemin de montage : chemin d'accès au périphérique système sur le périphérique principal.
 - Autorisation — (Facultatif) L'autorisation d'accéder au périphérique système depuis le conteneur. Sélectionnez parmi les options suivantes :
 - Lecture seule — La fonction Lambda dispose d'un accès en lecture seule au périphérique système. Il s'agit de la sélection par défaut.
 - Lecture-écriture — La fonction Lambda dispose d'un accès en lecture/écriture au dossier source.

- Ajouter le propriétaire du groupe — (Facultatif) Indique s'il faut ou non ajouter le groupe système qui exécute le composant de fonction Lambda en tant que propriétaire du périphérique système. La valeur par défaut est False.

Étape 6 : Création du composant de fonction Lambda

Après avoir configuré les paramètres de votre composant de fonction Lambda, choisissez Create pour terminer la création du nouveau composant.

Pour exécuter la fonction Lambda sur votre appareil principal, vous pouvez ensuite déployer le nouveau composant sur vos appareils principaux. Pour de plus amples informations, veuillez consulter [Déployer AWS IoT Greengrass des composants sur des appareils](#).

Importer une fonction Lambda en tant que composant ()AWS CLI

Utilisez cette [CreateComponentVersion](#) opération pour créer des composants à partir de fonctions Lambda. Lorsque vous appelez cette opération, spécifiez d'importer une fonction Lambda.

Tâches

- [Étape 1 : Définition de la configuration de la fonction Lambda](#)
- [Étape 2 : Création du composant de fonction Lambda](#)

Étape 1 : Définition de la configuration de la fonction Lambda

1. Créez un fichier appelé `lambda-function-component.json`, puis copiez l'objet JSON suivant dans le fichier. Remplacez le `lambdaArn` par l'ARN de la fonction Lambda à importer.

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1"
  }
}
```

 Important

Vous devez spécifier un ARN qui inclut la version de la fonction à importer. Vous ne pouvez pas utiliser des alias de version tels que \$LATEST.

2. (Facultatif) Spécifiez le nom (componentName) du composant. Si vous omettez ce paramètre, AWS IoT Greengrass crée le composant avec le nom de la fonction Lambda.

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda"
  }
}
```

3. (Facultatif) Spécifiez la version (componentVersion) du composant. Si vous omettez ce paramètre, AWS IoT Greengrass crée le composant avec la version de la fonction Lambda en tant que version sémantique valide. Par exemple, si la version de votre fonction équivaut à 3, la version du composant devient 3.0.0.

 Note

Chaque version de composant que vous chargez doit être unique. Assurez-vous de télécharger la bonne version du composant, car vous ne pourrez pas la modifier après l'avoir chargée.

AWS IoT Greengrass utilise des versions sémantiques pour les composants. Les versions sémantiques suivent une majeure. mineur. système de numéro de patch. Par exemple, la version 1.0.0 représente la première version majeure d'un composant. Pour plus d'informations, consultez la [spécification de version sémantique](#).

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0"
  }
}
```

4. (Facultatif) Spécifiez les plateformes prises en charge par cette fonction Lambda. Chaque plateforme contient une carte des attributs qui l'identifie. Tous les appareils principaux ont des attributs pour le système d'exploitation (os) et l'architecture (architecture). Le logiciel AWS IoT Greengrass de base peut ajouter d'autres attributs de plate-forme. Vous pouvez également spécifier des attributs de plate-forme personnalisés lorsque vous déployez le [composant Greengrass nucleus](#) sur un appareil principal. Procédez comme suit :
- Ajoutez une liste de plateformes (componentPlatforms) à la fonction Lambda dans. `lambda-function-component.json`

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [

    ]
  }
}
```

- Ajoutez chaque plateforme prise en charge à la liste. Chaque plateforme dispose d'un outil convivial name permettant de l'identifier et d'une carte des attributs. L'exemple suivant indique que cette fonction prend en charge les périphériques x86 qui exécutent Linux.

```
{
  "name": "Linux x86",
  "attributes": {
    "os": "linux",
    "architecture": "x86"
  }
}
```

Vous `lambda-function-component.json` pouvez contenir un document similaire à l'exemple suivant.

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
```

```

    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
        }
      }
    ]
  }
}

```

5. (Facultatif) Spécifiez les dépendances des composants pour votre fonction Lambda. Lorsque vous déployez le composant de fonction Lambda, le déploiement inclut ces dépendances pour l'exécution de votre fonction.

Important

Pour importer une fonction Lambda que vous avez créée pour être exécutée sur la version AWS IoT Greengrass 1, vous devez définir les dépendances des composants individuels pour les fonctionnalités utilisées par votre fonction, telles que les secrets, les ombres locales et le gestionnaire de flux. Définissez ces composants comme des [dépendances fixes](#) afin que le composant de votre fonction Lambda redémarre si la dépendance change d'état. Pour de plus amples informations, veuillez consulter [Importer les fonctions Lambda de la V1](#).

Procédez comme suit :

- a. Ajoutez une carte des dépendances des composants (componentDependencies) à la fonction Lambda dans `lambda-function-component.json`

```

{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-id:function>HelloWorld:1",
    "componentName": "com.example>HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {

```



```
        "os": "linux",
        "architecture": "x86"
      }
    ],
    "componentDependencies": {

    }
  }
}
```

- b. Ajoutez chaque dépendance des composants à la carte. Spécifiez le nom du composant comme clé et spécifiez un objet avec les paramètres suivants :
- **versionRequirement**— La contrainte de version sémantique de style npm qui identifie les versions compatibles de la dépendance du composant. Vous pouvez spécifier une version unique ou une série de versions. Pour plus d'informations sur les contraintes de version sémantiques, consultez le calculateur [npm semver](#).
 - **dependencyType**— (Facultatif) Type de dépendance. Choisissez parmi les options suivantes :
 - **SOFT**— Le composant de la fonction Lambda ne redémarre pas si la dépendance change d'état.
 - **HARD**— Le composant de la fonction Lambda redémarre si la dépendance change d'état.

L'argument par défaut est HARD.

L'exemple suivant indique que cette fonction Lambda dépend de n'importe quelle version de la première version majeure du composant du [gestionnaire de flux](#). Le composant de la fonction Lambda redémarre lorsque le gestionnaire de flux redémarre ou est mis à jour.

```
{
  "aws.greengrass.StreamManager": {
    "versionRequirement": "^1.0.0",
    "dependencyType": "HARD"
  }
}
```

Vous `lambda-function-component.json` pouvez contenir un document similaire à l'exemple suivant.

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
        }
      }
    ],
    "componentDependencies": {
      "aws.greengrass.StreamManager": {
        "versionRequirement": "^1.0.0",
        "dependencyType": "HARD"
      }
    }
  }
}
```

6. (Facultatif) Configurez les paramètres de la fonction Lambda à utiliser pour exécuter la fonction. Vous pouvez configurer des options telles que les variables d'environnement, les sources d'événements des messages, les délais d'expiration et les paramètres des conteneurs. Procédez comme suit :

- a. Ajoutez l'objet de paramètres Lambda (`componentLambdaParameters`) à la fonction Lambda dans `lambda-function-component.json`

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [
      {
```

```

    "name": "Linux x86",
    "attributes": {
      "os": "linux",
      "architecture": "x86"
    }
  ],
  "componentDependencies": {
    "aws.greengrass.StreamManager": {
      "versionRequirement": "^1.0.0",
      "dependencyType": "HARD"
    }
  },
  "componentLambdaParameters": {
  }
}

```

- b. (Facultatif) Spécifiez les sources d'événements auxquelles la fonction Lambda s'abonne pour les messages professionnels. Vous pouvez spécifier des sources d'événements pour abonner cette fonction aux messages de publication/d'abonnement locaux et aux messages AWS IoT Core MQTT. La fonction Lambda est appelée lorsqu'elle reçoit un message d'une source d'événement.

 Note

Pour abonner cette fonction à des messages provenant d'autres fonctions ou composants Lambda, déployez l'[ancien composant routeur d'abonnement lorsque vous déployez ce composant](#) de fonction Lambda. Lorsque vous déployez l'ancien composant routeur d'abonnement, spécifiez les abonnements utilisés par la fonction Lambda.

Procédez comme suit :

- i. Ajoutez la liste des sources d'événements (eventSources) aux paramètres de la fonction Lambda.

```

{
  "lambdaFunction": {

```

```

"lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1",
"componentName": "com.example.HelloWorldLambda",
"componentVersion": "1.0.0",
"componentPlatforms": [
  {
    "name": "Linux x86",
    "attributes": {
      "os": "linux",
      "architecture": "x86"
    }
  }
],
"componentDependencies": {
  "aws.greengrass.StreamManager": {
    "versionRequirement": "^1.0.0",
    "dependencyType": "HARD"
  }
},
"componentLambdaParameters": {
  "eventSources": [

  ]
}
}

```

ii. Ajoutez chaque source d'événement à la liste. Chaque source d'événements possède les paramètres suivants :

- **topic**— Le sujet pour s'abonner aux messages.
- **type**— Type de source d'événement. Sélectionnez parmi les options suivantes :
 - **PUB_SUB** – Abonnez-vous aux messages locaux de publication/abonnement.

Si vous utilisez [Greengrass nucleus](#) v2.6.0 ou version ultérieure et [Lambda Manager](#) v2.2.5 ou version ultérieure, vous pouvez utiliser des caractères génériques de sujet MQTT (+et) lorsque vous spécifiez ce type. # topic

- **IOT_CORE**— Abonnez-vous aux messages AWS IoT Core MQTT.

Vous pouvez utiliser des caractères génériques de rubrique MQTT (+et#) topic lorsque vous spécifiez ce type.

L'exemple suivant s'abonne à AWS IoT Core MQTT sur des sujets qui correspondent au filtre de `hello/world/+` sujet.

```
{
  "topic": "hello/world/+",
  "type": "IOT_CORE"
}
```

Votre `lambda-function-component.json` image peut ressembler à l'exemple suivant.

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
        }
      }
    ],
    "componentDependencies": {
      "aws.greengrass.StreamManager": {
        "versionRequirement": "^1.0.0",
        "dependencyType": "HARD"
      }
    },
    "componentLambdaParameters": {
      "eventSources": [
        {
          "topic": "hello/world/+",
          "type": "IOT_CORE"
        }
      ]
    }
  }
}
```

```
}
```

c. (Facultatif) Spécifiez l'un des paramètres suivants dans l'objet de paramètres de la fonction Lambda :

- `environmentVariables`— La carte des variables d'environnement disponibles pour la fonction Lambda lors de son exécution.
- `execArgs`— Liste des arguments à transmettre à la fonction Lambda lors de son exécution.
- `inputPayloadEncodingType`— Le type de charge utile pris en charge par la fonction Lambda. Sélectionnez parmi les options suivantes :
 - `json`
 - `binary`

Par défaut : `json`

- `pinned`— Indique si la fonction Lambda est épinglée ou non. L'argument par défaut est `true`.
 - Une fonction Lambda épinglée (ou à longue durée de vie) démarre au démarrage et continue de s' AWS IoT Greengrass exécuter dans son propre conteneur.
 - Une fonction Lambda non épinglée (ou à la demande) démarre uniquement lorsqu'elle reçoit un élément de travail et s'arrête une fois qu'elle est restée inactive pendant une durée d'inactivité maximale spécifiée. Si la fonction comporte plusieurs éléments de travail, le logiciel AWS IoT Greengrass Core crée plusieurs instances de la fonction.

`maxIdleTimeInSeconds` À utiliser pour définir le temps d'inactivité maximal de votre fonction.

- `timeoutInSeconds`— Durée maximale en secondes pendant laquelle la fonction Lambda peut être exécutée avant son expiration. Le durée par défaut est de 3 secondes.
- `statusTimeoutInSeconds`— Intervalle en secondes pendant lequel le composant de la fonction Lambda envoie des mises à jour de statut au composant du gestionnaire Lambda. Ce paramètre s'applique uniquement aux fonctions épinglées. Le durée par défaut est 60 secondes.
- `maxIdleTimeInSeconds`— Durée maximale en secondes pendant laquelle une fonction Lambda non épinglée peut être inactive avant que le logiciel AWS IoT Greengrass Core n'arrête son processus. Le durée par défaut est 60 secondes.

- **maxInstancesCount**— Le nombre maximum d'instances qu'une fonction Lambda non épinglée peut exécuter simultanément. La valeur par défaut est de 100 instances.
- **maxQueueSize**— Taille maximale de la file d'attente de messages pour le composant de fonction Lambda. Le logiciel AWS IoT Greengrass Core stocke les messages dans une file d'attente FIFO (first-in-first-out) jusqu'à ce qu'il puisse exécuter la fonction Lambda pour consommer chaque message. La valeur par défaut est de 1 000 messages.

Vous `lambda-function-component.json` pouvez contenir un document similaire à l'exemple suivant.

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
        }
      }
    ],
    "componentDependencies": {
      "aws.greengrass.StreamManager": {
        "versionRequirement": "^1.0.0",
        "dependencyType": "HARD"
      }
    },
    "componentLambdaParameters": {
      "eventSources": [
        {
          "topic": "hello/world/+",
          "type": "IOT_CORE"
        }
      ],
      "environmentVariables": {
        "LIMIT": "300"
      },
      "execArgs": [
```

```

        "-d"
      ],
      "inputPayloadEncodingType": "json",
      "pinned": true,
      "timeoutInSeconds": 120,
      "statusTimeoutInSeconds": 30,
      "maxIdleTimeInSeconds": 30,
      "maxInstancesCount": 50,
      "maxQueueSize": 500
    }
  }
}

```

- d. (Facultatif) Configurez les paramètres du conteneur pour la fonction Lambda. Par défaut, les fonctions Lambda s'exécutent dans un environnement d'exécution isolé au sein du logiciel AWS IoT Greengrass Core. Vous pouvez également choisir d'exécuter la fonction Lambda en tant que processus sans aucune isolation. Si vous exécutez la fonction Lambda dans un conteneur, vous configurez la taille de la mémoire du conteneur et les ressources système disponibles pour la fonction Lambda. Procédez comme suit :
 - i. Ajoutez l'objet de paramètres de processus Linux (`linuxProcessParams`) à l'objet de paramètres Lambda dans `lambda-function-component.json`

```

{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
        }
      }
    ],
    "componentDependencies": {
      "aws.greengrass.StreamManager": {
        "versionRequirement": "^1.0.0",
        "dependencyType": "HARD"
      }
    }
  }
}

```



```

    },
    "componentLambdaParameters": {
      "eventSources": [
        {
          "topic": "hello/world/+",
          "type": "IOT_CORE"
        }
      ],
      "environmentVariables": {
        "LIMIT": "300"
      },
      "execArgs": [
        "-d"
      ],
      "inputPayloadEncodingType": "json",
      "pinned": true,
      "timeoutInSeconds": 120,
      "statusTimeoutInSeconds": 30,
      "maxIdleTimeInSeconds": 30,
      "maxInstancesCount": 50,
      "maxQueueSize": 500,
      "linuxProcessParams": {

      }
    }
  }
}

```

- ii. (Facultatif) Spécifiez si la fonction Lambda s'exécute ou non dans un conteneur. Ajoutez le `isolationMode` paramètre à l'objet des paramètres de processus, puis choisissez l'une des options suivantes :
 - `GreengrassContainer`— La fonction Lambda s'exécute dans un conteneur.
 - `NoContainer`— La fonction Lambda s'exécute comme un processus sans aucune isolation.

L'argument par défaut est `GreengrassContainer`.

- iii. (Facultatif) Si vous exécutez la fonction Lambda dans un conteneur, vous pouvez configurer la quantité de mémoire et les ressources système, telles que les volumes et les périphériques, à mettre à la disposition du conteneur. Procédez comme suit :

- A. Ajoutez l'objet de paramètres de conteneur (containerParams) à l'objet de paramètres de processus Linux dans `lambda-function-component.json`.

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
        }
      }
    ],
    "componentDependencies": {
      "aws.greengrass.StreamManager": {
        "versionRequirement": "^1.0.0",
        "dependencyType": "HARD"
      }
    },
    "componentLambdaParameters": {
      "eventSources": [
        {
          "topic": "hello/world/+",
          "type": "IOT_CORE"
        }
      ],
      "environmentVariables": {
        "LIMIT": "300"
      },
      "execArgs": [
        "-d"
      ],
      "inputPayloadEncodingType": "json",
      "pinned": true,
      "timeoutInSeconds": 120,
      "statusTimeoutInSeconds": 30,
      "maxIdleTimeInSeconds": 30,
    }
  }
}
```

```

    "maxInstancesCount": 50,
    "maxQueueSize": 500,
    "linuxProcessParams": {
        "containerParams": {
            }
        }
    }
}
}
}

```

- B. (Facultatif) Ajoutez le `memorySizeInKB` paramètre pour spécifier la taille de la mémoire du conteneur. La valeur par défaut est 16 384 Ko (16 Mo).
- C. (Facultatif) Ajoutez le `mountROSysfs` paramètre pour spécifier si le conteneur peut ou non lire les informations du `/sys` dossier du périphérique. L'argument par défaut est `false`.
- D. (Facultatif) Configurez les volumes locaux auxquels la fonction Lambda conteneurisée peut accéder. Lorsque vous définissez un volume, le logiciel AWS IoT Greengrass Core monte les fichiers source vers la destination à l'intérieur du conteneur. Procédez comme suit :
 - I. Ajoutez la liste des volumes (`volumes`) aux paramètres du conteneur.

```

{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
        }
      }
    ],
    "componentDependencies": {
      "aws.greengrass.StreamManager": {
        "versionRequirement": "^1.0.0",
        "dependencyType": "HARD"
      }
    }
  }
}

```

```

    }
  },
  "componentLambdaParameters": {
    "eventSources": [
      {
        "topic": "hello/world/+",
        "type": "IOT_CORE"
      }
    ],
    "environmentVariables": {
      "LIMIT": "300"
    },
    "execArgs": [
      "-d"
    ],
    "inputPayloadEncodingType": "json",
    "pinned": true,
    "timeoutInSeconds": 120,
    "statusTimeoutInSeconds": 30,
    "maxIdleTimeInSeconds": 30,
    "maxInstancesCount": 50,
    "maxQueueSize": 500,
    "linuxProcessParams": {
      "containerParams": {
        "memorySizeInKB": 32768,
        "mountR0Sysfs": true,
        "volumes": [

          ]
      }
    }
  }
}

```

II. Ajoutez chaque volume à la liste. Chaque volume possède les paramètres suivants :

- **sourcePath**— Le chemin d'accès au dossier source sur le périphérique principal.
- **destinationPath**— Le chemin d'accès au dossier de destination dans le conteneur.

- **permission**— (Facultatif) L'autorisation d'accéder au dossier source depuis le conteneur. Sélectionnez parmi les options suivantes :
 - **ro**— La fonction Lambda dispose d'un accès en lecture seule au dossier source.
 - **rw**— La fonction Lambda dispose d'un accès en lecture-écriture au dossier source.

L'argument par défaut est **ro**.

- **addGroupOwner**— (Facultatif) S'il faut ou non ajouter le groupe système qui exécute le composant de fonction Lambda en tant que propriétaire du dossier source. L'argument par défaut est **false**.

Vous `lambda-function-component.json` pouvez contenir un document similaire à l'exemple suivant.

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-
id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
        }
      }
    ],
    "componentDependencies": {
      "aws.greengrass.StreamManager": {
        "versionRequirement": "^1.0.0",
        "dependencyType": "HARD"
      }
    },
    "componentLambdaParameters": {
      "eventSources": [
        {
          "topic": "hello/world/+",

```

```
"type": "IOT_CORE"  
    }  
],  
"environmentVariables": {  
    "LIMIT": "300"  
},  
"execArgs": [  
    "-d"  
],  
"inputPayloadEncodingType": "json",  
"pinned": true,  
"timeoutInSeconds": 120,  
"statusTimeoutInSeconds": 30,  
"maxIdleTimeInSeconds": 30,  
"maxInstancesCount": 50,  
"maxQueueSize": 500,  
"linuxProcessParams": {  
    "containerParams": {  
        "memorySizeInKB": 32768,  
        "mountROSysfs": true,  
        "volumes": [  
            {  
                "sourcePath": "/var/data/src",  
                "destinationPath": "/var/data/dest",  
                "permission": "rw",  
                "addGroupOwner": true  
            }  
        ]  
    }  
}  
}
```

- E. (Facultatif) Configurez les périphériques du système local auxquels la fonction Lambda conteneurisée peut accéder. Procédez comme suit :
- Ajoutez la liste des périphériques système (devices) aux paramètres du conteneur.

```
{
  "lambdaFunction": {
```

```
"lambdaArn": "arn:aws:lambda:region:account-  
id:function:HelloWorld:1",  
"componentName": "com.example.HelloWorldLambda",  
"componentVersion": "1.0.0",  
"componentPlatforms": [  
  {  
    "name": "Linux x86",  
    "attributes": {  
      "os": "linux",  
      "architecture": "x86"  
    }  
  }  
],  
"componentDependencies": {  
  "aws.greengrass.StreamManager": {  
    "versionRequirement": "^1.0.0",  
    "dependencyType": "HARD"  
  }  
},  
"componentLambdaParameters": {  
  "eventSources": [  
    {  
      "topic": "hello/world/+",  
      "type": "IOT_CORE"  
    }  
  ],  
  "environmentVariables": {  
    "LIMIT": "300"  
  },  
  "execArgs": [  
    "-d"  
  ],  
  "inputPayloadEncodingType": "json",  
  "pinned": true,  
  "timeoutInSeconds": 120,  
  "statusTimeoutInSeconds": 30,  
  "maxIdleTimeInSeconds": 30,  
  "maxInstancesCount": 50,  
  "maxQueueSize": 500,  
  "linuxProcessParams": {  
    "containerParams": {  
      "memorySizeInKB": 32768,  
      "mountROSysfs": true,  
      "volumes": [  

```

```

        {
            "sourcePath": "/var/data/src",
            "destinationPath": "/var/data/dest",
            "permission": "rw",
            "addGroupOwner": true
        }
    ],
    "devices": [
        ]
    }
}

```

- II. Ajoutez chaque périphérique du système à la liste. Chaque périphérique du système possède les paramètres suivants :
- **path**— Le chemin d'accès au périphérique système sur le périphérique principal.
 - **permission**— (Facultatif) L'autorisation d'accéder au périphérique système depuis le conteneur. Sélectionnez parmi les options suivantes :
 - **ro**— La fonction Lambda dispose d'un accès en lecture seule au périphérique système.
 - **rw**— La fonction Lambda dispose d'un accès en lecture-écriture au périphérique système.

L'argument par défaut est `ro`.

- **addGroupOwner**— (Facultatif) S'il faut ou non ajouter le groupe système qui exécute le composant de fonction Lambda en tant que propriétaire du périphérique système. L'argument par défaut est `false`.

Vous `lambda-function-component.json` pouvez contenir un document similaire à l'exemple suivant.

```

{
  "lambdaFunction": {

```



```
"lambdaArn": "arn:aws:lambda:region:account-id:function>HelloWorld:1",
"componentName": "com.example>HelloWorldLambda",
"componentVersion": "1.0.0",
"componentPlatforms": [
  {
    "name": "Linux x86",
    "attributes": {
      "os": "linux",
      "architecture": "x86"
    }
  }
],
"componentDependencies": {
  "aws.greengrass.StreamManager": {
    "versionRequirement": "^1.0.0",
    "dependencyType": "HARD"
  }
},
"componentLambdaParameters": {
  "eventSources": [
    {
      "topic": "hello/world/+",
      "type": "IOT_CORE"
    }
  ],
  "environmentVariables": {
    "LIMIT": "300"
  },
  "execArgs": [
    "-d"
  ],
  "inputPayloadEncodingType": "json",
  "pinned": true,
  "timeoutInSeconds": 120,
  "statusTimeoutInSeconds": 30,
  "maxIdleTimeInSeconds": 30,
  "maxInstancesCount": 50,
  "maxQueueSize": 500,
  "linuxProcessParams": {
    "containerParams": {
      "memorySizeInKB": 32768,
      "mountROSysfs": true,
      "volumes": [
```

```

    {
      "sourcePath": "/var/data/src",
      "destinationPath": "/var/data/dest",
      "permission": "rw",
      "addGroupOwner": true
    }
  ],
  "devices": [
    {
      "path": "/dev/sda3",
      "permission": "rw",
      "addGroupOwner": true
    }
  ]
}
}
}
}
}

```

7. (Facultatif) Ajoutez des balises (tags) pour le composant. Pour de plus amples informations, veuillez consulter [Marquez vos AWS IoT Greengrass Version 2 ressources](#).

Étape 2 : Création du composant de fonction Lambda

1. Exécutez la commande suivante pour créer le composant de fonction Lambda à partir de. `lambda-function-component.json`

```
aws greengrassv2 create-component-version --cli-input-json file://lambda-function-component.json
```

La réponse ressemble à l'exemple suivant si la demande aboutit.

```

{
  "arn":
  "arn:aws:greengrass:region:123456789012:components:com.example.HelloWorldLambda:versions:1",
  "componentName": "com.example.HelloWorldLambda",
  "componentVersion": "1.0.0",
  "creationTimestamp": "Mon Dec 15 20:56:34 UTC 2020",
  "status": {
    "componentState": "REQUESTED",

```

```
    "message": "NONE",
    "errors": {}
  }
}
```

Copiez le arn depuis la sortie pour vérifier l'état du composant à l'étape suivante.

2. Lorsque vous créez un composant, son état est `REQUESTED`. AWS IoT Greengrass Vérifie ensuite que le composant est déployable. Vous pouvez exécuter la commande suivante pour connaître l'état du composant et vérifier que celui-ci est déployable. Remplacez le arn par l'ARN de l'étape précédente.

```
aws greengrassv2 describe-component \
  --arn "arn:aws:greengrass:region:account-
id:components:com.example.HelloWorldLambda:versions:1.0.0"
```

Si le composant est validé, la réponse indique que l'état du composant est `DEPLOYABLE`.

```
{
  "arn": "arn:aws:greengrass:region:account-
id:components:com.example.HelloWorldLambda:versions:1.0.0",
  "componentName": "com.example.HelloWorldLambda",
  "componentVersion": "1.0.0",
  "creationTimestamp": "2020-12-15T20:56:34.376000-08:00",
  "publisher": "AWS Lambda",
  "status": {
    "componentState": "DEPLOYABLE",
    "message": "NONE",
    "errors": {}
  },
  "platforms": [
    {
      "name": "Linux x86",
      "attributes": {
        "architecture": "x86",
        "os": "linux"
      }
    }
  ]
}
```

Une fois le composant activé `DEPLOYABLE`, vous pouvez déployer la fonction Lambda sur vos appareils principaux. Pour de plus amples informations, veuillez consulter [Déployer AWS IoT Greengrass des composants sur des appareils](#).

Utilisez le Kit SDK des appareils AWS IoT pour communiquer avec le noyau de Greengrass, les autres composants et AWS IoT Core

Les composants exécutés sur votre appareil principal peuvent utiliser la bibliothèque de communication interprocessus (IPC) du AWS IoT Greengrass Core Kit SDK des appareils AWS IoT pour communiquer avec le AWS IoT Greengrass noyau et les autres composants de Greengrass. Pour développer et exécuter des composants personnalisés utilisant IPC, vous devez utiliser le Kit SDK des appareils AWS IoT pour vous connecter au service IPC AWS IoT Greengrass principal et effectuer des opérations IPC.

L'interface IPC prend en charge deux types d'opérations :

- Demande/réponse

Les composants envoient une demande au service IPC et reçoivent une réponse contenant le résultat de la demande.

- Abonnement

Les composants envoient une demande d'abonnement au service IPC et attendent un flux de messages d'événements en réponse. Les composants fournissent un gestionnaire d'abonnement qui gère les messages d'événements, les erreurs et les fermetures de flux. Kit SDK des appareils AWS IoT Il inclut une interface de gestion avec les types de réponse et d'événement appropriés pour chaque opération IPC. Pour de plus amples informations, veuillez consulter [Abonnez-vous aux diffusions d'événements IPC](#).

Rubriques

- [Versions du client IPC](#)
- [Pris en charge SDKs pour la communication interprocessus](#)
- [Connectez-vous au service AWS IoT Greengrass Core IPC](#)
- [Autoriser les composants à effectuer des opérations IPC](#)
- [Abonnez-vous aux diffusions d'événements IPC](#)
- [Bonnes pratiques en matière d'IPC](#)
- [Publier/souscrire des messages locaux](#)

- [Publier/souscrire AWS IoT Core des messages MQTT](#)
- [Interagir avec le cycle de vie des composants](#)
- [Interagir avec la configuration des composants](#)
- [Récupérez les valeurs secrètes](#)
- [Interagissez avec les ombres locales](#)
- [Gérez les déploiements et les composants locaux](#)
- [Authentifier et autoriser les appareils clients](#)

Versions du client IPC

Dans les versions ultérieures de Java et Python SDKs, AWS IoT Greengrass fournit une version améliorée du client IPC, appelée client IPC V2. Client IPC V2 :

- Réduit la quantité de code que vous devez écrire pour utiliser les opérations IPC et permet d'éviter les erreurs courantes susceptibles de se produire avec le client IPC V1.
- Appelle les rappels du gestionnaire d'abonnement dans un thread séparé. Vous pouvez donc désormais exécuter du code de blocage, y compris des appels de fonction IPC supplémentaires, dans les rappels du gestionnaire d'abonnement. Le client IPC V1 utilise le même thread pour communiquer avec le serveur IPC et appeler les rappels du gestionnaire d'abonnement.
- Permet d'appeler des opérations d'abonnement à l'aide d'expressions Lambda (Java) ou de fonctions (Python). Le client IPC V1 nécessite que vous définissiez des classes de gestionnaires d'abonnement.
- Fournit des versions synchrones et asynchrones de chaque opération IPC. Le client IPC V1 fournit uniquement des versions asynchrones de chaque opération.

Nous vous recommandons d'utiliser le client IPC V2 pour tirer parti de ces améliorations. Cependant, de nombreux exemples présentés dans cette documentation et dans certains contenus en ligne montrent uniquement comment utiliser le client IPC V1. Vous pouvez utiliser les exemples et didacticiels suivants pour découvrir des exemples de composants utilisant le client IPC V2 :

- [PublishToTopicexemples](#)
- [SubscribeToTopicexemples](#)
- [Tutoriel : Développement d'un composant Greengrass qui reporte les mises à jour des composants](#)

- [Tutoriel : Interagissez avec des appareils IoT locaux via MQTT](#)

Actuellement, le Kit SDK des appareils AWS IoT for C++ v2 ne prend en charge que le client IPC V1.

Pris en charge SDKs pour la communication interprocessus

Les bibliothèques AWS IoT Greengrass Core IPC sont incluses dans les Kit SDK des appareils AWS IoT versions suivantes.

SDK	Version minimale	Utilisation	
Kit SDK des appareils AWS IoT pour Java v2	v1.6.0	Consultez Utilisation Kit SDK des appareils AWS IoT pour Java v2 (client IPC V2) .	
Kit SDK des appareils AWS IoT pour Python v2	v1.9.0	Consultez Utilisation Kit SDK des appareils AWS IoT pour Python v2 (client IPC V2) .	
Kit SDK des appareils AWS IoT pour C++ v2	v1.17.0	Consultez Utilisation Kit SDK des appareils AWS IoT pour C++ v2 .	
Kit SDK des appareils AWS IoT pour JavaScript v2	v1.12.0	Consultez Utilisation Kit SDK des appareils AWS IoT pour la JavaScript version 2 (client IPC V1) .	

Connectez-vous au service AWS IoT Greengrass Core IPC

Pour utiliser la communication interprocessus dans votre composant personnalisé, vous devez créer une connexion à un socket de serveur IPC exécuté par le logiciel AWS IoT Greengrass Core.

Effectuez les tâches suivantes pour télécharger et utiliser le Kit SDK des appareils AWS IoT dans la langue de votre choix.

Utilisation Kit SDK des appareils AWS IoT pour Java v2 (client IPC V2)

Pour utiliser le Kit SDK des appareils AWS IoT pour Java v2 (client IPC V2)

1. Téléchargez le [Kit SDK des appareils AWS IoT pour Java v2](#) (v1.6.0 ou version ultérieure).
2. Procédez de l'une des manières suivantes pour exécuter votre code personnalisé dans votre composant :
 - Créez votre composant sous la forme d'un fichier JAR qui inclut le Kit SDK des appareils AWS IoT, et exécutez ce fichier JAR dans votre recette de composant.
 - Définissez le Kit SDK des appareils AWS IoT JAR en tant qu'artefact de composant et ajoutez cet artefact au chemin de classe lorsque vous exécutez votre application dans votre recette de composant.
3. Utilisez le code suivant pour créer le client IPC.

```
try (GreengrassCoreIPCClientV2 ipcClient =
    GreengrassCoreIPCClientV2.builder().build()) {
    // Use client.
} catch (Exception e) {
    LOGGER.log(Level.SEVERE, "Exception occurred when using IPC.", e);
    System.exit(1);
}
```

Utilisation Kit SDK des appareils AWS IoT pour Python v2 (client IPC V2)

Pour utiliser le Kit SDK des appareils AWS IoT for Python v2 (client IPC V2)

1. Téléchargez le [Kit SDK des appareils AWS IoT pour Python](#) (v1.9.0 ou version ultérieure).
2. Ajoutez les [étapes d'installation](#) du SDK au cycle de vie d'installation dans la recette de votre composant.
3. Créez une connexion au service AWS IoT Greengrass Core IPC. Utilisez le code suivant pour créer le client IPC.

```
from awsiot.greengrasscoreipc.clientv2 import GreengrassCoreIPCClientV2
```



```
try:
    ipc_client = GreengrassCoreIPCClientV2()
    # Use IPC client.
except Exception:
    print('Exception occurred when using IPC.', file=sys.stderr)
    traceback.print_exc()
    exit(1)
```

Utilisation Kit SDK des appareils AWS IoT pour C++ v2

Pour créer la Kit SDK des appareils AWS IoT version 2 pour C++, un périphérique doit disposer des outils suivants :

- C++ 11 ou version ultérieure
- CMake 3.1 ou version ultérieure
- L'un des compilateurs suivants :
 - GCC 4.8 ou version ultérieure
 - Clang 3.9 ou version ultérieure
 - MSVC 2015 ou version ultérieure

Pour utiliser le Kit SDK des appareils AWS IoT pour C++ v2

1. Téléchargez le [Kit SDK des appareils AWS IoT pour C++ v2](#) (v1.17.0 ou version ultérieure).
2. Suivez les [instructions d'installation du fichier README](#) pour créer le Kit SDK des appareils AWS IoT pour C++ v2 à partir des sources.
3. Dans votre outil de génération C++, liez la bibliothèque IPC Greengrass que vous avez créée à l'étape précédente. `AWS::GreengrassIpc-cpp` L'`CMakeLists.txt` exemple suivant établit un lien entre la bibliothèque IPC Greengrass et un projet avec lequel vous créez un projet. CMake

```
cmake_minimum_required(VERSION 3.1)
project (greengrassv2_pubsub_subscriber)

file(GLOB MAIN_SRC
    "*.h"
    "*.cpp"
)
add_executable(${PROJECT_NAME} ${MAIN_SRC})
```

```

set_target_properties(${PROJECT_NAME} PROPERTIES
    LINKER_LANGUAGE CXX
    CXX_STANDARD 11)
find_package(aws-crt-cpp PATHS ~/sdk-cpp-workspace/build)
find_package(EventstreamRpc-cpp PATHS ~/sdk-cpp-workspace/build)
find_package(GreengrassIpc-cpp PATHS ~/sdk-cpp-workspace/build)
target_link_libraries(${PROJECT_NAME} AWS::GreengrassIpc-cpp)

```

4. Dans le code de votre composant, créez une connexion au service AWS IoT Greengrass Core IPC pour créer un client IPC (`Aws::Greengrass::GreengrassCoreIpcClient`). Vous devez définir un gestionnaire du cycle de vie des connexions IPC qui gère les événements de connexion, de déconnexion et d'erreur IPC. L'exemple suivant crée un client IPC et un gestionnaire de cycle de vie des connexions IPC qui imprime lorsque le client IPC se connecte, se déconnecte et rencontre des erreurs.

```

#include <iostream>

#include <aws/crt/Api.h>
#include <aws/greengrass/GreengrassCoreIpcClient.h>

using namespace Aws::Crt;
using namespace Aws::Greengrass;

class IpcClientLifecycleHandler : public ConnectionLifecycleHandler {
    void OnConnectCallback() override {
        std::cout << "OnConnectCallback" << std::endl;
    }

    void OnDisconnectCallback(RpcError error) override {
        std::cout << "OnDisconnectCallback: " << error.StatusToString() <<
std::endl;
        exit(-1);
    }

    bool OnErrorCallback(RpcError error) override {
        std::cout << "OnErrorCallback: " << error.StatusToString() << std::endl;
        return true;
    }
};

int main() {
    // Create the IPC client.
    ApiHandle apiHandle(g_allocator);

```

```

Io::EventLoopGroup eventLoopGroup(1);
Io::DefaultHostResolver socketResolver(eventLoopGroup, 64, 30);
Io::ClientBootstrap bootstrap(eventLoopGroup, socketResolver);
IpcClientLifecycleHandler ipcLifecycleHandler;
GreengrassCoreIpcClient ipcClient(bootstrap);
auto connectionStatus = ipcClient.Connect(ipcLifecycleHandler).get();
if (!connectionStatus) {
    std::cerr << "Failed to establish IPC connection: " <<
connectionStatus.StatusToString() << std::endl;
    exit(-1);
}

// Use the IPC client to create an operation request.

// Activate the operation request.
auto activate = operation.Activate(request, nullptr);
activate.wait();

// Wait for Greengrass Core to respond to the request.
auto responseFuture = operation.GetResult();
if (responseFuture.wait_for(std::chrono::seconds(timeout)) ==
std::future_status::timeout) {
    std::cerr << "Operation timed out while waiting for response from
Greengrass Core." << std::endl;
    exit(-1);
}

// Check the result of the request.
auto response = responseFuture.get();
if (response) {
    std::cout << "Successfully published to topic: " << topic << std::endl;
} else {
    // An error occurred.
    std::cout << "Failed to publish to topic: " << topic << std::endl;
    auto errorType = response.GetResultType();
    if (errorType == OPERATION_ERROR) {
        auto *error = response.GetOperationError();
        std::cout << "Operation error: " << error->GetMessage().value() <<
std::endl;
    } else {
        std::cout << "RPC error: " << response.GetRpcError() << std::endl;
    }
    exit(-1);
}

```

```
    return 0;
}
```

5. Pour exécuter votre code personnalisé dans votre composant, créez votre code sous forme d'artefact binaire et exécutez l'artefact binaire dans la recette de votre composant. Définissez l'Execute autorisation de l'artefact pour OWNER permettre au logiciel AWS IoT Greengrass Core d'exécuter l'artefact binaire.

La Manifests section de la recette de votre composant peut ressembler à l'exemple suivant.

JSON

```
{
  ...
  "Manifests": [
    {
      "Lifecycle": {
        "Run": "{artifacts:path}/greengrassv2_pubsub_subscriber"
      },
      "Artifacts": [
        {
          "URI": "s3://amzn-s3-demo-bucket/artifacts/
com.example.PubSubSubscriberCpp/1.0.0/greengrassv2_pubsub_subscriber",
          "Permission": {
            "Execute": "OWNER"
          }
        }
      ]
    }
  ]
}
```

YAML

```
...
Manifests:
  - Lifecycle:
      Run: {artifacts:path}/greengrassv2_pubsub_subscriber
    Artifacts:
      - URI: s3://amzn-s3-demo-bucket/artifacts/
com.example.PubSubSubscriberCpp/1.0.0/greengrassv2_pubsub_subscriber
```

```
Permission:  
Execute: OWNER
```

Utilisation Kit SDK des appareils AWS IoT pour la JavaScript version 2 (client IPC V1)

Pour créer le Kit SDK des appareils AWS IoT for JavaScript v2 à utiliser avec NodeJS, un appareil doit disposer des outils suivants :

- NodeJS 10.0 ou version ultérieure
 - Exécutez `node -v` pour vérifier la version du nœud.
- CMake 3.1 ou version ultérieure

Pour utiliser le Kit SDK des appareils AWS IoT for JavaScript v2 (client IPC V1)

1. Téléchargez le [Kit SDK des appareils AWS IoT pour la version JavaScript 2](#) (v1.12.10 ou version ultérieure).
2. Suivez les [instructions d'installation du fichier README](#) pour créer le Kit SDK des appareils AWS IoT for JavaScript v2 à partir des sources.
3. Créez une connexion au service AWS IoT Greengrass Core IPC. Procédez comme suit pour créer le client IPC et établir une connexion.
4. Utilisez le code suivant pour créer le client IPC.

```
import * as greengrasscoreipc from 'aws-iot-device-sdk-v2';  
  
let client = greengrasscoreipc.createClient();
```

5. Utilisez le code suivant pour établir une connexion entre votre composant et le noyau de Greengrass.

```
await client.connect();
```

Autoriser les composants à effectuer des opérations IPC

Pour permettre à vos composants personnalisés d'utiliser certaines opérations IPC, vous devez définir des politiques d'autorisation qui permettent au composant d'effectuer l'opération sur certaines ressources. Chaque politique d'autorisation définit une liste d'opérations et une liste de ressources

autorisées par la politique. Par exemple, le service de messagerie IPC de publication/d'abonnement définit les opérations de publication et d'abonnement pour les ressources thématiques. Vous pouvez utiliser le * caractère générique pour autoriser l'accès à toutes les opérations ou à toutes les ressources.

Vous définissez les politiques d'autorisation à l'aide du paramètre de `accessControl` configuration, que vous pouvez définir dans la recette du composant ou lorsque vous déployez le composant. L'`accessControl` objet associe les identifiants de service IPC à des listes de politiques d'autorisation. Vous pouvez définir plusieurs politiques d'autorisation pour chaque service IPC afin de contrôler l'accès. Chaque politique d'autorisation possède un identifiant de politique, qui doit être unique parmi tous les composants.

Tip

Pour créer une politique unique IDs, vous pouvez combiner le nom du composant, le nom du service IPC et un compteur. Par exemple, un composant nommé `com.example.HelloWorld` peut définir deux politiques d'autorisation de publication/d'abonnement avec les éléments suivants : IDs

- `com.example.HelloWorld:pubsub:1`
- `com.example.HelloWorld:pubsub:2`

Les politiques d'autorisation utilisent le format suivant. Cet objet est le paramètre `accessControl` de configuration.

JSON

```
{
  "IPC service identifier": {
    "policyId": {
      "policyDescription": "description",
      "operations": [
        "operation1",
        "operation2"
      ],
      "resources": [
        "resource1",
        "resource2"
      ]
    }
  }
}
```

```

    }
  }
}
```

YAML

```

IPC service identifier:
  policyId:
    policyDescription: description
    operations:
      - operation1
      - operation2
    resources:
      - resource1
      - resource2
```

Des caractères génériques dans les politiques d'autorisation

Vous pouvez utiliser le * caractère générique dans l'`resources` élément des politiques d'autorisation IPC pour autoriser l'accès à plusieurs ressources dans le cadre d'une seule politique d'autorisation.

- Dans toutes les versions du [noyau Greengrass](#), vous pouvez spécifier un seul * personnage comme ressource pour autoriser l'accès à toutes les ressources.
- Dans [Greengrass nucleus](#) v2.6.0 et versions ultérieures, vous pouvez spécifier le * personnage d'une ressource pour qu'il corresponde à n'importe quelle combinaison de caractères. Par exemple, vous pouvez autoriser `factory/1/devices/Thermostat*/status` l'accès à une rubrique d'état pour tous les thermostats d'une usine, où le nom de chaque appareil commence par `Thermostat`.

Lorsque vous définissez des politiques d'autorisation pour le service AWS IoT Core MQTT IPC, vous pouvez également utiliser des caractères génériques MQTT (+et#) pour associer plusieurs ressources. Pour plus d'informations, voir les [caractères génériques MQTT dans les politiques d'autorisation AWS IoT Core MQTT IPC](#).

Variables de recette dans les politiques d'autorisation

[Si vous utilisez Greengrass nucleus v2.6.0 ou version ultérieure et que vous définissez l'option de `interpolateComponentConfiguration` du noyau Greengrass sur, vous pouvez utiliser la](#)

[variable de recette dans les true politiques d'autorisation. {iot:thingName}](#) Lorsque vous avez besoin d'une politique d'autorisation incluant le nom du périphérique principal, par exemple pour les sujets MQTT ou les ombres d'appareils, vous pouvez utiliser cette variable de recette pour configurer une politique d'autorisation unique pour un groupe de périphériques principaux. Par exemple, vous pouvez autoriser un composant à accéder à la ressource suivante pour les opérations IPC parallèles.

```
$aws/things/{iot:thingName}/shadow/
```

Caractères spéciaux dans les politiques d'autorisation

Pour spécifier un littéral * ou un ? caractère dans une politique d'autorisation, vous devez utiliser une séquence d'échappement. Les séquences d'échappement suivantes indiquent au logiciel AWS IoT Greengrass Core d'utiliser la valeur littérale au lieu de la signification particulière du caractère. Par exemple, le * caractère est un [joker](#) qui correspond à n'importe quelle combinaison de caractères.

Caractère littéral	Séquence d'échappement	Remarques
*	<code>\${*}</code>	
?	<code>\${?}</code>	AWS IoT Greengrass ne prend actuellement pas en charge le ? joker, qui correspond à n'importe quel caractère.
\$	<code>\${\$}</code>	Utilisez cette séquence d'échappement pour faire correspondre une ressource contenant\$. Par exemple, pour faire correspondre une ressource nommée\${resourceName} , vous devez spécifier\${\$}{resourceName} . Sinon, pour faire correspondre une ressource contenant\$, vous pouvez utiliser un littéral\$, par

Caractère littéral	Séquence d'échappement	Remarques
		exemple pour autoriser l'accès à une rubrique commençant \$aws par.

Exemples de politiques d'autorisation

Vous pouvez vous référer aux exemples de politiques d'autorisation suivants pour vous aider à configurer les politiques d'autorisation pour vos composants.

Exemple de recette de composant avec une politique d'autorisation

L'exemple de recette de composant suivant inclut un `accessControl` objet qui définit une politique d'autorisation. Cette politique autorise le `com.example.HelloWorld` composant à publier dans le `test/topic` sujet.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.HelloWorld",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that publishes messages.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.pubsub": {
          "com.example.HelloWorld:pubsub:1": {
            "policyDescription": "Allows access to publish to test/topic.",
            "operations": [
              "aws.greengrass#PublishToTopic"
            ],
            "resources": [
              "test/topic"
            ]
          }
        }
      }
    }
  }
}
```

```

},
"Manifests": [
  {
    "Lifecycle": {
      "Run": "java -jar {artifacts:path}/HelloWorld.jar"
    }
  }
]
}

```

YAML

```

---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.HelloWorld
ComponentVersion: '1.0.0'
ComponentDescription: A component that publishes messages.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    accessControl:
      aws.greengrass.ipc.pubsub:
        "com.example.HelloWorld:pubsub:1":
          policyDescription: Allows access to publish to test/topic.
          operations:
            - "aws.greengrass#PublishToTopic"
          resources:
            - "test/topic"
Manifests:
  - Lifecycle:
      Run: |-
        java -jar {artifacts:path}/HelloWorld.jar

```

Exemple Exemple de mise à jour de configuration de composant avec une politique d'autorisation

L'exemple de mise à jour de configuration suivant dans un déploiement indique de configurer un composant avec un `accessControl` objet qui définit une politique d'autorisation. Cette politique autorise le `com.example.HelloWorld` composant à publier dans le `test/topic` sujet.

Console

Configuration à fusionner

```
{
  "accessControl": {
    "aws.greengrass.ipc.pubsub": {
      "com.example.HelloWorld:pubsub:1": {
        "policyDescription": "Allows access to publish to test/topic.",
        "operations": [
          "aws.greengrass#PublishToTopic"
        ],
        "resources": [
          "test/topic"
        ]
      }
    }
  }
}
```

AWS CLI

La commande suivante crée un déploiement sur un périphérique principal.

```
aws greengrassv2 create-deployment --cli-input-json file://hello-world-
deployment.json
```

Le `hello-world-deployment.json` fichier contient le document JSON suivant.

```
{
  "targetArn": "arn:aws:iot:us-west-2:123456789012:thing/MyGreengrassCore",
  "deploymentName": "Deployment for MyGreengrassCore",
  "components": {
    "com.example.HelloWorld": {
      "componentVersion": "1.0.0",
      "configurationUpdate": {
        "merge": "{\"accessControl\":{\"aws.greengrass.ipc.pubsub\":{
          \"com.example.HelloWorld:pubsub:1\":{\"policyDescription\":\"Allows access to
publish to test/topic.\",\"operations\":[\"aws.greengrass#PublishToTopic\"],
          \"resources\":[\"test/topic\"]}}}}}"
      }
    }
  }
}
```

```
}  
}
```

Greengrass CLI

La commande [Greengrass CLI](#) suivante crée un déploiement local sur un périphérique principal.

```
sudo greengrass-cli deployment create \  
  --recipeDir recipes \  
  --artifactDir artifacts \  
  --merge "com.example.HelloWorld=1.0.0" \  
  --update-config hello-world-configuration.json
```

Le `hello-world-configuration.json` fichier contient le document JSON suivant.

```
{  
  "com.example.HelloWorld": {  
    "MERGE": {  
      "accessControl": {  
        "aws.greengrass.ipc.pubsub": {  
          "com.example.HelloWorld:pubsub:1": {  
            "policyDescription": "Allows access to publish to test/topic.",  
            "operations": [  
              "aws.greengrass#PublishToTopic"  
            ],  
            "resources": [  
              "test/topic"  
            ]  
          }  
        }  
      }  
    }  
  }  
}
```

Abonnez-vous aux diffusions d'événements IPC

Vous pouvez utiliser les opérations IPC pour vous abonner à des flux d'événements sur un appareil principal de Greengrass. Pour utiliser une opération d'abonnement, définissez un gestionnaire d'abonnement et créez une demande auprès du service IPC. Ensuite, le client IPC exécute les

fonctions du gestionnaire d'abonnement chaque fois que le périphérique principal transmet un message d'événement à votre composant.

Vous pouvez fermer un abonnement pour arrêter le traitement des messages relatifs aux événements. Pour ce faire, appelez `closeStream()` (Java), `close()` (Python) ou `Close()` (C++) sur l'objet d'opération d'abonnement que vous avez utilisé pour ouvrir l'abonnement.

Le service AWS IoT Greengrass Core IPC prend en charge les opérations d'abonnement suivantes :

- [SubscribeToTopic](#)
- [SubscribeToIoTCore](#)
- [SubscribeToComponentUpdates](#)
- [SubscribeToConfigurationUpdate](#)
- [SubscribeToValidateConfigurationUpdates](#)

Rubriques

- [Définition des gestionnaires d'abonnements](#)
- [Exemples de gestionnaires d'abonnements](#)

Définition des gestionnaires d'abonnements

Pour définir un gestionnaire d'abonnement, définissez des fonctions de rappel qui gèrent les messages d'événements, les erreurs et les fermetures de flux. Si vous utilisez le client IPC V1, vous devez définir ces fonctions dans une classe. Si vous utilisez le client IPC V2, disponible dans les versions ultérieures de Java et Python SDKs, vous pouvez définir ces fonctions sans créer de classe de gestionnaire d'abonnement.

Java

Si vous utilisez le client IPC V1, vous devez implémenter `software.amazon.awssdk.eventstreamrpc.StreamResponseHandler<StreamEventType>` générique. *StreamEventType* est le type de message d'événement pour l'opération d'abonnement. Définissez les fonctions suivantes pour gérer les messages d'événements, les erreurs et les fermetures de flux.

Si vous utilisez le client IPC V2, vous pouvez définir ces fonctions en dehors d'une classe de gestionnaire d'abonnement ou utiliser des expressions [lambda](#).

```
void onStreamEvent(StreamEventType event)
```

Le rappel que le client IPC appelle lorsqu'il reçoit un message d'événement, tel qu'un message MQTT ou une notification de mise à jour de composant.

```
boolean onStreamError(Throwable error)
```

Le rappel que le client IPC appelle lorsqu'une erreur de flux se produit.

Renvoie true pour fermer le stream d'abonnement suite à l'erreur, ou renvoie false pour garder le stream ouvert.

```
void onStreamClosed()
```

Le rappel que le client IPC appelle lorsque le flux se ferme.

Python

Si vous utilisez le client IPC V1, vous devez étendre la classe du gestionnaire de réponse aux flux correspondant à l'opération d'abonnement. Kit SDK des appareils AWS IoT II inclut une classe de gestionnaire d'abonnement pour chaque opération d'abonnement. *StreamEventType* est le type de message d'événement pour l'opération d'abonnement. Définissez les fonctions suivantes pour gérer les messages d'événements, les erreurs et les fermetures de flux.

Si vous utilisez le client IPC V2, vous pouvez définir ces fonctions en dehors d'une classe de gestionnaire d'abonnement ou utiliser des expressions [lambda](#).

```
def on_stream_event(self, event: StreamEventType) -> None
```

Le rappel que le client IPC appelle lorsqu'il reçoit un message d'événement, tel qu'un message MQTT ou une notification de mise à jour de composant.

```
def on_stream_error(self, error: Exception) -> bool
```

Le rappel que le client IPC appelle lorsqu'une erreur de flux se produit.

Renvoie true pour fermer le stream d'abonnement suite à l'erreur, ou renvoie false pour garder le stream ouvert.

```
def on_stream_closed(self) -> None
```

Le rappel que le client IPC appelle lorsque le flux se ferme.

C++

Implémentez une classe dérivée de la classe du gestionnaire de réponse au flux correspondant à l'opération d'abonnement. Kit SDK des appareils AWS IoT II inclut une classe de base de gestionnaire d'abonnement pour chaque opération d'abonnement. *StreamEventType* est le type de message d'événement pour l'opération d'abonnement. Définissez les fonctions suivantes pour gérer les messages d'événements, les erreurs et les fermetures de flux.

```
void OnStreamEvent(StreamEventType *event)
```

Le rappel que le client IPC appelle lorsqu'il reçoit un message d'événement, tel qu'un message MQTT ou une notification de mise à jour de composant.

```
bool OnStreamError(OperationError *error)
```

Le rappel que le client IPC appelle lorsqu'une erreur de flux se produit.

Renvoie true pour fermer le stream d'abonnement suite à l'erreur, ou renvoie false pour garder le stream ouvert.

```
void OnStreamClosed()
```

Le rappel que le client IPC appelle lorsque le flux se ferme.

JavaScript

Implémentez une classe dérivée de la classe du gestionnaire de réponse au flux correspondant à l'opération d'abonnement. Kit SDK des appareils AWS IoT II inclut une classe de base de gestionnaire d'abonnement pour chaque opération d'abonnement. *StreamEventType* est le type de message d'événement pour l'opération d'abonnement. Définissez les fonctions suivantes pour gérer les messages d'événements, les erreurs et les fermetures de flux.

```
on(event: 'ended', listener: StreamingOperationEndedListener)
```

Le rappel que le client IPC appelle lorsque le flux se ferme.

```
on(event: 'streamError', listener: StreamingRpcErrorListener)
```

Le rappel que le client IPC appelle lorsqu'une erreur de flux se produit.

Renvoie true pour fermer le stream d'abonnement suite à l'erreur, ou renvoie false pour garder le stream ouvert.

```
on(event: 'message', listener: (message: InboundMessageType) => void)
```

Le rappel que le client IPC appelle lorsqu'il reçoit un message d'événement, tel qu'un message MQTT ou une notification de mise à jour de composant.

Exemples de gestionnaires d'abonnements

L'exemple suivant montre comment utiliser l'[SubscribeToTopic](#) opération et un gestionnaire d'abonnement pour s'abonner à des messages de publication/d'abonnement locaux.

Java (IPC client V2)

Exemple Exemple : s'abonner à des messages locaux de publication/d'abonnement

```
package com.aws.greengrass.docs.samples.ipc;

import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClientV2;
import software.amazon.awssdk.aws.greengrass.SubscribeToTopicResponseHandler;
import software.amazon.awssdk.aws.greengrass.model.*;

import java.nio.charset.StandardCharsets;
import java.util.Optional;

public class SubscribeToTopicV2 {

    public static void main(String[] args) {
        String topic = args[0];
        try (GreengrassCoreIPCClientV2 ipcClient =
GreengrassCoreIPCClientV2.builder().build()) {
            SubscribeToTopicRequest request = new
SubscribeToTopicRequest().withTopic(topic);
            GreengrassCoreIPCClientV2.StreamingResponse<SubscribeToTopicResponse,
SubscribeToTopicResponseHandler> response =
ipcClient.subscribeToTopic(request,
SubscribeToTopicV2::onStreamEvent,
Optional.of(SubscribeToTopicV2::onStreamError),
Optional.of(SubscribeToTopicV2::onStreamClosed));
            SubscribeToTopicResponseHandler responseHandler =
response.getHandler();
            System.out.println("Successfully subscribed to topic: " + topic);

            // Keep the main thread alive, or the process will exit.
```



```

        try {
            while (true) {
                Thread.sleep(10000);
            }
        } catch (InterruptedException e) {
            System.out.println("Subscribe interrupted.");
        }

        // To stop subscribing, close the stream.
        responseHandler.closeStream();
    } catch (Exception e) {
        if (e.getCause() instanceof UnauthorizedError) {
            System.err.println("Unauthorized error while publishing to topic: "
+ topic);
        } else {
            System.err.println("Exception occurred when using IPC.");
        }
        e.printStackTrace();
        System.exit(1);
    }
}

public static void onStreamEvent(SubscriptionResponseMessage
subscriptionResponseMessage) {
    try {
        BinaryMessage binaryMessage =
subscriptionResponseMessage.getBinaryMessage();
        String message = new String(binaryMessage.getMessage(),
StandardCharsets.UTF_8);
        String topic = binaryMessage.getContext().getTopic();
        System.out.printf("Received new message on topic %s: %s\n", topic,
message);
    } catch (Exception e) {
        System.err.println("Exception occurred while processing subscription
response " +
            "message.");
        e.printStackTrace();
    }
}

public static boolean onStreamError(Throwable error) {
    System.err.println("Received a stream error.");
    error.printStackTrace();
    return false; // Return true to close stream, false to keep stream open.
}

```

```

    }

    public static void onStreamClosed() {
        System.out.println("Subscribe to topic stream closed.");
    }
}

```

Python (IPC client V2)

Exemple Exemple : s'abonner à des messages locaux de publication/d'abonnement

```

import sys
import time
import traceback

from awsiot.greengrasscoreipc.clientv2 import GreengrassCoreIPCClientV2
from awsiot.greengrasscoreipc.model import (
    SubscriptionResponseMessage,
    UnauthorizedError
)

def main():
    args = sys.argv[1:]
    topic = args[0]

    try:
        ipc_client = GreengrassCoreIPCClientV2()
        # Subscription operations return a tuple with the response and the
        operation.
        _, operation = ipc_client.subscribe_to_topic(topic=topic,
on_stream_event=on_stream_event,

on_stream_error=on_stream_error, on_stream_closed=on_stream_closed)
        print('Successfully subscribed to topic: ' + topic)

        # Keep the main thread alive, or the process will exit.
        try:
            while True:
                time.sleep(10)
        except InterruptedError:
            print('Subscribe interrupted.')

        # To stop subscribing, close the stream.

```

```

        operation.close()
    except UnauthorizedError:
        print('Unauthorized error while subscribing to topic: ' +
              topic, file=sys.stderr)
        traceback.print_exc()
        exit(1)
    except Exception:
        print('Exception occurred', file=sys.stderr)
        traceback.print_exc()
        exit(1)

def on_stream_event(event: SubscriptionResponseMessage) -> None:
    try:
        message = str(event.binary_message.message, 'utf-8')
        topic = event.binary_message.context.topic
        print('Received new message on topic %s: %s' % (topic, message))
    except:
        traceback.print_exc()

def on_stream_error(error: Exception) -> bool:
    print('Received a stream error.', file=sys.stderr)
    traceback.print_exc()
    return False # Return True to close stream, False to keep stream open.

def on_stream_closed() -> None:
    print('Subscribe to topic stream closed.')

if __name__ == '__main__':
    main()

```

C++

Exemple Exemple : s'abonner à des messages locaux de publication/d'abonnement

```

#include <iostream>

#include <./crt/Api.h>
#include <aws/greengrass/GreengrassCoreIpcClient.h>

using namespace Aws::Crt;

```

```

using namespace Aws::Greengrass;

class SubscribeResponseHandler : public SubscribeToTopicStreamHandler {
public:
    virtual ~SubscribeResponseHandler() {}

private:
    void OnStreamEvent(SubscriptionResponseMessage *response) override {
        auto jsonMessage = response->GetJsonMessage();
        if (jsonMessage.has_value() &&
            jsonMessage.value().GetMessage().has_value()) {
            auto messageString =
                jsonMessage.value().GetMessage().value().View().WriteReadable();
            // Handle JSON message.
        } else {
            auto binaryMessage = response->GetBinaryMessage();
            if (binaryMessage.has_value() &&
                binaryMessage.value().GetMessage().has_value()) {
                auto messageBytes = binaryMessage.value().GetMessage().value();
                std::string messageString(messageBytes.begin(),
                    messageBytes.end());
                // Handle binary message.
            }
        }
    }

    bool OnStreamError(OperationError *error) override {
        // Handle error.
        return false; // Return true to close stream, false to keep stream open.
    }

    void OnStreamClosed() override {
        // Handle close.
    }
};

class IpcClientLifecycleHandler : public ConnectionLifecycleHandler {
    void OnConnectCallback() override {
        // Handle connection to IPC service.
    }

    void OnDisconnectCallback(RpcError error) override {
        // Handle disconnection from IPC service.
    }
}

```

```

    bool OnErrorCallback(RpcError error) override {
        // Handle IPC service connection error.
        return true;
    }
};

int main() {
    ApiHandle apiHandle(g_allocator);
    Io::EventLoopGroup eventLoopGroup(1);
    Io::DefaultHostResolver socketResolver(eventLoopGroup, 64, 30);
    Io::ClientBootstrap bootstrap(eventLoopGroup, socketResolver);
    IpcClientLifecycleHandler ipcLifecycleHandler;
    GreengrassCoreIpcClient ipcClient(bootstrap);
    auto connectionStatus = ipcClient.Connect(ipcLifecycleHandler).get();
    if (!connectionStatus) {
        std::cerr << "Failed to establish IPC connection: " <<
connectionStatus.StatusToString() << std::endl;
        exit(-1);
    }

    String topic("my/topic");
    int timeout = 10;

    SubscribeToTopicRequest request;
    request.SetTopic(topic);

    //SubscribeResponseHandler streamHandler;
    auto streamHandler = MakeShared<SubscribeResponseHandler>(DefaultAllocator());
    auto operation = ipcClient.NewSubscribeToTopic(streamHandler);
    auto activate = operation->Activate(request, nullptr);
    activate.wait();

    auto responseFuture = operation->GetResult();
    if (responseFuture.wait_for(std::chrono::seconds(timeout)) ==
std::future_status::timeout) {
        std::cerr << "Operation timed out while waiting for response from Greengrass
Core." << std::endl;
        exit(-1);
    }

    auto response = responseFuture.get();
    if (!response) {
        // Handle error.

```

```

    auto errorType = response.GetResultType();
    if (errorType == OPERATION_ERROR) {
        auto *error = response.GetOperationError();
        (void)error;
        // Handle operation error.
    } else {
        // Handle RPC error.
    }
    exit(-1);
}

// Keep the main thread alive, or the process will exit.
while (true) {
    std::this_thread::sleep_for(std::chrono::seconds(10));
}

operation->Close();
return 0;
}

```

JavaScript

Exemple Exemple : s'abonner à des messages locaux de publication/d'abonnement

```

import * as greengrasscoreipc from "aws-iot-device-sdk-v2/dist/greengrasscoreipc";
import {SubscribeToTopicRequest, SubscriptionResponseMessage} from "aws-iot-device-
sdk-v2/dist/greengrasscoreipc/model";
import {RpcError} from "aws-iot-device-sdk-v2/dist/eventstream_rpc";

class SubscribeToTopic {
    private ipcClient : greengrasscoreipc.Client
    private readonly topic : string;

    constructor() {
        // define your own constructor, e.g.
        this.topic = "<define_your_topic>";
        this.subscribeToTopic().then(r => console.log("Started workflow"));
    }

    private async subscribeToTopic() {
        try {
            this.ipcClient = await getIpClient();

```

```

        const subscribeToTopicRequest : SubscribeToTopicRequest = {
            topic: this.topic,
        }

        const streamingOperation =
this.ipcClient.subscribeToTopic(subscribeToTopicRequest, undefined); //
conditionally apply options

        streamingOperation.on("message", (message: SubscriptionResponseMessage)
=> {
            // parse the message depending on your use cases, e.g.
            if(message.binaryMessage && message.binaryMessage.message) {
                const receivedMessage =
message.binaryMessage?.message.toString();
            }
        });

        streamingOperation.on("streamError", (error : RpcError) => {
            // define your own error handling logic
        })

        streamingOperation.on("ended", () => {
            // define your own logic
        })

        await streamingOperation.activate();

        // Keep the main thread alive, or the process will exit.
        await new Promise((resolve) => setTimeout(resolve, 10000))
    } catch (e) {
        // parse the error depending on your use cases
        throw e
    }
}

export async function getIpcClient(){
    try {
        const ipcClient = greengrasscoreipc.createClient();
        await ipcClient.connect()
            .catch(error => {
                // parse the error depending on your use cases
                throw error;
            });
    }
}

```

```
        return ipcClient
    } catch (err) {
        // parse the error depending on your use cases
        throw err
    }
}

// starting point
const subscribeToTopic = new SubscribeToTopic();
```

Bonnes pratiques en matière d'IPC

Les meilleures pratiques d'utilisation d'IPC dans des composants personnalisés diffèrent entre le client IPC V1 et le client IPC V2. Suivez les meilleures pratiques pour la version du client IPC que vous utilisez.

IPC client V2

Le client IPC V2 exécute les fonctions de rappel dans un thread séparé. Par conséquent, par rapport au client IPC V1, vous devez suivre moins de directives lorsque vous utilisez IPC et écrivez des fonctions de gestion d'abonnement.

- Réutiliser un client IPC

Après avoir créé un client IPC, gardez-le ouvert et réutilisez-le pour toutes les opérations IPC. La création de plusieurs clients utilise des ressources supplémentaires et peut entraîner des fuites de ressources.

- Gérer les exceptions

Le client IPC V2 enregistre les exceptions non détectées dans les fonctions du gestionnaire d'abonnement. Vous devez intercepter les exceptions dans les fonctions de votre gestionnaire pour gérer les erreurs qui se produisent dans votre code.

IPC client V1

Le client IPC V1 utilise un seul thread qui communique avec le serveur IPC et appelle les gestionnaires d'abonnement. Vous devez tenir compte de ce comportement synchrone lorsque vous écrivez des fonctions de gestion d'abonnement.

- Réutiliser un client IPC

Après avoir créé un client IPC, gardez-le ouvert et réutilisez-le pour toutes les opérations IPC. La création de plusieurs clients utilise des ressources supplémentaires et peut entraîner des fuites de ressources.

- Exécuter le code de blocage de manière asynchrone

Le client IPC V1 ne peut pas envoyer de nouvelles demandes ou traiter de nouveaux messages d'événements lorsque le thread est bloqué. Vous devez exécuter le code de blocage dans un thread distinct que vous exécutez à partir de la fonction de gestion. Le code de blocage inclut `sleep` les appels, les boucles qui s'exécutent en continu et les demandes d'E/S synchrones dont le traitement prend du temps.

- Envoyer de nouvelles demandes IPC de manière asynchrone

Le client IPC V1 ne peut pas envoyer de nouvelle demande depuis les fonctions du gestionnaire d'abonnement, car la demande bloque la fonction du gestionnaire si vous attendez une réponse. Vous devez envoyer les requêtes IPC dans un thread distinct que vous exécutez à partir de la fonction de gestion.

- Gérer les exceptions

Le client IPC V1 ne gère pas les exceptions non détectées dans les fonctions du gestionnaire d'abonnement. Si votre fonction de gestion génère une exception, l'abonnement prend fin et l'exception n'apparaît pas dans les journaux de vos composants. Vous devez détecter les exceptions dans les fonctions de votre gestionnaire afin de maintenir l'abonnement ouvert et de consigner les erreurs qui se produisent dans votre code.

Publier/souscrire des messages locaux

La messagerie Publish/subscribe (pubsub) vous permet d'envoyer et de recevoir des messages relatifs à des sujets. Les composants peuvent publier des messages dans des rubriques pour envoyer des messages à d'autres composants. Les composants abonnés à cette rubrique peuvent ensuite agir sur les messages qu'ils reçoivent.

 Note

Vous ne pouvez pas utiliser ce service IPC de publication/abonnement pour publier ou vous abonner à MQTT. AWS IoT Core Pour plus d'informations sur l'échange de messages avec AWS IoT Core MQTT, consultez [Publier/souscrire AWS IoT Core des messages MQTT](#).

Rubriques

- [Versions minimales du SDK](#)
- [Autorisation](#)
- [PublishToTopic](#)
- [SubscribeToTopic](#)
- [Exemples](#)

Versions minimales du SDK

Le tableau suivant répertorie les versions minimales Kit SDK des appareils AWS IoT que vous devez utiliser pour publier et vous abonner à des messages à destination et en provenance de sujets locaux.

SDK	Version minimale	
Kit SDK des appareils AWS IoT pour Java v2	v1.2.10	
Kit SDK des appareils AWS IoT pour Python v2	v1.5.3	
Kit SDK des appareils AWS IoT pour C++ v2	v1.17.0	
Kit SDK des appareils AWS IoT pour JavaScript v2	v1.12.0	

Autorisation

Pour utiliser la messagerie locale de publication/d'abonnement dans un composant personnalisé, vous devez définir des politiques d'autorisation qui permettent à votre composant d'envoyer et de recevoir des messages à des sujets. Pour plus d'informations sur la définition des politiques d'autorisation, consultez [Autoriser les composants à effectuer des opérations IPC](#).

Les politiques d'autorisation pour les messages de publication/d'abonnement présentent les propriétés suivantes.

Identifiant du service IPC : `aws.greengrass.ipc.pubsub`

Opération	Description	Ressources
<code>aws.greengrass#PublishToTopic</code>	Permet à un composant de publier des messages dans les rubriques que vous spécifiez.	Une chaîne de rubrique, telle que <code>test/topic</code>. Utilisez <code>*</code> pour correspondre à n'importe quelle combinaison de caractères dans une rubrique. Cette chaîne de rubrique ne prend pas en charge les caractères génériques de rubrique MQTT (<code>#et+</code>).
<code>aws.greengrass#SubscribeToTopic</code>	Permet à un composant de s'abonner à des messages pour les sujets que vous spécifiez.	Une chaîne de rubrique, telle que <code>test/topic</code>. Utilisez <code>*</code> pour correspondre à n'importe quelle combinaison de caractères dans une rubrique. Dans Greengrass noyau v2.6.0 et versions ultérieures, vous pouvez vous abonner à des sujets contenant des caractères génériques MQTT (<code>et</code>). <code># +</code> Cette chaîne de

Opération	Description	Ressources
		rubrique prend en charge les caractères génériques des rubriques MQTT sous forme de caractères littéraux . Par exemple, si la politique d'autorisation d'un composant accorde l'accès à <code>test/topic/#</code> , le composant peut s'abonner , mais pas <code>test/topic/filter</code> .

Opération	Description	Ressources
*	Permet à un composant de publier des messages et de s'y abonner pour les sujets que vous spécifiez.	Une chaîne de rubrique, telle que <code>test/topic</code>. Utilisez <code>an *</code> pour correspondre à n'importe quelle combinaison de caractères dans une rubrique. Dans Greengrass nucleus v2.6.0 et versions ultérieures, vous pouvez vous abonner à des sujets contenant des caractères génériques MQTT (et). <code>#</code> + Cette chaîne de rubrique prend en charge les caractères génériques des rubriques MQTT sous forme de caractères littéraux. Par exemple, si la politique d'autorisation d'un composant accorde l'accès à <code>test/topic/#</code>, le composant peut s'abonner, mais pas <code>test/topic/filter</code>.

Exemples de politiques d'autorisation

Vous pouvez vous référer à l'exemple de politique d'autorisation suivant pour vous aider à configurer les politiques d'autorisation pour vos composants.

Exemple Exemple de politique d'autorisation

L'exemple de politique d'autorisation suivant permet à un composant de publier et de s'abonner à toutes les rubriques.

```
{
```

```

"accessControl": {
  "aws.greengrass.ipc.pubsub": {
    "com.example.MyLocalPubSubComponent:pubsub:1": {
      "policyDescription": "Allows access to publish/subscribe to all topics.",
      "operations": [
        "aws.greengrass#PublishToTopic",
        "aws.greengrass#SubscribeToTopic"
      ],
      "resources": [
        "*"
      ]
    }
  }
}

```

PublishToTopic

Publier un message dans une rubrique

Demande

La demande de cette opération comporte les paramètres suivants :

`topic`

Rubrique dans laquelle le message doit être publié.

`publishMessage`(Python :`publish_message`)

Le message à publier. Cet objet contient `PublishMessage` les informations suivantes. Vous devez spécifier l'une des valeurs suivantes : `jsonMessage` et `binaryMessage`.

`jsonMessage`(Python :`json_message`)

(Facultatif) Un message JSON. Cet objet contient `JsonMessage` les informations suivantes :

`message`

Le message JSON en tant qu'objet.

`context`

Le contexte du message, tel que le sujet dans lequel le message a été publié.

Cette fonctionnalité est disponible pour les versions 2.6.0 et ultérieures du composant [Greengrass](#) nucleus. Le tableau suivant répertorie les versions minimales Kit SDK des appareils AWS IoT que vous devez utiliser pour accéder au contexte du message.

SDK	Version minimale	
Kit SDK des appareils AWS IoT pour Java v2	v1.9.3	
Kit SDK des appareils AWS IoT pour Python v2	v1.11.3	
Kit SDK des appareils AWS IoT pour C++ v2	v1.18.4	
Kit SDK des appareils AWS IoT pour JavaScript v2	v1.12.0	

 Note

Le logiciel AWS IoT Greengrass Core utilise les mêmes objets de message dans les `SubscribeToTopic` opérations `PublishToTopic` et. Le logiciel AWS IoT Greengrass Core définit cet objet de contexte dans les messages lorsque vous vous abonnez et ignore cet objet de contexte dans les messages que vous publiez.

Cet objet contient `MessageContext` les informations suivantes :

`topic`

Sujet dans lequel le message a été publié.

`binaryMessage`(Python :`binary_message`)

(Facultatif) Un message binaire. Cet objet contient `BinaryMessage` les informations suivantes :

`message`

Le message binaire sous forme de blob.

context

Le contexte du message, tel que le sujet dans lequel le message a été publié.

Cette fonctionnalité est disponible pour les versions 2.6.0 et ultérieures du composant [Greengrass](#) nucleus. Le tableau suivant répertorie les versions minimales Kit SDK des appareils AWS IoT que vous devez utiliser pour accéder au contexte du message.

SDK	Version minimale	
Kit SDK des appareils AWS IoT pour Java v2	v1.9.3	
Kit SDK des appareils AWS IoT pour Python v2	v1.11.3	
Kit SDK des appareils AWS IoT pour C++ v2	v1.18.4	
Kit SDK des appareils AWS IoT pour JavaScript v2	v1.12.0	

Note

Le logiciel AWS IoT Greengrass Core utilise les mêmes objets de message dans les `SubscribeToTopic` opérations `PublishToTopic` et. Le logiciel AWS IoT Greengrass Core définit cet objet de contexte dans les messages lorsque vous vous abonnez et ignore cet objet de contexte dans les messages que vous publiez.

Cet objet contient `MessageContext` les informations suivantes :

topic

Sujet dans lequel le message a été publié.

Réponse

Cette opération ne fournit aucune information dans sa réponse.

Exemples

Les exemples suivants montrent comment appeler cette opération dans le code de composant personnalisé.

Java (IPC client V2)

Exemple Exemple : publier un message binaire

```
package com.aws.greengrass.docs.samples.ipc;

import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClientV2;
import software.amazon.awssdk.aws.greengrass.model.BinaryMessage;
import software.amazon.awssdk.aws.greengrass.model.PublishMessage;
import software.amazon.awssdk.aws.greengrass.model.PublishToTopicRequest;
import software.amazon.awssdk.aws.greengrass.model.PublishToTopicResponse;
import software.amazon.awssdk.aws.greengrass.model.UnauthorizedError;

import java.nio.charset.StandardCharsets;

public class PublishToTopicV2 {

    public static void main(String[] args) {
        String topic = args[0];
        String message = args[1];
        try (GreengrassCoreIPCClientV2 ipcClient =
GreengrassCoreIPCClientV2.builder().build()) {
            PublishToTopicV2.publishBinaryMessageToTopic(ipcClient, topic,
message);
            System.out.println("Successfully published to topic: " + topic);
        } catch (Exception e) {
            if (e.getCause() instanceof UnauthorizedError) {
                System.err.println("Unauthorized error while publishing to topic: "
+ topic);
            } else {
                System.err.println("Exception occurred when using IPC.");
            }
            e.printStackTrace();
            System.exit(1);
        }
    }
}
```

```

    }

    public static PublishToTopicResponse publishBinaryMessageToTopic(
        GreengrassCoreIPCCClientV2 ipcClient, String topic, String message)
    throws InterruptedException {
        BinaryMessage binaryMessage =
            new
        BinaryMessage().withMessage(message.getBytes(StandardCharsets.UTF_8));
        PublishMessage publishMessage = new
        PublishMessage().withBinaryMessage(binaryMessage);
        PublishToTopicRequest publishToTopicRequest =
            new
        PublishToTopicRequest().withTopic(topic).withPublishMessage(publishMessage);
        return ipcClient.publishToTopic(publishToTopicRequest);
    }
}

```

Python (IPC client V2)

Exemple Exemple : publier un message binaire

```

import sys
import traceback

from awsiot.greengrasscoreipc.clientv2 import GreengrassCoreIPCCClientV2
from awsiot.greengrasscoreipc.model import (
    PublishMessage,
    BinaryMessage
)

def main():
    args = sys.argv[1:]
    topic = args[0]
    message = args[1]

    try:
        ipc_client = GreengrassCoreIPCCClientV2()
        publish_binary_message_to_topic(ipc_client, topic, message)
        print('Successfully published to topic: ' + topic)
    except Exception:
        print('Exception occurred', file=sys.stderr)
        traceback.print_exc()
        exit(1)

```

```
def publish_binary_message_to_topic(ipc_client, topic, message):
    binary_message = BinaryMessage(message=bytes(message, 'utf-8'))
    publish_message = PublishMessage(binary_message=binary_message)
    return ipc_client.publish_to_topic(topic=topic,
    publish_message=publish_message)

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

C++

Exemple Exemple : publier un message binaire

```
#include <iostream>

#include <aws/crt/Api.h>
#include <aws/greengrass/GreengrassCoreIpcClient.h>

using namespace Aws::Crt;
using namespace Aws::Greengrass;

class IpcClientLifecycleHandler : public ConnectionLifecycleHandler {
    void OnConnectCallback() override {
        // Handle connection to IPC service.
    }

    void OnDisconnectCallback(RpcError error) override {
        // Handle disconnection from IPC service.
    }

    bool OnErrorCallback(RpcError error) override {
        // Handle IPC service connection error.
        return true;
    }
};

int main() {
    ApiHandle apiHandle(g_allocator);
    Io::EventLoopGroup eventLoopGroup(1);
    Io::DefaultHostResolver socketResolver(eventLoopGroup, 64, 30);
    Io::ClientBootstrap bootstrap(eventLoopGroup, socketResolver);
```

```
IpClientLifecycleHandler ipcLifecycleHandler;
GreengrassCoreIpClient ipcClient(bootstrap);
auto connectionStatus = ipcClient.Connect(ipcLifecycleHandler).get();
if (!connectionStatus) {
    std::cerr << "Failed to establish IPC connection: " <<
connectionStatus.StatusToString() << std::endl;
    exit(-1);
}

String topic("my/topic");
String message("Hello, World!");
int timeout = 10;

PublishToTopicRequest request;
Vector<uint8_t> messageData({message.begin(), message.end()});
BinaryMessage binaryMessage;
binaryMessage.SetMessage(messageData);
PublishMessage publishMessage;
publishMessage.SetBinaryMessage(binaryMessage);
request.SetTopic(topic);
request.SetPublishMessage(publishMessage);

auto operation = ipcClient.NewPublishToTopic();
auto activate = operation->Activate(request, nullptr);
activate.wait();

auto responseFuture = operation->GetResult();
if (responseFuture.wait_for(std::chrono::seconds(timeout)) ==
std::future_status::timeout) {
    std::cerr << "Operation timed out while waiting for response from Greengrass
Core." << std::endl;
    exit(-1);
}

auto response = responseFuture.get();
if (!response) {
    // Handle error.
    auto errorType = response.GetResultType();
    if (errorType == OPERATION_ERROR) {
        auto *error = response.GetOperationError();
        (void)error;
        // Handle operation error.
    } else {
        // Handle RPC error.
    }
}
```

```
    }  
  }  
  return 0;  
}
```

JavaScript

Exemple Exemple : publier un message binaire

```
import * as greengrasscoreipc from "aws-iot-device-sdk-v2/dist/greengrasscoreipc";  
import {BinaryMessage, PublishMessage, PublishToTopicRequest} from "aws-iot-device-  
sdk-v2/dist/greengrasscoreipc/model";  
  
class PublishToTopic {  
  private ipcClient : greengrasscoreipc.Client  
  private readonly topic : string;  
  private readonly messageString : string;  
  
  constructor() {  
    // define your own constructor, e.g.  
    this.topic = "<define_your_topic>";  
    this.messageString = "<define_your_message_string>";  
    this.publishToTopic().then(r => console.log("Started workflow"));  
  }  
  
  private async publishToTopic() {  
    try {  
      this.ipcClient = await getIpcClient();  
  
      const binaryMessage : BinaryMessage = {  
        message: this.messageString  
      }  
  
      const publishMessage : PublishMessage = {  
        binaryMessage: binaryMessage  
      }  
  
      const request : PublishToTopicRequest = {  
        topic: this.topic,  
        publishMessage: publishMessage  
      }  
    }  
  }  
}
```

```

        this.ipcClient.publishToTopic(request).finally(() =>
console.log(`Published message ${publishMessage.binaryMessage?.message} to topic`))

        } catch (e) {
            // parse the error depending on your use cases
            throw e
        }
    }
}

export async function getIpcClient(){
    try {
        const ipcClient = greengrasscoreipc.createClient();
        await ipcClient.connect()
            .catch(error => {
                // parse the error depending on your use cases
                throw error;
            });
        return ipcClient
    } catch (err) {
        // parse the error depending on your use cases
        throw err
    }
}

// starting point
const publishToTopic = new PublishToTopic();

```

SubscribeToTopic

Abonnez-vous aux messages sur un sujet.

Il s'agit d'une opération d'abonnement dans le cadre de laquelle vous vous abonnez à un flux de messages d'événements. Pour utiliser cette opération, définissez un gestionnaire de réponse au flux avec des fonctions qui gèrent les messages d'événements, les erreurs et la fermeture du flux. Pour de plus amples informations, veuillez consulter [Abonnez-vous aux diffusions d'événements IPC](#).

Type de message d'événement : SubscriptionResponseMessage

Demande

La demande de cette opération comporte les paramètres suivants :

`topic`

Rubrique à laquelle vous souhaitez vous abonner.

Note

Dans [Greengrass nucleus](#) v2.6.0 et versions ultérieures, cette rubrique prend en charge les caractères génériques du thème MQTT (et). # +

`receiveMode(Python :receive_mode)`

(Facultatif) Comportement qui indique si le composant reçoit des messages de lui-même.

Vous pouvez modifier ce comportement pour permettre à un composant d'agir sur ses propres messages. Le comportement par défaut dépend de la présence ou non d'un caractère générique MQTT dans le sujet. Sélectionnez parmi les options suivantes :

- `RECEIVE_ALL_MESSAGES`— Recevez tous les messages correspondant au sujet, y compris les messages du composant abonné.

Ce mode est l'option par défaut lorsque vous vous abonnez à un sujet qui ne contient pas de joker MQTT.

- `RECEIVE_MESSAGES_FROM_OTHERS`— Recevez tous les messages correspondant au sujet, à l'exception des messages du composant abonné.

Ce mode est l'option par défaut lorsque vous vous abonnez à un sujet contenant un caractère générique MQTT.

Cette fonctionnalité est disponible pour les versions 2.6.0 et ultérieures du composant [Greengrass nucleus](#). Le tableau suivant répertorie les versions minimales Kit SDK des appareils AWS IoT que vous devez utiliser pour définir le mode de réception.

SDK	Version minimale	
Kit SDK des appareils AWS IoT pour Java v2	v1.9.3	

SDK	Version minimale	
Kit SDK des appareils AWS IoT pour Python v2	v1.11.3	
Kit SDK des appareils AWS IoT pour C++ v2	v1.18.4	
Kit SDK des appareils AWS IoT pour JavaScript v2	v1.12.0	

Réponse

La réponse de cette opération contient les informations suivantes :

messages

Le flux de messages. Cet objet contient SubscriptionResponseMessage les informations suivantes. Chaque message contient jsonMessage ou binaryMessage.

jsonMessage(Python :json_message)

(Facultatif) Un message JSON. Cet objet contient JsonMessage les informations suivantes :

message

Le message JSON en tant qu'objet.

context

Le contexte du message, tel que le sujet dans lequel le message a été publié.

Cette fonctionnalité est disponible pour les versions 2.6.0 et ultérieures du composant [Greengrass](#) nucleus. Le tableau suivant répertorie les versions minimales Kit SDK des appareils AWS IoT que vous devez utiliser pour accéder au contexte du message.

SDK	Version minimale	
Kit SDK des appareils AWS IoT pour Java v2	v1.9.3	

SDK	Version minimale	
Kit SDK des appareils AWS IoT pour Python v2	v1.11.3	
Kit SDK des appareils AWS IoT pour C++ v2	v1.18.4	
Kit SDK des appareils AWS IoT pour JavaScript v2	v1.12.0	

 Note

Le logiciel AWS IoT Greengrass Core utilise les mêmes objets de message dans les `SubscribeToTopic` opérations `PublishToTopic` et. Le logiciel AWS IoT Greengrass Core définit cet objet de contexte dans les messages lorsque vous vous abonnez et ignore cet objet de contexte dans les messages que vous publiez.

Cet objet contient `MessageContext` les informations suivantes :

`topic`

Sujet dans lequel le message a été publié.

`binaryMessage(Python :binary_message)`

(Facultatif) Un message binaire. Cet objet contient `BinaryMessage` les informations suivantes :

`message`

Le message binaire sous forme de blob.

`context`

Le contexte du message, tel que le sujet dans lequel le message a été publié.

Cette fonctionnalité est disponible pour les versions 2.6.0 et ultérieures du composant [Greengrass](#) nucleus. Le tableau suivant répertorie les versions minimales Kit SDK des appareils AWS IoT que vous devez utiliser pour accéder au contexte du message.

SDK	Version minimale	
Kit SDK des appareils AWS IoT pour Java v2	v1.9.3	
Kit SDK des appareils AWS IoT pour Python v2	v1.11.3	
Kit SDK des appareils AWS IoT pour C++ v2	v1.18.4	
Kit SDK des appareils AWS IoT pour JavaScript v2	v1.12.0	

 Note

Le logiciel AWS IoT Greengrass Core utilise les mêmes objets de message dans les `SubscribeToTopic` opérations `PublishToTopic` et. Le logiciel AWS IoT Greengrass Core définit cet objet de contexte dans les messages lorsque vous vous abonnez et ignore cet objet de contexte dans les messages que vous publiez.

Cet objet contient `MessageContext` les informations suivantes :

`topic`

Sujet dans lequel le message a été publié.

`topicName(Python :topic_name)`

Sujet dans lequel le message a été publié.

 Note

Cette propriété n'est pas utilisée actuellement. Dans [Greengrass nucleus v2.6.0](#) et versions ultérieures, vous pouvez obtenir la `(jsonMessage | binaryMessage).context.topic` valeur de a `SubscriptionResponseMessage` pour obtenir le sujet dans lequel le message a été publié.

Exemples

Les exemples suivants montrent comment appeler cette opération dans le code de composant personnalisé.

Java (IPC client V2)

Exemple Exemple : s'abonner à des messages locaux de publication/d'abonnement

```
package com.aws.greengrass.docs.samples.ipc;

import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClientV2;
import software.amazon.awssdk.aws.greengrass.SubscribeToTopicResponseHandler;
import software.amazon.awssdk.aws.greengrass.model.*;

import java.nio.charset.StandardCharsets;
import java.util.Optional;

public class SubscribeToTopicV2 {

    public static void main(String[] args) {
        String topic = args[0];
        try (GreengrassCoreIPCClientV2 ipcClient =
GreengrassCoreIPCClientV2.builder().build()) {
            SubscribeToTopicRequest request = new
SubscribeToTopicRequest().withTopic(topic);
            GreengrassCoreIPCClientV2.StreamingResponse<SubscribeToTopicResponse,
SubscribeToTopicResponseHandler> response =
ipcClient.subscribeToTopic(request,
SubscribeToTopicV2::onStreamEvent,
Optional.of(SubscribeToTopicV2::onStreamError),
Optional.of(SubscribeToTopicV2::onStreamClosed));
            SubscribeToTopicResponseHandler responseHandler =
response.getHandler();
            System.out.println("Successfully subscribed to topic: " + topic);

            // Keep the main thread alive, or the process will exit.
            try {
                while (true) {
                    Thread.sleep(10000);
                }
            } catch (InterruptedException e) {
                System.out.println("Subscribe interrupted.");
            }
        }
    }
}
```

```

    }

    // To stop subscribing, close the stream.
    responseHandler.closeStream();
} catch (Exception e) {
    if (e.getCause() instanceof UnauthorizedError) {
        System.err.println("Unauthorized error while publishing to topic: "
+ topic);
    } else {
        System.err.println("Exception occurred when using IPC.");
    }
    e.printStackTrace();
    System.exit(1);
}
}

public static void onStreamEvent(SubscriptionResponseMessage
subscriptionResponseMessage) {
    try {
        BinaryMessage binaryMessage =
subscriptionResponseMessage.getBinaryMessage();
        String message = new String(binaryMessage.getMessage(),
StandardCharsets.UTF_8);
        String topic = binaryMessage.getContext().getTopic();
        System.out.printf("Received new message on topic %s: %s\n", topic,
message);
    } catch (Exception e) {
        System.err.println("Exception occurred while processing subscription
response " +
            "message.");
        e.printStackTrace();
    }
}

public static boolean onStreamError(Throwable error) {
    System.err.println("Received a stream error.");
    error.printStackTrace();
    return false; // Return true to close stream, false to keep stream open.
}

public static void onStreamClosed() {
    System.out.println("Subscribe to topic stream closed.");
}
}

```

```
}
```

Python (IPC client V2)

Exemple Exemple : s'abonner à des messages locaux de publication/d'abonnement

```
import sys
import time
import traceback

from awsiot.greengrasscoreipc.clientv2 import GreengrassCoreIPCClientV2
from awsiot.greengrasscoreipc.model import (
    SubscriptionResponseMessage,
    UnauthorizedError
)

def main():
    args = sys.argv[1:]
    topic = args[0]

    try:
        ipc_client = GreengrassCoreIPCClientV2()
        # Subscription operations return a tuple with the response and the
        operation.
        _, operation = ipc_client.subscribe_to_topic(topic=topic,
            on_stream_event=on_stream_event,

on_stream_error=on_stream_error, on_stream_closed=on_stream_closed)
        print('Successfully subscribed to topic: ' + topic)

        # Keep the main thread alive, or the process will exit.
        try:
            while True:
                time.sleep(10)
        except InterruptedError:
            print('Subscribe interrupted.')

        # To stop subscribing, close the stream.
        operation.close()
    except UnauthorizedError:
        print('Unauthorized error while subscribing to topic: ' +
            topic, file=sys.stderr)
        traceback.print_exc()
```

```

        exit(1)
    except Exception:
        print('Exception occurred', file=sys.stderr)
        traceback.print_exc()
        exit(1)

def on_stream_event(event: SubscriptionResponseMessage) -> None:
    try:
        message = str(event.binary_message.message, 'utf-8')
        topic = event.binary_message.context.topic
        print('Received new message on topic %s: %s' % (topic, message))
    except:
        traceback.print_exc()

def on_stream_error(error: Exception) -> bool:
    print('Received a stream error.', file=sys.stderr)
    traceback.print_exc()
    return False # Return True to close stream, False to keep stream open.

def on_stream_closed() -> None:
    print('Subscribe to topic stream closed.')

if __name__ == '__main__':
    main()

```

C++

Exemple Exemple : s'abonner à des messages locaux de publication/d'abonnement

```

#include <iostream>

#include </crt/Api.h>
#include <aws/greengrass/GreengrassCoreIpcClient.h>

using namespace Aws::Crt;
using namespace Aws::Greengrass;

class SubscribeResponseHandler : public SubscribeToTopicStreamHandler {
public:
    virtual ~SubscribeResponseHandler() {}

```

```

private:
    void OnStreamEvent(SubscriptionResponseMessage *response) override {
        auto jsonMessage = response->GetJsonMessage();
        if (jsonMessage.has_value() &&
jsonMessage.value().GetMessage().has_value()) {
            auto messageString =
jsonMessage.value().GetMessage().value().View().WriteReadable();
            // Handle JSON message.
        } else {
            auto binaryMessage = response->GetBinaryMessage();
            if (binaryMessage.has_value() &&
binaryMessage.value().GetMessage().has_value()) {
                auto messageBytes = binaryMessage.value().GetMessage().value();
                std::string messageString(messageBytes.begin(),
messageBytes.end());
                // Handle binary message.
            }
        }
    }

    bool OnStreamError(OperationError *error) override {
        // Handle error.
        return false; // Return true to close stream, false to keep stream open.
    }

    void OnStreamClosed() override {
        // Handle close.
    }
};

class IpcClientLifecycleHandler : public ConnectionLifecycleHandler {
    void OnConnectCallback() override {
        // Handle connection to IPC service.
    }

    void OnDisconnectCallback(RpcError error) override {
        // Handle disconnection from IPC service.
    }

    bool OnErrorCallback(RpcError error) override {
        // Handle IPC service connection error.
        return true;
    }
}

```

```

};

int main() {
    ApiHandle apiHandle(g_allocator);
    Io::EventLoopGroup eventLoopGroup(1);
    Io::DefaultHostResolver socketResolver(eventLoopGroup, 64, 30);
    Io::ClientBootstrap bootstrap(eventLoopGroup, socketResolver);
    IpcClientLifecycleHandler ipcLifecycleHandler;
    GreengrassCoreIpcClient ipcClient(bootstrap);
    auto connectionStatus = ipcClient.Connect(ipcLifecycleHandler).get();
    if (!connectionStatus) {
        std::cerr << "Failed to establish IPC connection: " <<
connectionStatus.StatusToString() << std::endl;
        exit(-1);
    }

    String topic("my/topic");
    int timeout = 10;

    SubscribeToTopicRequest request;
    request.SetTopic(topic);

    //SubscribeResponseHandler streamHandler;
    auto streamHandler = MakeShared<SubscribeResponseHandler>(DefaultAllocator());
    auto operation = ipcClient.NewSubscribeToTopic(streamHandler);
    auto activate = operation->Activate(request, nullptr);
    activate.wait();

    auto responseFuture = operation->GetResult();
    if (responseFuture.wait_for(std::chrono::seconds(timeout)) ==
std::future_status::timeout) {
        std::cerr << "Operation timed out while waiting for response from Greengrass
Core." << std::endl;
        exit(-1);
    }

    auto response = responseFuture.get();
    if (!response) {
        // Handle error.
        auto errorType = response.GetResultType();
        if (errorType == OPERATION_ERROR) {
            auto *error = response.GetOperationError();
            (void)error;
            // Handle operation error.

```



```

    } else {
        // Handle RPC error.
    }
    exit(-1);
}

// Keep the main thread alive, or the process will exit.
while (true) {
    std::this_thread::sleep_for(std::chrono::seconds(10));
}

operation->Close();
return 0;
}

```

JavaScript

Exemple Exemple : s'abonner à des messages locaux de publication/d'abonnement

```

import * as greengrasscoreipc from "aws-iot-device-sdk-v2/dist/greengrasscoreipc";
import {SubscribeToTopicRequest, SubscriptionResponseMessage} from "aws-iot-device-sdk-v2/dist/greengrasscoreipc/model";
import {RpcError} from "aws-iot-device-sdk-v2/dist/eventstream_rpc";

class SubscribeToTopic {
    private ipcClient : greengrasscoreipc.Client
    private readonly topic : string;

    constructor() {
        // define your own constructor, e.g.
        this.topic = "<define_your_topic>";
        this.subscribeToTopic().then(r => console.log("Started workflow"));
    }

    private async subscribeToTopic() {
        try {
            this.ipcClient = await getIpcClient();

            const subscribeToTopicRequest : SubscribeToTopicRequest = {
                topic: this.topic,
            }

```

```

        const streamingOperation =
this.ipcClient.subscribeToTopic(subscribeToTopicRequest, undefined); //
conditionally apply options

        streamingOperation.on("message", (message: SubscriptionResponseMessage)
=> {
            // parse the message depending on your use cases, e.g.
            if(message.binaryMessage && message.binaryMessage.message) {
                const receivedMessage =
message.binaryMessage?.message.toString();
            }
        });

        streamingOperation.on("streamError", (error : RpcError) => {
            // define your own error handling logic
        })

        streamingOperation.on("ended", () => {
            // define your own logic
        })

        await streamingOperation.activate();

        // Keep the main thread alive, or the process will exit.
        await new Promise((resolve) => setTimeout(resolve, 10000))
    } catch (e) {
        // parse the error depending on your use cases
        throw e
    }
}

export async function getIpcClient(){
    try {
        const ipcClient = greengrasscoreipc.createClient();
        await ipcClient.connect()
            .catch(error => {
                // parse the error depending on your use cases
                throw error;
            });
        return ipcClient
    } catch (err) {
        // parse the error depending on your use cases
        throw err
    }
}

```

```
    }  
}  
  
// starting point  
const subscribeToTopic = new SubscribeToTopic();
```

Exemples

Utilisez les exemples suivants pour savoir comment utiliser le service IPC de publication/abonnement dans vos composants.

Exemple d'éditeur de publication/abonnement (Java, client IPC V1)

L'exemple de recette suivant permet au composant de publier dans toutes les rubriques.

JSON

```
{  
  "RecipeFormatVersion": "2020-01-25",  
  "ComponentName": "com.example.PubSubPublisherJava",  
  "ComponentVersion": "1.0.0",  
  "ComponentDescription": "A component that publishes messages.",  
  "ComponentPublisher": "Amazon",  
  "ComponentConfiguration": {  
    "DefaultConfiguration": {  
      "accessControl": {  
        "aws.greengrass.ipc.pubsub": {  
          "com.example.PubSubPublisherJava:pubsub:1": {  
            "policyDescription": "Allows access to publish to all topics.",  
            "operations": [  
              "aws.greengrass#PublishToTopic"  
            ],  
            "resources": [  
              "*"   
            ]  
          }  
        }  
      }  
    }  
  },  
  "Manifests": [  
    {
```

```

    "Lifecycle": {
      "Run": "java -jar {artifacts:path}/PubSubPublisher.jar"
    }
  }
]
}

```

YAML

```

---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.PubSubPublisherJava
ComponentVersion: '1.0.0'
ComponentDescription: A component that publishes messages.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    accessControl:
      aws.greengrass.ipc.pubsub:
        'com.example.PubSubPublisherJava:pubsub:1':
          policyDescription: Allows access to publish to all topics.
          operations:
            - 'aws.greengrass#PublishToTopic'
          resources:
            - '*'
Manifests:
  - Lifecycle:
      Run: |-
        java -jar {artifacts:path}/PubSubPublisher.jar

```

L'exemple d'application Java suivant montre comment utiliser le service IPC de publication/abonnement pour publier des messages destinés à d'autres composants.

```

/* Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: Apache-2.0 */

package com.example.ipc.pubsub;

import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClient;
import software.amazon.awssdk.aws.greengrass.model.*;
import software.amazon.awssdk.eventstreamrpc.EventStreamRPCConnection;

```

```
import java.nio.charset.StandardCharsets;
import java.util.Optional;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;

public class PubSubPublisher {

    public static void main(String[] args) {
        String message = "Hello from the pub/sub publisher (Java).";
        String topic = "test/topic/java";

        try (EventStreamRPCConnection eventStreamRPCConnection =
IPCUtils.getEventStreamRpcConnection()) {
            GreengrassCoreIPCClient ipcClient = new
GreengrassCoreIPCClient(eventStreamRPCConnection);

            while (true) {
                PublishToTopicRequest publishRequest = new PublishToTopicRequest();
                PublishMessage publishMessage = new PublishMessage();
                BinaryMessage binaryMessage = new BinaryMessage();
                binaryMessage.setMessage(message.getBytes(StandardCharsets.UTF_8));
                publishMessage.setBinaryMessage(binaryMessage);
                publishRequest.setPublishMessage(publishMessage);
                publishRequest.setTopic(topic);
                CompletableFuture<PublishToTopicResponse> futureResponse = ipcClient
                    .publishToTopic(publishRequest,
Optional.empty()).getResponse();

                try {
                    futureResponse.get(10, TimeUnit.SECONDS);
                    System.out.println("Successfully published to topic: " + topic);
                } catch (TimeoutException e) {
                    System.err.println("Timeout occurred while publishing to topic: " +
topic);
                } catch (ExecutionException e) {
                    if (e.getCause() instanceof UnauthorizedError) {
                        System.err.println("Unauthorized error while publishing to
topic: " + topic);
                    } else {
                        System.err.println("Execution exception while publishing to
topic: " + topic);
                    }
                }
            }
        }
    }
}
```

```

        throw e;
    }
    Thread.sleep(5000);
}
} catch (InterruptedException e) {
    System.out.println("Publisher interrupted.");
} catch (Exception e) {
    System.err.println("Exception occurred when using IPC.");
    e.printStackTrace();
    System.exit(1);
}
}
}

```

Exemple d'abonnement publier/souscrire (Java, client IPC V1)

L'exemple de recette suivant permet au composant de s'abonner à toutes les rubriques.

JSON

```

{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.PubSubSubscriberJava",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that subscribes to messages.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.pubsub": {
          "com.example.PubSubSubscriberJava:pubsub:1": {
            "policyDescription": "Allows access to subscribe to all topics.",
            "operations": [
              "aws.greengrass#SubscribeToTopic"
            ],
            "resources": [
              "*"
            ]
          }
        }
      }
    }
  },
  "Manifests": [

```

```
{
  "Lifecycle": {
    "Run": "java -jar {artifacts:path}/PubSubSubscriber.jar"
  }
}
]
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.PubSubSubscriberJava
ComponentVersion: '1.0.0'
ComponentDescription: A component that subscribes to messages.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    accessControl:
      aws.greengrass.ipc.pubsub:
        'com.example.PubSubSubscriberJava:pubsub:1':
          policyDescription: Allows access to subscribe to all topics.
          operations:
            - 'aws.greengrass#SubscribeToTopic'
          resources:
            - '*'
Manifests:
  - Lifecycle:
      Run: |-
        java -jar {artifacts:path}/PubSubSubscriber.jar
```

L'exemple d'application Java suivant montre comment utiliser le service IPC de publication/abonnement pour s'abonner à des messages envoyés à d'autres composants.

```
/* Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: Apache-2.0 */

package com.example.ipc.pubsub;

import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClient;
import software.amazon.awssdk.aws.greengrass.SubscribeToTopicResponseHandler;
import software.amazon.awssdk.aws.greengrass.model.SubscribeToTopicRequest;
```

```
import software.amazon.awssdk.aws.greengrass.model.SubscribeToTopicResponse;
import software.amazon.awssdk.aws.greengrass.model.SubscriptionResponseMessage;
import software.amazon.awssdk.aws.greengrass.model.UnauthorizedError;
import software.amazon.awssdk.eventstreamrpc.EventStreamRPCConnection;
import software.amazon.awssdk.eventstreamrpc.StreamResponseHandler;

import java.nio.charset.StandardCharsets;
import java.util.Optional;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;

public class PubSubSubscriber {

    public static void main(String[] args) {
        String topic = "test/topic/java";

        try (EventStreamRPCConnection eventStreamRPCConnection =
IPCUtils.getEventStreamRpcConnection()) {
            GreengrassCoreIPCClient ipcClient = new
GreengrassCoreIPCClient(eventStreamRPCConnection);

            SubscribeToTopicRequest subscribeRequest = new SubscribeToTopicRequest();
            subscribeRequest.setTopic(topic);
            SubscribeToTopicResponseHandler operationResponseHandler = ipcClient
                .subscribeToTopic(subscribeRequest, Optional.of(new
SubscribeResponseHandler()));
            CompletableFuture<SubscribeToTopicResponse> futureResponse =
operationResponseHandler.getResponse();

            try {
                futureResponse.get(10, TimeUnit.SECONDS);
                System.out.println("Successfully subscribed to topic: " + topic);
            } catch (TimeoutException e) {
                System.err.println("Timeout occurred while subscribing to topic: " +
topic);
                throw e;
            } catch (ExecutionException e) {
                if (e.getCause() instanceof UnauthorizedError) {
                    System.err.println("Unauthorized error while subscribing to topic:
" + topic);
                } else {

```



```

        System.err.println("Execution exception while subscribing to topic:
" + topic);
    }
    throw e;
}

// Keep the main thread alive, or the process will exit.
try {
    while (true) {
        Thread.sleep(10000);
    }
} catch (InterruptedException e) {
    System.out.println("Subscribe interrupted.");
}
} catch (Exception e) {
    System.err.println("Exception occurred when using IPC.");
    e.printStackTrace();
    System.exit(1);
}
}

private static class SubscribeResponseHandler implements
StreamResponseHandler<SubscriptionResponseMessage> {

    @Override
    public void onStreamEvent(SubscriptionResponseMessage
subscriptionResponseMessage) {
        try {
            String message = new
String(subscriptionResponseMessage.getBinaryMessage()
.getMessage(), StandardCharsets.UTF_8);
            System.out.println("Received new message: " + message);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    @Override
    public boolean onStreamError(Throwable error) {
        System.err.println("Received a stream error.");
        error.printStackTrace();
        return false; // Return true to close stream, false to keep stream open.
    }
}

```

```

    @Override
    public void onStreamClosed() {
        System.out.println("Subscribe to topic stream closed.");
    }
}
}

```

Exemple d'éditeur de publication/abonnement (Python, client IPC V1)

L'exemple de recette suivant permet au composant de publier dans toutes les rubriques.

JSON

```

{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.PubSubPublisherPython",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that publishes messages.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.pubsub": {
          "com.example.PubSubPublisherPython:pubsub:1": {
            "policyDescription": "Allows access to publish to all topics.",
            "operations": [
              "aws.greengrass#PublishToTopic"
            ],
            "resources": [
              "*"
            ]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "install": "python3 -m pip install --user awsiotsdk",
        "Run": "python3 -u {artifacts:path}/pubsub_publisher.py"
      }
    }
  ]
}

```

```

    }
  },
  {
    "Platform": {
      "os": "windows"
    },
    "Lifecycle": {
      "install": "py -3 -m pip install --user awsiotsdk",
      "Run": "py -3 -u {artifacts:path}/pubsub_publisher.py"
    }
  }
]
}

```

YAML

```

---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.PubSubPublisherPython
ComponentVersion: 1.0.0
ComponentDescription: A component that publishes messages.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    accessControl:
      aws.greengrass.ipc.pubsub:
        com.example.PubSubPublisherPython:pubsub:1:
          policyDescription: Allows access to publish to all topics.
          operations:
            - aws.greengrass#PublishToTopic
          resources:
            - "*"
Manifests:
  - Platform:
      os: linux
      Lifecycle:
        install: python3 -m pip install --user awsiotsdk
        Run: python3 -u {artifacts:path}/pubsub_publisher.py
  - Platform:
      os: windows
      Lifecycle:
        install: py -3 -m pip install --user awsiotsdk
        Run: py -3 -u {artifacts:path}/pubsub_publisher.py

```

L'exemple d'application Python suivant montre comment utiliser le service IPC publish/subscribe pour publier des messages destinés à d'autres composants.

```
import concurrent.futures
import sys
import time
import traceback

import awsiot.greengrasscoreipc
from awsiot.greengrasscoreipc.model import (
    PublishToTopicRequest,
    PublishMessage,
    BinaryMessage,
    UnauthorizedError
)

topic = "test/topic/python"
message = "Hello from the pub/sub publisher (Python)."
TIMEOUT = 10

try:
    ipc_client = awsiot.greengrasscoreipc.connect()

    while True:
        request = PublishToTopicRequest()
        request.topic = topic
        publish_message = PublishMessage()
        publish_message.binary_message = BinaryMessage()
        publish_message.binary_message.message = bytes(message, "utf-8")
        request.publish_message = publish_message
        operation = ipc_client.new_publish_to_topic()
        operation.activate(request)
        future_response = operation.get_response()

        try:
            future_response.result(TIMEOUT)
            print('Successfully published to topic: ' + topic)
        except concurrent.futures.TimeoutError:
            print('Timeout occurred while publishing to topic: ' + topic,
                  file=sys.stderr)
        except UnauthorizedError as e:
```

```

        print('Unauthorized error while publishing to topic: ' + topic,
              file=sys.stderr)
        raise e
    except Exception as e:
        print('Exception while publishing to topic: ' + topic, file=sys.stderr)
        raise e
    time.sleep(5)
except InterruptedError:
    print('Publisher interrupted.')
except Exception:
    print('Exception occurred when using IPC.', file=sys.stderr)
    traceback.print_exc()
    exit(1)

```

Exemple d'abonné publier/souscrire (Python, client IPC V1)

L'exemple de recette suivant permet au composant de s'abonner à toutes les rubriques.

JSON

```

{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.PubSubSubscriberPython",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that subscribes to messages.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.pubsub": {
          "com.example.PubSubSubscriberPython:pubsub:1": {
            "policyDescription": "Allows access to subscribe to all topics.",
            "operations": [
              "aws.greengrass#SubscribeToTopic"
            ],
            "resources": [
              "*"
            ]
          }
        }
      }
    }
  },
  "Manifests": [

```

```

{
  "Platform": {
    "os": "linux"
  },
  "Lifecycle": {
    "install": "python3 -m pip install --user awsiotsdk",
    "Run": "python3 -u {artifacts:path}/pubsub_subscriber.py"
  }
},
{
  "Platform": {
    "os": "windows"
  },
  "Lifecycle": {
    "install": "py -3 -m pip install --user awsiotsdk",
    "Run": "py -3 -u {artifacts:path}/pubsub_subscriber.py"
  }
}
]
}

```

YAML

```

---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.PubSubSubscriberPython
ComponentVersion: 1.0.0
ComponentDescription: A component that subscribes to messages.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    accessControl:
      aws.greengrass.ipc.pubsub:
        com.example.PubSubSubscriberPython:pubsub:1:
          policyDescription: Allows access to subscribe to all topics.
          operations:
            - aws.greengrass#SubscribeToTopic
          resources:
            - "*"
Manifests:
  - Platform:
      os: linux
    Lifecycle:

```

```

install: python3 -m pip install --user awsiotsdk
Run: python3 -u {artifacts:path}/pubsub_subscriber.py
- Platform:
  os: windows
Lifecycle:
  install: py -3 -m pip install --user awsiotsdk
  Run: py -3 -u {artifacts:path}/pubsub_subscriber.py

```

L'exemple d'application Python suivant montre comment utiliser le service IPC publish/subscribe pour s'abonner à des messages envoyés à d'autres composants.

```

import concurrent.futures
import sys
import time
import traceback

import awsiot.greengrasscoreipc
import awsiot.greengrasscoreipc.client as client
from awsiot.greengrasscoreipc.model import (
    SubscribeToTopicRequest,
    SubscriptionResponseMessage,
    UnauthorizedError
)

topic = "test/topic/python"
TIMEOUT = 10

class StreamHandler(client.SubscribeToTopicStreamHandler):
    def __init__(self):
        super().__init__()

    def on_stream_event(self, event: SubscriptionResponseMessage) -> None:
        try:
            message = str(event.binary_message.message, "utf-8")
            print("Received new message: " + message)
        except:
            traceback.print_exc()

    def on_stream_error(self, error: Exception) -> bool:
        print("Received a stream error.", file=sys.stderr)
        traceback.print_exc()

```

```
        return False # Return True to close stream, False to keep stream open.

def on_stream_closed(self) -> None:
    print('Subscribe to topic stream closed.')

try:
    ipc_client = awsiot.greengrasscoreipc.connect()

    request = SubscribeToTopicRequest()
    request.topic = topic
    handler = StreamHandler()
    operation = ipc_client.new_subscribe_to_topic(handler)
    operation.activate(request)
    future_response = operation.get_response()

    try:
        future_response.result(TIMEOUT)
        print('Successfully subscribed to topic: ' + topic)
    except concurrent.futures.TimeoutError as e:
        print('Timeout occurred while subscribing to topic: ' + topic,
file=sys.stderr)
        raise e
    except UnauthorizedError as e:
        print('Unauthorized error while subscribing to topic: ' + topic,
file=sys.stderr)
        raise e
    except Exception as e:
        print('Exception while subscribing to topic: ' + topic, file=sys.stderr)
        raise e

    # Keep the main thread alive, or the process will exit.
    try:
        while True:
            time.sleep(10)
    except InterruptedError:
        print('Subscribe interrupted.')
except Exception:
    print('Exception occurred when using IPC.', file=sys.stderr)
    traceback.print_exc()
    exit(1)
```


Exemple d'éditeur de publication/abonnement (C++)

L'exemple de recette suivant permet au composant de publier dans toutes les rubriques.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.PubSubPublisherCpp",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that publishes messages.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.pubsub": {
          "com.example.PubSubPublisherCpp:pubsub:1": {
            "policyDescription": "Allows access to publish to all topics.",
            "operations": [
              "aws.greengrass#PublishToTopic"
            ],
            "resources": [
              "*"
            ]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Lifecycle": {
        "Run": "{artifacts:path}/greengrassv2_pubsub_publisher"
      },
      "Artifacts": [
        {
          "URI": "s3://amzn-s3-demo-bucket/artifacts/com.example.PubSubPublisherCpp/1.0.0/greengrassv2_pubsub_publisher",
          "Permission": {
            "Execute": "OWNER"
          }
        }
      ]
    }
  ]
}
```

```
]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.PubSubPublisherCpp
ComponentVersion: 1.0.0
ComponentDescription: A component that publishes messages.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    accessControl:
      aws.greengrass.ipc.pubsub:
        com.example.PubSubPublisherCpp:pubsub:1:
          policyDescription: Allows access to publish to all topics.
          operations:
            - aws.greengrass#PublishToTopic
          resources:
            - "*"
Manifests:
  - Lifecycle:
      Run: "{artifacts:path}/greengrassv2_pubsub_publisher"
      Artifacts:
        - URI: s3://amzn-s3-demo-bucket/artifacts/
          com.example.PubSubPublisherCpp/1.0.0/greengrassv2_pubsub_publisher
      Permission:
        Execute: OWNER
```

L'exemple d'application C++ suivant montre comment utiliser le service IPC de publication/abonnement pour publier des messages destinés à d'autres composants.

```
#include <iostream>

#include <aws/crt/Api.h>
#include <aws/greengrass/GreengrassCoreIpcClient.h>

using namespace Aws::Crt;
using namespace Aws::Greengrass;

class IpcClientLifecycleHandler : public ConnectionLifecycleHandler {
```

```
void OnConnectCallback() override {
    std::cout << "OnConnectCallback" << std::endl;
}

void OnDisconnectCallback(RpcError error) override {
    std::cout << "OnDisconnectCallback: " << error.StatusToString() << std::endl;
    exit(-1);
}

bool OnErrorCallback(RpcError error) override {
    std::cout << "OnErrorCallback: " << error.StatusToString() << std::endl;
    return true;
}
};

int main() {
    String message("Hello from the pub/sub publisher (C++).");
    String topic("test/topic/cpp");
    int timeout = 10;

    ApiHandle apiHandle(g_allocator);
    Io::EventLoopGroup eventLoopGroup(1);
    Io::DefaultHostResolver socketResolver(eventLoopGroup, 64, 30);
    Io::ClientBootstrap bootstrap(eventLoopGroup, socketResolver);
    IpcClientLifecycleHandler ipcLifecycleHandler;
    GreengrassCoreIpcClient ipcClient(bootstrap);
    auto connectionStatus = ipcClient.Connect(ipcLifecycleHandler).get();
    if (!connectionStatus) {
        std::cerr << "Failed to establish IPC connection: " <<
connectionStatus.StatusToString() << std::endl;
        exit(-1);
    }

    while (true) {
        PublishToTopicRequest request;
        Vector<uint8_t> messageData({message.begin(), message.end()});
        BinaryMessage binaryMessage;
        binaryMessage.SetMessage(messageData);
        PublishMessage publishMessage;
        publishMessage.SetBinaryMessage(binaryMessage);
        request.SetTopic(topic);
        request.SetPublishMessage(publishMessage);

        auto operation = ipcClient.NewPublishToTopic();
```

```

    auto activate = operation->Activate(request, nullptr);
    activate.wait();

    auto responseFuture = operation->GetResult();
    if (responseFuture.wait_for(std::chrono::seconds(timeout)) ==
std::future_status::timeout) {
        std::cerr << "Operation timed out while waiting for response from
Greengrass Core." << std::endl;
        exit(-1);
    }

    auto response = responseFuture.get();
    if (response) {
        std::cout << "Successfully published to topic: " << topic << std::endl;
    } else {
        // An error occurred.
        std::cout << "Failed to publish to topic: " << topic << std::endl;
        auto errorType = response.GetResultType();
        if (errorType == OPERATION_ERROR) {
            auto *error = response.GetOperationError();
            std::cout << "Operation error: " << error->GetMessage().value() <<
std::endl;
        } else {
            std::cout << "RPC error: " << response.GetRpcError() << std::endl;
        }
        exit(-1);
    }

    std::this_thread::sleep_for(std::chrono::seconds(5));
}

return 0;
}

```

Exemple d'abonnement publier/souscrire (C++)

L'exemple de recette suivant permet au composant de s'abonner à toutes les rubriques.

JSON

```

{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.PubSubSubscriberCpp",
  "ComponentVersion": "1.0.0",

```

```

"ComponentDescription": "A component that subscribes to messages.",
"ComponentPublisher": "Amazon",
"ComponentConfiguration": {
  "DefaultConfiguration": {
    "accessControl": {
      "aws.greengrass.ipc.pubsub": {
        "com.example.PubSubSubscriberCpp:pubsub:1": {
          "policyDescription": "Allows access to subscribe to all topics.",
          "operations": [
            "aws.greengrass#SubscribeToTopic"
          ],
          "resources": [
            "*"
          ]
        }
      }
    }
  },
  "Manifests": [
    {
      "Lifecycle": {
        "Run": "{artifacts:path}/greengrassv2_pub_sub_subscriber"
      },
      "Artifacts": [
        {
          "URI": "s3://amzn-s3-demo-bucket/artifacts/
com.example.PubSubSubscriberCpp/1.0.0/greengrassv2_pub_sub_subscriber",
          "Permission": {
            "Execute": "OWNER"
          }
        }
      ]
    }
  ]
}

```

YAML

```

---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.PubSubSubscriberCpp
ComponentVersion: 1.0.0

```

```

ComponentDescription: A component that subscribes to messages.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    accessControl:
      aws.greengrass.ipc.pubsub:
        com.example.PubSubSubscriberCpp:pubsub:1:
          policyDescription: Allows access to subscribe to all topics.
          operations:
            - aws.greengrass#SubscribeToTopic
          resources:
            - "*"
Manifests:
  - Lifecycle:
      Run: "{artifacts:path}/greengrassv2_pub_sub_subscriber"
    Artifacts:
      - URI: s3://amzn-s3-demo-bucket/artifacts/
        com.example.PubSubSubscriberCpp/1.0.0/greengrassv2_pub_sub_subscriber
      Permission:
        Execute: OWNER

```

L'exemple d'application C++ suivant montre comment utiliser le service IPC publish/subscribe pour s'abonner à des messages envoyés à d'autres composants.

```

#include <iostream>

#include <aws/crt/Api.h>
#include <aws/greengrass/GreengrassCoreIpcClient.h>

using namespace Aws::Crt;
using namespace Aws::Greengrass;

class SubscribeResponseHandler : public SubscribeToTopicStreamHandler {
public:
    virtual ~SubscribeResponseHandler() {}

private:
    void OnStreamEvent(SubscriptionResponseMessage *response) override {
        auto jsonMessage = response->GetJsonMessage();
        if (jsonMessage.has_value() &&
            jsonMessage.value().GetMessage().has_value()) {

```

```

        auto messageString =
jsonMessage.value().GetMessage().value().View().WriteReadable();
        std::cout << "Received new message: " << messageString << std::endl;
    } else {
        auto binaryMessage = response->GetBinaryMessage();
        if (binaryMessage.has_value() &&
binaryMessage.value().GetMessage().has_value()) {
            auto messageBytes = binaryMessage.value().GetMessage().value();
            std::string messageString(messageBytes.begin(),
messageBytes.end());
            std::cout << "Received new message: " << messageString <<
std::endl;
        }
    }
}

bool OnStreamError(OperationError *error) override {
    std::cout << "Received an operation error: ";
    if (error->GetMessage().has_value()) {
        std::cout << error->GetMessage().value();
    }
    std::cout << std::endl;
    return false; // Return true to close stream, false to keep stream open.
}

void OnStreamClosed() override {
    std::cout << "Subscribe to topic stream closed." << std::endl;
}

};

class IpcClientLifecycleHandler : public ConnectionLifecycleHandler {
    void OnConnectCallback() override {
        std::cout << "OnConnectCallback" << std::endl;
    }

    void OnDisconnectCallback(RpcError error) override {
        std::cout << "OnDisconnectCallback: " << error.StatusToString() << std::endl;
        exit(-1);
    }

    bool OnErrorCallback(RpcError error) override {
        std::cout << "OnErrorCallback: " << error.StatusToString() << std::endl;
        return true;
    }
}

```

```
};

int main() {
    String topic("test/topic/cpp");
    int timeout = 10;

    ApiHandle apiHandle(g_allocator);
    Io::EventLoopGroup eventLoopGroup(1);
    Io::DefaultHostResolver socketResolver(eventLoopGroup, 64, 30);
    Io::ClientBootstrap bootstrap(eventLoopGroup, socketResolver);
    IpcClientLifecycleHandler ipcLifecycleHandler;
    GreengrassCoreIpcClient ipcClient(bootstrap);
    auto connectionStatus = ipcClient.Connect(ipcLifecycleHandler).get();
    if (!connectionStatus) {
        std::cerr << "Failed to establish IPC connection: " <<
connectionStatus.StatusToString() << std::endl;
        exit(-1);
    }

    SubscribeToTopicRequest request;
    request.SetTopic(topic);
    auto streamHandler = MakeShared<SubscribeResponseHandler>(DefaultAllocator());
    auto operation = ipcClient.NewSubscribeToTopic(streamHandler);
    auto activate = operation->Activate(request, nullptr);
    activate.wait();

    auto responseFuture = operation->GetResult();
    if (responseFuture.wait_for(std::chrono::seconds(timeout)) ==
std::future_status::timeout) {
        std::cerr << "Operation timed out while waiting for response from Greengrass
Core." << std::endl;
        exit(-1);
    }

    auto response = responseFuture.get();
    if (response) {
        std::cout << "Successfully subscribed to topic: " << topic << std::endl;
    } else {
        // An error occurred.
        std::cout << "Failed to subscribe to topic: " << topic << std::endl;
        auto errorType = response.GetResultType();
        if (errorType == OPERATION_ERROR) {
            auto *error = response.GetOperationError();

```



```
        std::cout << "Operation error: " << error->GetMessage().value() <<
std::endl;
    } else {
        std::cout << "RPC error: " << response.GetRpcError() << std::endl;
    }
    exit(-1);
}

// Keep the main thread alive, or the process will exit.
while (true) {
    std::this_thread::sleep_for(std::chrono::seconds(10));
}

operation->Close();
return 0;
}
```

Publier/souscrire AWS IoT Core des messages MQTT

Le service IPC de messagerie AWS IoT Core MQTT vous permet d'envoyer et de recevoir des messages MQTT depuis et vers. AWS IoT Core Les composants peuvent publier des messages AWS IoT Core et s'abonner à des sujets pour agir sur les messages MQTT provenant d'autres sources. Pour plus d'informations sur l' AWS IoT Core implémentation de MQTT, consultez [MQTT](#) dans le Guide du AWS IoT Core développeur.

Note

Ce service IPC de messagerie MQTT vous permet d'échanger des messages avec. AWS IoT Core Pour plus d'informations sur l'échange de messages entre composants, consultez [Publier/souscrire des messages locaux](#).

Rubriques

- [Versions minimales du SDK](#)
- [Autorisation](#)
- [PublishToIoTCore](#)
- [SubscribeToIoTCore](#)
- [Exemples](#)

Versions minimales du SDK

Le tableau suivant répertorie les versions minimales Kit SDK des appareils AWS IoT que vous devez utiliser pour publier et vous abonner à des messages MQTT en provenance AWS IoT Core et à destination.

SDK	Version minimale	
Kit SDK des appareils AWS IoT pour Java v2	v1.2.10	
Kit SDK des appareils AWS IoT pour Python v2	v1.5.3	
Kit SDK des appareils AWS IoT pour C++ v2	v1.17.0	
Kit SDK des appareils AWS IoT pour JavaScript v2	v1.12.0	

Autorisation

Pour utiliser la messagerie AWS IoT Core MQTT dans un composant personnalisé, vous devez définir des politiques d'autorisation qui permettent à votre composant d'envoyer et de recevoir des messages sur des sujets. Pour plus d'informations sur la définition des politiques d'autorisation, consultez [Autoriser les composants à effectuer des opérations IPC](#).

Les politiques d'autorisation pour AWS IoT Core la messagerie MQTT présentent les propriétés suivantes.

Identifiant du service IPC : `aws.greengrass.ipc.mqttproxy`

Opération	Description	Ressources
<code>aws.greengrass#PublishToIoTCore</code>	Permet à un composant de publier des messages AWS IoT Core sur les sujets MQTT que vous spécifiez.	Chaîne de rubrique, telle que <code>test/topic</code> , ou <code>*</code> pour autoriser l'accès à toutes les

Opération	Description	Ressources
		rubriques. Vous pouvez utiliser les caractères génériques des rubriques MQTT (#et+) pour associer plusieurs ressources.
<code>aws.greengrass#SubscribeToIoTCore</code>	Permet à un composant de s'abonner à des messages provenant AWS IoT Core des sujets que vous spécifiez.	Chaîne de rubrique, telle que <code>test/topic</code> , ou <code>*</code> pour autoriser l'accès à toutes les rubriques. Vous pouvez utiliser les caractères génériques des rubriques MQTT (#et+) pour associer plusieurs ressources.
<code>*</code>	Permet à un composant de publier des messages AWS IoT Core MQTT et de s'y abonner pour les sujets que vous spécifiez.	Chaîne de rubrique, telle que <code>test/topic</code> , ou <code>*</code> pour autoriser l'accès à toutes les rubriques. Vous pouvez utiliser les caractères génériques des rubriques MQTT (#et+) pour associer plusieurs ressources.

Caractères génériques MQTT dans les politiques d'autorisation AWS IoT Core MQTT

Vous pouvez utiliser des caractères génériques MQTT dans les politiques d'autorisation AWS IoT Core MQTT IPC. Les composants peuvent publier et s'abonner à des rubriques qui correspondent au filtre de rubrique que vous autorisez dans une politique d'autorisation. Par exemple, si la politique d'autorisation d'un composant accorde l'accès à `test/topic/#`, le composant peut s'abonner à `test/topic/#`, publier et s'abonner à `test/topic/filter`.

Variables de recette dans les AWS IoT Core politiques d'autorisation MQTT

Si vous utilisez la version 2.6.0 ou ultérieure du [noyau](#) Greengrass, vous pouvez utiliser la variable de `{iot:thingName}` recette dans les politiques d'autorisation. Cette fonctionnalité vous permet de configurer une politique d'autorisation unique pour un groupe de périphériques principaux, chaque périphérique principal ne pouvant accéder qu'aux rubriques contenant son propre nom. Par exemple, vous pouvez autoriser un composant à accéder à la ressource thématique suivante.

```
devices/{iot:thingName}/messages
```

Pour plus d'informations, consultez [Variables de recette](#) et [Utiliser des variables de recette dans les mises à jour de fusion](#).

Exemples de politiques d'autorisation

Vous pouvez vous référer aux exemples de politiques d'autorisation suivants pour vous aider à configurer les politiques d'autorisation pour vos composants.

Exemple de politique d'autorisation avec accès illimité

L'exemple de politique d'autorisation suivant permet à un composant de publier et de s'abonner à toutes les rubriques.

JSON

```
{
  "accessControl": {
    "aws.greengrass.ipc.mqttproxy": {
      "com.example.MyIoTCorePubSubComponent:mqttproxy:1": {
        "policyDescription": "Allows access to publish/subscribe to all topics.",
        "operations": [
          "aws.greengrass#PublishToIoTCore",
          "aws.greengrass#SubscribeToIoTCore"
        ],
        "resources": [
          "*"
        ]
      }
    }
  }
}
```

YAML

```
---
accessControl:
  aws.greengrass.ipc.mqttproxy:
    com.example.MyIoTCorePubSubComponent:mqttproxy:1:
      policyDescription: Allows access to publish/subscribe to all topics.
      operations:
```

```

- aws.greengrass#PublishToIoTCore
- aws.greengrass#SubscribeToIoTCore
resources:
- "*"

```

Exemple Exemple de politique d'autorisation avec accès limité

L'exemple de politique d'autorisation suivant permet à un composant de publier et de s'abonner à deux rubriques nommées `factory/1/events` et `factory/1/actions`.

JSON

```

{
  "accessControl": {
    "aws.greengrass.ipc.mqttproxy": {
      "com.example.MyIoTCorePubSubComponent:mqttproxy:1": {
        "policyDescription": "Allows access to publish/subscribe to factory 1 topics.",
        "operations": [
          "aws.greengrass#PublishToIoTCore",
          "aws.greengrass#SubscribeToIoTCore"
        ],
        "resources": [
          "factory/1/actions",
          "factory/1/events"
        ]
      }
    }
  }
}

```

YAML

```

---
accessControl:
  aws.greengrass.ipc.mqttproxy:
    "com.example.MyIoTCorePubSubComponent:mqttproxy:1":
      policyDescription: Allows access to publish/subscribe to factory 1 topics.
      operations:
        - aws.greengrass#PublishToIoTCore
        - aws.greengrass#SubscribeToIoTCore
      resources:

```

- factory/1/actions
- factory/1/events

Exemple Exemple de politique d'autorisation pour un groupe de périphériques principaux

Important

Cet exemple utilise une fonctionnalité disponible pour les versions 2.6.0 et ultérieures du composant [Greengrass](#) nucleus. Greengrass nucleus v2.6.0 ajoute la prise en charge de la plupart des [variables de recette](#), notamment dans les configurations de composants{iot:thingName}.

L'exemple de politique d'autorisation suivant permet à un composant de publier et de s'abonner à une rubrique contenant le nom du périphérique principal qui exécute le composant.

JSON

```
{
  "accessControl": {
    "aws.greengrass.ipc.mqttproxy": {
      "com.example.MyIoTCorePubSubComponent:mqttproxy:1": {
        "policyDescription": "Allows access to publish/subscribe to all topics.",
        "operations": [
          "aws.greengrass#PublishToIoTCore",
          "aws.greengrass#SubscribeToIoTCore"
        ],
        "resources": [
          "factory/1/devices/{iot:thingName}/controls"
        ]
      }
    }
  }
}
```

YAML

```
---
accessControl:
  aws.greengrass.ipc.mqttproxy:
    "com.example.MyIoTCorePubSubComponent:mqttproxy:1":
```

```
policyDescription: Allows access to publish/subscribe to all topics.
operations:
  - aws.greengrass#PublishToIoTCore
  - aws.greengrass#SubscribeToIoTCore
resources:
  - factory/1/devices/{iot:thingName}/controls
```

PublishToIoTCore

Publie un message MQTT AWS IoT Core sur un sujet.

Lorsque vous publiez des messages MQTT sur AWS IoT Core, il existe un quota de 100 transactions par seconde. Si vous dépassez ce quota, les messages sont mis en file d'attente pour traitement sur l'appareil Greengrass. Il existe également un quota de 512 Ko de données par seconde et un quota de 20 000 publications par seconde à l'échelle du compte (2 000 dans certains Régions AWS cas). Pour plus d'informations sur les limites du courtier de messages MQTT dans AWS IoT Core, consultez les sections [Limites et quotas du courtier de AWS IoT Core messages et du protocole](#).

Si vous dépassez ces quotas, l'appareil Greengrass limite la publication de messages à. AWS IoT Core Les messages sont stockés dans un spouleur en mémoire. Par défaut, la mémoire allouée au spouleur est de 2,5 Mo. Si le spouleur se remplit, les nouveaux messages sont rejetés. Vous pouvez augmenter la taille du spouleur. Pour plus d'informations, consultez la section [Configuration](#) dans la documentation [Noyau de Greengrass](#). Pour éviter de remplir le spouleur et d'avoir à augmenter la mémoire allouée, limitez les demandes de publication à un maximum de 100 demandes par seconde.

Lorsque votre application doit envoyer des messages à un débit plus élevé ou des messages plus volumineux, pensez à utiliser le [Gestionnaire de flux](#) pour envoyer des messages à Kinesis Data Streams. Le composant du gestionnaire de flux est conçu pour transférer de gros volumes de données vers le AWS Cloud. Pour de plus amples informations, veuillez consulter [Gérez les flux de données sur les appareils principaux de Greengrass](#).

Demande

La demande de cette opération comporte les paramètres suivants :

`topicName`(Python :`topic_name`)

Rubrique dans laquelle le message doit être publié.

qos

Les QoS MQTT à utiliser. Cette énumération possède QoS les valeurs suivantes :

- `AT_MOST_ONCE`— QoS 0. Le message MQTT est délivré au plus une fois.
- `AT_LEAST_ONCE`— QoS 1. Le message MQTT est délivré au moins une fois.

payload

(Facultatif) La charge utile du message sous forme de blob.

Les fonctionnalités suivantes sont disponibles pour la version 2.10.0 et les versions ultérieures [Noyau de Greengrass](#) lors de l'utilisation de MQTT 5. Ces fonctionnalités sont ignorées lorsque vous utilisez MQTT 3.1.1. Le tableau suivant répertorie la version minimale du SDK de l' AWS IoT appareil que vous devez utiliser pour accéder à ces fonctionnalités.

SDK	Version minimale
Kit SDK des appareils AWS IoT pour Python v2	v1.15.0
Kit SDK des appareils AWS IoT pour Java v2	v1.13.0
Kit SDK des appareils AWS IoT pour C++ v2	v1.24.0
Kit SDK des appareils AWS IoT pour JavaScript v2	v1.13.0

payloadFormat

(Facultatif) Format de la charge utile du message. Si vous ne définissez pas `payloadFormat`, le type est supposé être `BYTES`. L'enum possède les valeurs suivantes :

- `BYTES`— Le contenu de la charge utile est un blob binaire.
- `UTF8`— Le contenu de la charge utile est une UTF8 chaîne de caractères.

retain

(Facultatif) Indique s'il faut définir l'option de conservation MQTT sur `true` lors de la publication.

userProperties

(Facultatif) Liste des UserProperty objets spécifiques à l'application à envoyer.
L'UserPropertyobjet est défini comme suit :

```
UserProperty:  
  key: string  
  value: string
```

messageExpiryIntervalSeconds

(Facultatif) Nombre de secondes avant que le message n'expire et ne soit supprimé par le serveur. Si cette valeur n'est pas définie, le message n'expire pas.

correlationData

(Facultatif) Informations ajoutées à la demande qui peuvent être utilisées pour associer une demande à une réponse.

responseTopic

(Facultatif) Rubrique à utiliser pour le message de réponse.

contentType

(Facultatif) Identifiant spécifique à l'application du type de contenu du message.

Réponse

Cette opération ne fournit aucune information dans sa réponse.

Exemples

Les exemples suivants montrent comment appeler cette opération dans le code de composant personnalisé.

Java (IPC client V2)

Exemple Exemple : publier un message

```
package com.aws.greengrass.docs.samples.ipc;  
  
import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClientV2;  
import software.amazon.awssdk.aws.greengrass.model.PublishToIoTCoreRequest;
```

```

import software.amazon.awssdk.aws.greengrass.model.QoS;
import java.nio.charset.StandardCharsets;

public class PublishToIoTCore {

    public static void main(String[] args) {
        String topic = args[0];
        String message = args[1];
        QoS qos = QoS.get(args[2]);

        try (GreengrassCoreIPCClientV2 ipcClientV2 =
GreengrassCoreIPCClientV2.builder().build()) {
            ipcClientV2.publishToIoTCore(new PublishToIoTCoreRequest()
                .withTopicName(topic)
                .withPayload(message.getBytes(StandardCharsets.UTF_8))
                .withQos(qos));
            System.out.println("Successfully published to topic: " + topic);
        } catch (Exception e) {
            System.err.println("Exception occurred.");
            e.printStackTrace();
            System.exit(1);
        }
    }
}

```

Python (IPC client V2)

Exemple Exemple : publier un message

Note

Cet exemple suppose que vous utilisez la version 1.5.4 ou ultérieure de Kit SDK des appareils AWS IoT for Python v2.

```

import awsiot.greengrasscoreipc.clientv2 as clientV2

topic = 'my/topic'
qos = '1'
payload = 'Hello, World'

ipc_client = clientV2.GreengrassCoreIPCClientV2()
resp = ipc_client.publish_to_iot_core(topic_name=topic, qos=qos, payload=payload)

```

```
ipc_client.close()
```

Java (IPC client V1)

Exemple Exemple : publier un message

Note

Cet exemple utilise une `IPCUtils` classe pour créer une connexion au service AWS IoT Greengrass Core IPC. Pour de plus amples informations, veuillez consulter [Connectez-vous au service AWS IoT Greengrass Core IPC](#).

```
package com.aws.greengrass.docs.samples.ipc;

import com.aws.greengrass.docs.samples.ipc.util.IPCUtils;
import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClient;
import software.amazon.awssdk.aws.greengrass.PublishToIoTCoreResponseHandler;
import software.amazon.awssdk.aws.greengrass.model.PublishToIoTCoreRequest;
import software.amazon.awssdk.aws.greengrass.model.PublishToIoTCoreResponse;
import software.amazon.awssdk.aws.greengrass.model.QoS;
import software.amazon.awssdk.aws.greengrass.model.UnauthorizedError;
import software.amazon.awssdk.eventstreamrpc.EventStreamRPCConnection;

import java.nio.charset.StandardCharsets;
import java.util.Optional;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;

public class PublishToIoTCore {

    public static final int TIMEOUT_SECONDS = 10;

    public static void main(String[] args) {
        String topic = args[0];
        String message = args[1];
        QoS qos = QoS.get(args[2]);
        try (EventStreamRPCConnection eventStreamRPCConnection =
            IPCUtils.getEventStreamRpcConnection()) {
            GreengrassCoreIPCClient ipcClient =
```

```

        new GreengrassCoreIPCCClient(eventStreamRPCConnection);
        PublishToIoTCoreResponseHandler responseHandler =
            PublishToIoTCore.publishBinaryMessageToTopic(ipcClient, topic,
message, qos);
        CompletableFuture<PublishToIoTCoreResponse> futureResponse =
            responseHandler.getResponse();
        try {
            futureResponse.get(TIMEOUT_SECONDS, TimeUnit.SECONDS);
            System.out.println("Successfully published to topic: " + topic);
        } catch (TimeoutException e) {
            System.err.println("Timeout occurred while publishing to topic: " +
topic);
        } catch (ExecutionException e) {
            if (e.getCause() instanceof UnauthorizedError) {
                System.err.println("Unauthorized error while publishing to
topic: " + topic);
            } else {
                throw e;
            }
        } catch (InterruptedException e) {
            System.out.println("IPC interrupted.");
        } catch (ExecutionException e) {
            System.err.println("Exception occurred when using IPC.");
            e.printStackTrace();
            System.exit(1);
        }
    }

    public static PublishToIoTCoreResponseHandler
publishBinaryMessageToTopic(GreengrassCoreIPCCClient greengrassCoreIPCCClient, String
topic, String message, QOS qos) {
        PublishToIoTCoreRequest publishToIoTCoreRequest = new
PublishToIoTCoreRequest();
        publishToIoTCoreRequest.setTopicName(topic);

        publishToIoTCoreRequest.setPayload(message.getBytes(StandardCharsets.UTF_8));
        publishToIoTCoreRequest.setQos(qos);
        return greengrassCoreIPCCClient.publishToIoTCore(publishToIoTCoreRequest,
Optional.empty());
    }
}

```

Python (IPC client V1)

Exemple Exemple : publier un message

Note

Cet exemple suppose que vous utilisez la version 1.5.4 ou ultérieure de Kit SDK des appareils AWS IoT for Python v2.

```
import awsiot.greengrasscoreipc
import awsiot.greengrasscoreipc.client as client
from awsiot.greengrasscoreipc.model import (
    QOS,
    PublishToIoTCoreRequest
)

TIMEOUT = 10

ipc_client = awsiot.greengrasscoreipc.connect()

topic = "my/topic"
message = "Hello, World"
qos = QOS.AT_LEAST_ONCE

request = PublishToIoTCoreRequest()
request.topic_name = topic
request.payload = bytes(message, "utf-8")
request.qos = qos
operation = ipc_client.new_publish_to_iot_core()
operation.activate(request)
future_response = operation.get_response()
future_response.result(TIMEOUT)
```

C++

Exemple Exemple : publier un message

```
#include <iostream>

#include <aws/crt/Api.h>
#include <aws/greengrass/GreengrassCoreIpcClient.h>
```

```

using namespace Aws::Crt;
using namespace Aws::Greengrass;

class IpcClientLifecycleHandler : public ConnectionLifecycleHandler {
    void OnConnectCallback() override {
        // Handle connection to IPC service.
    }

    void OnDisconnectCallback(RpcError error) override {
        // Handle disconnection from IPC service.
    }

    bool OnErrorCallback(RpcError error) override {
        // Handle IPC service connection error.
        return true;
    }
};

int main() {
    ApiHandle apiHandle(g_allocator);
    Io::EventLoopGroup eventLoopGroup(1);
    Io::DefaultHostResolver socketResolver(eventLoopGroup, 64, 30);
    Io::ClientBootstrap bootstrap(eventLoopGroup, socketResolver);
    IpcClientLifecycleHandler ipcLifecycleHandler;
    GreengrassCoreIpcClient ipcClient(bootstrap);
    auto connectionStatus = ipcClient.Connect(ipcLifecycleHandler).get();
    if (!connectionStatus) {
        std::cerr << "Failed to establish IPC connection: " <<
connectionStatus.StatusToString() << std::endl;
        exit(-1);
    }

    String message("Hello, World!");
    String topic("my/topic");
    QoS qos = QoS_AT_MOST_ONCE;
    int timeout = 10;

    PublishToIoTCoreRequest request;
    Vector<uint8_t> messageData({message.begin(), message.end()});
    request.SetTopicName(topic);
    request.SetPayload(messageData);
    request.SetQos(qos);

    auto operation = ipcClient.NewPublishToIoTCore();

```

```

    auto activate = operation->Activate(request, nullptr);
    activate.wait();

    auto responseFuture = operation->GetResult();
    if (responseFuture.wait_for(std::chrono::seconds(timeout)) ==
std::future_status::timeout) {
        std::cerr << "Operation timed out while waiting for response from Greengrass
Core." << std::endl;
        exit(-1);
    }

    auto response = responseFuture.get();
    if (!response) {
        // Handle error.
        auto errorType = response.GetResultType();
        if (errorType == OPERATION_ERROR) {
            auto *error = response.GetOperationError();
            (void)error;
            // Handle operation error.
        } else {
            // Handle RPC error.
        }
    }

    return 0;
}

```

JavaScript

Exemple Exemple : publier un message

```

import * as greengrasscoreipc from "aws-iot-device-sdk-v2/dist/greengrasscoreipc";
import {QOS, PublishToIoTCoreRequest} from "aws-iot-device-sdk-v2/dist/
greengrasscoreipc/model";

class PublishToIoTCore {
    private ipcClient: greengrasscoreipc.Client
    private readonly topic: string;

    constructor() {
        // define your own constructor, e.g.
        this.topic = "<define_your_topic>";
        this.publishToIoTCore().then(r => console.log("Started workflow"));
    }
}

```

```
    }

    private async publishToIoTCore() {
      try {
        const request: PublishToIoTCoreRequest = {
          topicName: this.topic,
          qos: QOS.AT_LEAST_ONCE, // you can change this depending on your use
case
        }

        this.ipcClient = await getIpClient();

        await this.ipcClient.publishToIoTCore(request);
      } catch (e) {
        // parse the error depending on your use cases
        throw e
      }
    }
  }

export async function getIpClient(){
  try {
    const ipcClient = greengrasscoreipc.createClient();
    await ipcClient.connect()
      .catch(error => {
        // parse the error depending on your use cases
        throw error;
      });
    return ipcClient
  } catch (err) {
    // parse the error depending on your use cases
    throw err
  }
}

// starting point
const publishToIoTCore = new PublishToIoTCore();
```


SubscribeToIoTCore

Abonnez-vous aux messages MQTT à partir d'un AWS IoT Core sujet ou d'un filtre de sujet. Le logiciel AWS IoT Greengrass Core supprime les abonnements lorsque le composant atteint la fin de son cycle de vie.

Il s'agit d'une opération d'abonnement dans le cadre de laquelle vous vous abonnez à un flux de messages d'événements. Pour utiliser cette opération, définissez un gestionnaire de réponse au flux avec des fonctions qui gèrent les messages d'événements, les erreurs et la fermeture du flux. Pour de plus amples informations, veuillez consulter [Abonnez-vous aux diffusions d'événements IPC](#).

Type de message d'événement : `IoTCoreMessage`

Demande

La demande de cette opération comporte les paramètres suivants :

`topicName`(Python :`topic_name`)

Rubrique à laquelle vous souhaitez vous abonner. Vous pouvez utiliser les caractères génériques des rubriques MQTT (`#`et`+`) pour vous abonner à plusieurs rubriques.

`qos`

Les QoS MQTT à utiliser. Cette énumération possède QOS les valeurs suivantes :

- `AT_MOST_ONCE`— QoS 0. Le message MQTT est délivré au plus une fois.
- `AT_LEAST_ONCE`— QoS 1. Le message MQTT est délivré au moins une fois.

Réponse

La réponse de cette opération contient les informations suivantes :

`messages`

Le flux de messages MQTT. Cet objet contient `IoTCoreMessage` les informations suivantes :

`message`

Le message MQTT. Cet objet contient `MQTTMessage` les informations suivantes :

`topicName`(Python :`topic_name`)

Sujet dans lequel le message a été publié.

payload

(Facultatif) La charge utile du message sous forme de blob.

Les fonctionnalités suivantes sont disponibles pour la version 2.10.0 et les versions ultérieures [Noyau de Greengrass](#) lors de l'utilisation de MQTT 5. Ces fonctionnalités sont ignorées lorsque vous utilisez MQTT 3.1.1. Le tableau suivant répertorie la version minimale du SDK de l'AWS IoT appareil que vous devez utiliser pour accéder à ces fonctionnalités.

SDK	Version minimale
Kit SDK des appareils AWS IoT pour Python v2	v1.15.0
Kit SDK des appareils AWS IoT pour Java v2	v1.13.0
Kit SDK des appareils AWS IoT pour C++ v2	v1.24.0
Kit SDK des appareils AWS IoT pour JavaScript v2	v1.13.0

payloadFormat

(Facultatif) Format de la charge utile du message. Si vous ne définissez pas `payloadFormat`, le type est supposé être `BYTES`. L'enum possède les valeurs suivantes :

- `BYTES`— Le contenu de la charge utile est un blob binaire.
- `UTF8`— Le contenu de la charge utile est une UTF8 chaîne de caractères.

retain

(Facultatif) Indique s'il faut définir l'option de conservation MQTT sur `true` lors de la publication.

userProperties

(Facultatif) Liste des `UserProperty` objets spécifiques à l'application à envoyer. L'`UserProperty` objet est défini comme suit :

```
UserProperty:  
  key: string  
  value: string
```

messageExpiryIntervalSeconds

(Facultatif) Nombre de secondes avant que le message n'expire et ne soit supprimé par le serveur. Si cette valeur n'est pas définie, le message n'expire pas.

correlationData

(Facultatif) Informations ajoutées à la demande qui peuvent être utilisées pour associer une demande à une réponse.

responseTopic

(Facultatif) Rubrique à utiliser pour le message de réponse.

contentType

(Facultatif) Identifiant spécifique à l'application du type de contenu du message.

Exemples

Les exemples suivants montrent comment appeler cette opération dans le code de composant personnalisé.

Java (IPC client V2)

Exemple Exemple : s'abonner à des messages

```
package com.aws.greengrass.docs.samples.ipc;  
  
import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClientV2;  
import software.amazon.awssdk.aws.greengrass.SubscribeToIoTCoreResponseHandler;  
import software.amazon.awssdk.aws.greengrass.model.QOS;  
import software.amazon.awssdk.aws.greengrass.model.IoTCoreMessage;  
import software.amazon.awssdk.aws.greengrass.model.SubscribeToIoTCoreRequest;  
import software.amazon.awssdk.aws.greengrass.model.SubscribeToIoTCoreResponse;  
  
import java.nio.charset.StandardCharsets;  
import java.util.Optional;  
import java.util.function.Consumer;  
import java.util.function.Function;
```

```
public class SubscribeToIoTCore {

    public static void main(String[] args) {
        String topic = args[0];
        QoS qos = QoS.get(args[1]);

        Consumer<IoTCoreMessage> onStreamEvent = iotCoreMessage ->
            System.out.printf("Received new message on topic %s: %s%n",
                iotCoreMessage.getMessage().getTopicName(),
                new String(iotCoreMessage.getMessage().getPayload(),
StandardCharsets.UTF_8));

        Optional<Function<Throwable, Boolean>> onStreamError =
            Optional.of(e -> {
                System.err.println("Received a stream error.");
                e.printStackTrace();
                return false;
            });

        Optional<Runnable> onStreamClosed = Optional.of(() ->
            System.out.println("Subscribe to IoT Core stream closed.));

        try (GreengrassCoreIPCClientV2 ipcClientV2 =
GreengrassCoreIPCClientV2.builder().build()) {
            SubscribeToIoTCoreRequest request = new SubscribeToIoTCoreRequest()
                .withTopicName(topic)
                .withQos(qos);

            GreengrassCoreIPCClientV2.StreamingResponse<SubscribeToIoTCoreResponse,
SubscribeToIoTCoreResponseHandler>
                streamingResponse = ipcClientV2.subscribeToIoTCore(request,
onStreamEvent, onStreamError, onStreamClosed);

            streamingResponse.getResponse();
            System.out.println("Successfully subscribed to topic: " + topic);

            // Keep the main thread alive, or the process will exit.
            while (true) {
                Thread.sleep(10000);
            }

            // To stop subscribing, close the stream.
```

```

        streamingResponse.getHandler().closeStream();
    } catch (InterruptedException e) {
        System.out.println("Subscribe interrupted.");
    } catch (Exception e) {
        System.err.println("Exception occurred.");
        e.printStackTrace();
        System.exit(1);
    }
}
}
}

```

Python (IPC client V2)

Exemple Exemple : s'abonner à des messages

Note

Cet exemple suppose que vous utilisez la version 1.5.4 ou ultérieure de Kit SDK des appareils AWS IoT for Python v2.

```

import threading
import traceback

import awsiot.greengrasscoreipc.clientv2 as clientV2

topic = 'my/topic'
qos = '1'

def on_stream_event(event):
    try:
        topic_name = event.message.topic_name
        message = str(event.message.payload, 'utf-8')
        print(f'Received new message on topic {topic_name}: {message}')
    except:
        traceback.print_exc()

def on_stream_error(error):
    # Return True to close stream, False to keep stream open.
    return True

def on_stream_closed():

```

```

pass

ipc_client = clientV2.GreengrassCoreIPCClientV2()
resp, operation = ipc_client.subscribe_to_iot_core(
    topic_name=topic,
    qos=qos,
    on_stream_event=on_stream_event,
    on_stream_error=on_stream_error,
    on_stream_closed=on_stream_closed
)

# Keep the main thread alive, or the process will exit.
event = threading.Event()
event.wait()

# To stop subscribing, close the operation stream.
operation.close()
ipc_client.close()

```

Java (IPC client V1)

Exemple Exemple : s'abonner à des messages

Note

Cet exemple utilise une `IPCUtils` classe pour créer une connexion au service AWS IoT Greengrass Core IPC. Pour de plus amples informations, veuillez consulter [Connectez-vous au service AWS IoT Greengrass Core IPC](#).

```

package com.aws.greengrass.docs.samples.ipc;

import com.aws.greengrass.docs.samples.ipc.util.IPCUtils;
import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClient;
import software.amazon.awssdk.aws.greengrass.SubscribeToIoTCoreResponseHandler;
import software.amazon.awssdk.aws.greengrass.model.*;
import software.amazon.awssdk.eventstreamrpc.EventStreamRPCConnection;
import software.amazon.awssdk.eventstreamrpc.StreamResponseHandler;

import java.nio.charset.StandardCharsets;
import java.util.Optional;
import java.util.concurrent.CompletableFuture;

```

```
import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;

public class SubscribeToIoTCore {

    public static final int TIMEOUT_SECONDS = 10;

    public static void main(String[] args) {
        String topic = args[0];
        QoS qos = QoS.get(args[1]);
        try (EventStreamRPCConnection eventStreamRPCConnection =
            IPCUtils.getEventStreamRpcConnection()) {
            GreengrassCoreIPCClient ipcClient =
                new GreengrassCoreIPCClient(eventStreamRPCConnection);
            StreamResponseHandler<IoTCoreMessage> streamResponseHandler =
                new SubscriptionResponseHandler();
            SubscribeToIoTCoreResponseHandler responseHandler =
                SubscribeToIoTCore.subscribeToIoTCore(ipcClient, topic, qos,
                    streamResponseHandler);
            CompletableFuture<SubscribeToIoTCoreResponse> futureResponse =
                responseHandler.getResponse();
            try {
                futureResponse.get(TIMEOUT_SECONDS, TimeUnit.SECONDS);
                System.out.println("Successfully subscribed to topic: " + topic);
            } catch (TimeoutException e) {
                System.err.println("Timeout occurred while subscribing to topic: " +
topic);
            } catch (ExecutionException e) {
                if (e.getCause() instanceof UnauthorizedError) {
                    System.err.println("Unauthorized error while subscribing to
topic: " + topic);
                } else {
                    throw e;
                }
            }
        }

        // Keep the main thread alive, or the process will exit.
        try {
            while (true) {
                Thread.sleep(10000);
            }
        } catch (InterruptedException e) {
            System.out.println("Subscribe interrupted.");
        }
    }
}
```

```

    }

    // To stop subscribing, close the stream.
    responseHandler.closeStream();
} catch (InterruptedException e) {
    System.out.println("IPC interrupted.");
} catch (ExecutionException e) {
    System.err.println("Exception occurred when using IPC.");
    e.printStackTrace();
    System.exit(1);
}
}

public static SubscribeToIoTCoreResponseHandler
subscribeToIoTCore(GreengrassCoreIPCClient greengrassCoreIPCClient, String topic,
QoS qos, StreamResponseHandler<IoTCoreMessage> streamResponseHandler) {
    SubscribeToIoTCoreRequest subscribeToIoTCoreRequest = new
SubscribeToIoTCoreRequest();
    subscribeToIoTCoreRequest.setTopicName(topic);
    subscribeToIoTCoreRequest.setQos(qos);
    return
greengrassCoreIPCClient.subscribeToIoTCore(subscribeToIoTCoreRequest,
Optional.of(streamResponseHandler));
}

public static class SubscriptionResponseHandler implements
StreamResponseHandler<IoTCoreMessage> {

    @Override
    public void onStreamEvent(IoTCoreMessage iotCoreMessage) {
        try {
            String topic = iotCoreMessage.getMessage().getTopicName();
            String message = new
String(iotCoreMessage.getMessage().getPayload(),
StandardCharsets.UTF_8);
            System.out.printf("Received new message on topic %s: %s\n", topic,
message);
        } catch (Exception e) {
            System.err.println("Exception occurred while processing subscription
response " +
                "message.");
            e.printStackTrace();
        }
    }
}

```



```

    @Override
    public boolean onStreamError(Throwable error) {
        System.err.println("Received a stream error.");
        error.printStackTrace();
        return false;
    }

    @Override
    public void onStreamClosed() {
        System.out.println("Subscribe to IoT Core stream closed.");
    }
}
}

```

Python (IPC client V1)

Exemple Exemple : s'abonner à des messages

Note

Cet exemple suppose que vous utilisez la version 1.5.4 ou ultérieure de Kit SDK des appareils AWS IoT for Python v2.

```

import time
import traceback

import awsiot.greengrasscoreipc
import awsiot.greengrasscoreipc.client as client
from awsiot.greengrasscoreipc.model import (
    IoTCoreMessage,
    QOS,
    SubscribeToIoTCoreRequest
)

TIMEOUT = 10

ipc_client = awsiot.greengrasscoreipc.connect()

class StreamHandler(client.SubscribeToIoTCoreStreamHandler):
    def __init__(self):
        super().__init__()

```

```

def on_stream_event(self, event: IoTCoreMessage) -> None:
    try:
        message = str(event.message.payload, "utf-8")
        topic_name = event.message.topic_name
        # Handle message.
    except:
        traceback.print_exc()

def on_stream_error(self, error: Exception) -> bool:
    # Handle error.
    return True # Return True to close stream, False to keep stream open.

def on_stream_closed(self) -> None:
    # Handle close.
    pass

topic = "my/topic"
qos = QOS.AT_MOST_ONCE

request = SubscribeToIoTCoreRequest()
request.topic_name = topic
request.qos = qos
handler = StreamHandler()
operation = ipc_client.new_subscribe_to_iot_core(handler)
operation.activate(request)
future_response = operation.get_response()
future_response.result(TIMEOUT)

# Keep the main thread alive, or the process will exit.
while True:
    time.sleep(10)

# To stop subscribing, close the operation stream.
operation.close()

```

C++

Exemple Exemple : s'abonner à des messages

```

#include <iostream>

#include <aws/crt/Api.h>

```

```
#include <aws/greengrass/GreengrassCoreIpcClient.h>

using namespace Aws::Crt;
using namespace Aws::Greengrass;

class IoTCoreResponseHandler : public SubscribeToIoTCoreStreamHandler {

public:
    virtual ~IoTCoreResponseHandler() {}

private:
    void OnStreamEvent(IoTCoreMessage *response) override {
        auto message = response->GetMessage();
        if (message.has_value() && message.value().GetPayload().has_value()) {
            auto messageBytes = message.value().GetPayload().value();
            std::string messageString(messageBytes.begin(), messageBytes.end());
            std::string topicName =
message.value().GetTopicName().value().c_str();
            // Handle message.
        }
    }

    bool OnStreamError(OperationError *error) override {
        // Handle error.
        return false; // Return true to close stream, false to keep stream open.
    }

    void OnStreamClosed() override {
        // Handle close.
    }
};

class IpcClientLifecycleHandler : public ConnectionLifecycleHandler {
    void OnConnectCallback() override {
        // Handle connection to IPC service.
    }

    void OnDisconnectCallback(RpcError error) override {
        // Handle disconnection from IPC service.
    }

    bool OnErrorCallback(RpcError error) override {
        // Handle IPC service connection error.
        return true;
    }
};
```

```

    }
};

int main() {
    ApiHandle apiHandle(g_allocator);
    Io::EventLoopGroup eventLoopGroup(1);
    Io::DefaultHostResolver socketResolver(eventLoopGroup, 64, 30);
    Io::ClientBootstrap bootstrap(eventLoopGroup, socketResolver);
    IpcClientLifecycleHandler ipcLifecycleHandler;
    GreengrassCoreIpcClient ipcClient(bootstrap);
    auto connectionStatus = ipcClient.Connect(ipcLifecycleHandler).get();
    if (!connectionStatus) {
        std::cerr << "Failed to establish IPC connection: " <<
connectionStatus.StatusToString() << std::endl;
        exit(-1);
    }

    String topic("my/topic");
    QoS qos = QoS_AT_MOST_ONCE;
    int timeout = 10;

    SubscribeToIoTCoreRequest request;
    request.SetTopicName(topic);
    request.SetQos(qos);
    auto streamHandler = MakeShared<IoTCoreResponseHandler>(DefaultAllocator());
    auto operation = ipcClient.NewSubscribeToIoTCore(streamHandler);
    auto activate = operation->Activate(request, nullptr);
    activate.wait();

    auto responseFuture = operation->GetResult();
    if (responseFuture.wait_for(std::chrono::seconds(timeout)) ==
std::future_status::timeout) {
        std::cerr << "Operation timed out while waiting for response from Greengrass
Core." << std::endl;
        exit(-1);
    }

    auto response = responseFuture.get();
    if (!response) {
        // Handle error.
        auto errorType = response.GetResultType();
        if (errorType == OPERATION_ERROR) {
            auto *error = response.GetOperationError();
            (void)error;

```

```

        // Handle operation error.
    } else {
        // Handle RPC error.
    }
    exit(-1);
}

// Keep the main thread alive, or the process will exit.
while (true) {
    std::this_thread::sleep_for(std::chrono::seconds(10));
}

operation->Close();
return 0;
}

```

JavaScript

Exemple Exemple : s'abonner à des messages

```

import * as greengrasscoreipc from "aws-iot-device-sdk-v2/dist/greengrasscoreipc";
import {IoTCoreMessage, QOS, SubscribeToIoTCoreRequest} from "aws-iot-device-sdk-v2/dist/greengrasscoreipc/model";
import {RpcError} from "aws-iot-device-sdk-v2/dist/eventstream_rpc";

class SubscribeToIoTCore {
    private ipcClient: greengrasscoreipc.Client
    private readonly topic: string;

    constructor() {
        // define your own constructor, e.g.
        this.topic = "<define_your_topic>";
        this.subscribeToIoTCore().then(r => console.log("Started workflow"));
    }

    private async subscribeToIoTCore() {
        try {
            const request: SubscribeToIoTCoreRequest = {
                topicName: this.topic,
                qos: QOS.AT_LEAST_ONCE, // you can change this depending on your use
            };
        }
    }
}

```

```

    this.ipcClient = await getIpcClient();

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

    streamingOperation.on('message', (message: IoTCoreMessage) => {
        // parse the message depending on your use cases, e.g.
        if (message.message && message.message.payload) {
            const receivedMessage = message.message.payload.toString();
        }
    });

    streamingOperation.on('streamError', (error : RpcError) => {
        // define your own error handling logic
    });

    streamingOperation.on('ended', () => {
        // define your own logic
    });

    await streamingOperation.activate();

    // Keep the main thread alive, or the process will exit.
    await new Promise((resolve) => setTimeout(resolve, 10000))
} catch (e) {
    // parse the error depending on your use cases
    throw e
}
}

export async function getIpcClient(){
    try {
        const ipcClient = greengrasscoreipc.createClient();
        await ipcClient.connect()
            .catch(error => {
                // parse the error depending on your use cases
                throw error;
            });
        return ipcClient
    } catch (err) {
        // parse the error depending on your use cases
        throw err
    }
}

```

```
// starting point
const subscribeToIoTCore = new SubscribeToIoTCore();
```

Exemples

Utilisez les exemples suivants pour apprendre à utiliser le service AWS IoT Core MQTT IPC dans vos composants.

Exemple d'éditeur AWS IoT Core MQTT (C++)

L'exemple de recette suivant permet au composant de publier dans toutes les rubriques.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.IoTCorePublisherCpp",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that publishes MQTT messages to IoT Core.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.mqttproxy": {
          "com.example.IoTCorePublisherCpp:mqttproxy:1": {
            "policyDescription": "Allows access to publish to all topics.",
            "operations": [
              "aws.greengrass#PublishToIoTCore"
            ],
            "resources": [
              "*"
            ]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Lifecycle": {
        "Run": "{artifacts:path}/greengrassv2_iotcore_publisher"
```

```

    },
    "Artifacts": [
      {
        "URI": "s3://amzn-s3-demo-bucket/artifacts/
com.example.IoTCorePublisherCpp/1.0.0/greengrassv2_iotcore_publisher",
        "Permission": {
          "Execute": "OWNER"
        }
      }
    ]
  }
}

```

YAML

```

---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.IoTCorePublisherCpp
ComponentVersion: 1.0.0
ComponentDescription: A component that publishes MQTT messages to IoT Core.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    accessControl:
      aws.greengrass.ipc.mqttproxy:
        com.example.IoTCorePublisherCpp:mqttproxy:1:
          policyDescription: Allows access to publish to all topics.
          operations:
            - aws.greengrass#PublishToIoTCore
          resources:
            - "*"
Manifests:
  - Lifecycle:
      Run: "{artifacts:path}/greengrassv2_iotcore_publisher"
    Artifacts:
      - URI: s3://amzn-s3-demo-bucket/artifacts/
com.example.IoTCorePublisherCpp/1.0.0/greengrassv2_iotcore_publisher
      Permission:
        Execute: OWNER

```


L'exemple d'application C++ suivant montre comment utiliser le service AWS IoT Core MQTT IPC pour publier des messages sur AWS IoT Core

```
#include <iostream>

#include <aws/crt/Api.h>
#include <aws/greengrass/GreengrassCoreIpcClient.h>

using namespace Aws::Crt;
using namespace Aws::Greengrass;

class IpcClientLifecycleHandler : public ConnectionLifecycleHandler {
    void OnConnectCallback() override {
        std::cout << "OnConnectCallback" << std::endl;
    }

    void OnDisconnectCallback(RpcError error) override {
        std::cout << "OnDisconnectCallback: " << error.StatusToString() << std::endl;
        exit(-1);
    }

    bool OnErrorCallback(RpcError error) override {
        std::cout << "OnErrorCallback: " << error.StatusToString() << std::endl;
        return true;
    }
};

int main() {
    String message("Hello from the Greengrass IPC MQTT publisher (C++).");
    String topic("test/topic/cpp");
    QOS qos = QOS_AT_LEAST_ONCE;
    int timeout = 10;

    ApiHandle apiHandle(g_allocator);
    Io::EventLoopGroup eventLoopGroup(1);
    Io::DefaultHostResolver socketResolver(eventLoopGroup, 64, 30);
    Io::ClientBootstrap bootstrap(eventLoopGroup, socketResolver);
    IpcClientLifecycleHandler ipcLifecycleHandler;
    GreengrassCoreIpcClient ipcClient(bootstrap);
    auto connectionStatus = ipcClient.Connect(ipcLifecycleHandler).get();
    if (!connectionStatus) {
        std::cerr << "Failed to establish IPC connection: " <<
        connectionStatus.StatusToString() << std::endl;
    }
}
```

```

        exit(-1);
    }

    while (true) {
        PublishToIoTCoreRequest request;
        Vector<uint8_t> messageData({message.begin(), message.end()});
        request.SetTopicName(topic);
        request.SetPayload(messageData);
        request.SetQos(qos);

        auto operation = ipcClient.NewPublishToIoTCore();
        auto activate = operation->Activate(request, nullptr);
        activate.wait();

        auto responseFuture = operation->GetResult();
        if (responseFuture.wait_for(std::chrono::seconds(timeout)) ==
std::future_status::timeout) {
            std::cerr << "Operation timed out while waiting for response from
Greengrass Core." << std::endl;
            exit(-1);
        }

        auto response = responseFuture.get();
        if (response) {
            std::cout << "Successfully published to topic: " << topic << std::endl;
        } else {
            // An error occurred.
            std::cout << "Failed to publish to topic: " << topic << std::endl;
            auto errorType = response.GetResultType();
            if (errorType == OPERATION_ERROR) {
                auto *error = response.GetOperationError();
                std::cout << "Operation error: " << error->GetMessage().value() <<
std::endl;
            } else {
                std::cout << "RPC error: " << response.GetRpcError() << std::endl;
            }
            exit(-1);
        }

        std::this_thread::sleep_for(std::chrono::seconds(5));
    }

    return 0;

```

```
}
```

Exemple d'abonné AWS IoT Core MQTT (C++)

L'exemple de recette suivant permet au composant de s'abonner à toutes les rubriques.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.IoTCoreSubscriberCpp",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that subscribes to MQTT messages from IoT
Core.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.mqttproxy": {
          "com.example.IoTCoreSubscriberCpp:mqttproxy:1": {
            "policyDescription": "Allows access to subscribe to all topics.",
            "operations": [
              "aws.greengrass#SubscribeToIoTCore"
            ],
            "resources": [
              "*"
            ]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Lifecycle": {
        "Run": "{artifacts:path}/greengrassv2_iotcore_subscriber"
      },
      "Artifacts": [
        {
          "URI": "s3://amzn-s3-demo-bucket/artifacts/
com.example.IoTCoreSubscriberCpp/1.0.0/greengrassv2_iotcore_subscriber",
          "Permission": {
            "Execute": "OWNER"
          }
        }
      ]
    }
  ]
}
```

```

    }
  ]
}
]
}

```

YAML

```

---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.IoTCoreSubscriberCpp
ComponentVersion: 1.0.0
ComponentDescription: A component that subscribes to MQTT messages from IoT Core.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    accessControl:
      aws.greengrass.ipc.mqttproxy:
        com.example.IoTCoreSubscriberCpp:mqttproxy:1:
          policyDescription: Allows access to subscribe to all topics.
          operations:
            - aws.greengrass#SubscribeToIoTCore
          resources:
            - "*"
Manifests:
  - Lifecycle:
      Run: "{artifacts:path}/greengrassv2_iotcore_subscriber"
    Artifacts:
      - URI: s3://amzn-s3-demo-bucket/artifacts/
        com.example.IoTCoreSubscriberCpp/1.0.0/greengrassv2_iotcore_subscriber
      Permission:
        Execute: OWNER

```

L'exemple d'application C++ suivant montre comment utiliser le service AWS IoT Core MQTT IPC pour s'abonner à des messages provenant de. AWS IoT Core

```

#include <iostream>

#include <aws/crt/Api.h>
#include <aws/greengrass/GreengrassCoreIpcClient.h>

using namespace Aws::Crt;

```

```
using namespace Aws::Greengrass;

class IoTCoreResponseHandler : public SubscribeToIoTCoreStreamHandler {

public:
    virtual ~IoTCoreResponseHandler() {}

private:

    void OnStreamEvent(IoTCoreMessage *response) override {
        auto message = response->GetMessage();
        if (message.has_value() && message.value().GetPayload().has_value()) {
            auto messageBytes = message.value().GetPayload().value();
            std::string messageString(messageBytes.begin(), messageBytes.end());
            std::string messageTopic =
message.value().GetTopicName().value().c_str();
            std::cout << "Received new message on topic: " << messageTopic <<
std::endl;
            std::cout << "Message: " << messageString << std::endl;
        }
    }

    bool OnStreamError(OperationError *error) override {
        std::cout << "Received an operation error: ";
        if (error->GetMessage().has_value()) {
            std::cout << error->GetMessage().value();
        }
        std::cout << std::endl;
        return false; // Return true to close stream, false to keep stream open.
    }

    void OnStreamClosed() override {
        std::cout << "Subscribe to IoT Core stream closed." << std::endl;
    }
};

class IpcClientLifecycleHandler : public ConnectionLifecycleHandler {
    void OnConnectCallback() override {
        std::cout << "OnConnectCallback" << std::endl;
    }

    void OnDisconnectCallback(RpcError error) override {
        std::cout << "OnDisconnectCallback: " << error.StatusToString() << std::endl;
        exit(-1);
    }
};
```

```

    }

    bool OnErrorCallback(RpcError error) override {
        std::cout << "OnErrorCallback: " << error.StatusToString() << std::endl;
        return true;
    }
};

int main() {
    String topic("test/topic/cpp");
    QOS qos = QOS_AT_LEAST_ONCE;
    int timeout = 10;

    ApiHandle apiHandle(g_allocator);
    Io::EventLoopGroup eventLoopGroup(1);
    Io::DefaultHostResolver socketResolver(eventLoopGroup, 64, 30);
    Io::ClientBootstrap bootstrap(eventLoopGroup, socketResolver);
    IpcClientLifecycleHandler ipcLifecycleHandler;
    GreengrassCoreIpcClient ipcClient(bootstrap);
    auto connectionStatus = ipcClient.Connect(ipcLifecycleHandler).get();
    if (!connectionStatus) {
        std::cerr << "Failed to establish IPC connection: " <<
connectionStatus.StatusToString() << std::endl;
        exit(-1);
    }

    SubscribeToIoTCoreRequest request;
    request.SetTopicName(topic);
    request.SetQos(qos);
    auto streamHandler = MakeShared<IoTCoreResponseHandler>(DefaultAllocator());
    auto operation = ipcClient.NewSubscribeToIoTCore(streamHandler);
    auto activate = operation->Activate(request, nullptr);
    activate.wait();

    auto responseFuture = operation->GetResult();
    if (responseFuture.wait_for(std::chrono::seconds(timeout)) ==
std::future_status::timeout) {
        std::cerr << "Operation timed out while waiting for response from Greengrass
Core." << std::endl;
        exit(-1);
    }

    auto response = responseFuture.get();
    if (response) {

```

```
        std::cout << "Successfully subscribed to topic: " << topic << std::endl;
    } else {
        // An error occurred.
        std::cout << "Failed to subscribe to topic: " << topic << std::endl;
        auto errorType = response.GetResultType();
        if (errorType == OPERATION_ERROR) {
            auto *error = response.GetOperationError();
            std::cout << "Operation error: " << error->GetMessage().value() <<
std::endl;
        } else {
            std::cout << "RPC error: " << response.GetRpcError() << std::endl;
        }
        exit(-1);
    }

    // Keep the main thread alive, or the process will exit.
    while (true) {
        std::this_thread::sleep_for(std::chrono::seconds(10));
    }

    operation->Close();
    return 0;
}
```

Interagir avec le cycle de vie des composants

Utilisez le service IPC du cycle de vie des composants pour :

- Mettez à jour l'état du composant sur le périphérique principal.
- Abonnez-vous aux mises à jour de l'état des composants.
- Empêchez le noyau d'arrêter le composant pour appliquer une mise à jour lors d'un déploiement.
- Suspendez et reprenez les processus relatifs aux composants.

Rubriques

- [Versions minimales du SDK](#)
- [Autorisation](#)
- [UpdateState](#)
- [SubscribeToComponentUpdates](#)

- [DeferComponentUpdate](#)
- [PauseComponent](#)
- [ResumeComponent](#)

Versions minimales du SDK

Le tableau suivant répertorie les versions minimales Kit SDK des appareils AWS IoT que vous devez utiliser pour interagir avec le cycle de vie des composants.

SDK	Version minimale	
Kit SDK des appareils AWS IoT pour Java v2	v1.2.10	
Kit SDK des appareils AWS IoT pour Python v2	v1.5.3	
Kit SDK des appareils AWS IoT pour C++ v2	v1.17.0	
Kit SDK des appareils AWS IoT pour JavaScript v2	v1.12.0	

Autorisation

Pour suspendre ou reprendre d'autres composants d'un composant personnalisé, vous devez définir des politiques d'autorisation qui permettent à votre composant de gérer d'autres composants. Pour plus d'informations sur la définition des politiques d'autorisation, consultez [Autoriser les composants à effectuer des opérations IPC](#).

Les politiques d'autorisation pour la gestion du cycle de vie des composants présentent les propriétés suivantes.

Identifiant du service IPC : `aws.greengrass.ipc.lifecycle`

Opération	Description	Ressources
<code>aws.greengrass#PauseComponent</code>	Permet à un composant de suspendre les composants que vous spécifiez.	Un nom de composant, ou * pour autoriser l'accès à tous les composants.
<code>aws.greengrass#ResumeComponent</code>	Permet à un composant de reprendre les composants que vous spécifiez.	Un nom de composant, ou * pour autoriser l'accès à tous les composants.
*	Permet à un composant de suspendre et de reprendre les composants que vous spécifiez.	Un nom de composant, ou * pour autoriser l'accès à tous les composants.

Exemples de politiques d'autorisation

Vous pouvez vous référer à l'exemple de politique d'autorisation suivant pour vous aider à configurer les politiques d'autorisation pour vos composants.

Exemple Exemple de politique d'autorisation

L'exemple de politique d'autorisation suivant permet à un composant de suspendre et de reprendre tous les composants.

```
{
  "accessControl": {
    "aws.greengrass.ipc.lifecycle": {
      "com.example.MyLocalLifecycleComponent:lifecycle:1": {
        "policyDescription": "Allows access to pause/resume all components.",
        "operations": [
          "aws.greengrass#PauseComponent",
          "aws.greengrass#ResumeComponent"
        ],
        "resources": [
          "*"
        ]
      }
    }
  }
}
```

```
}
```

UpdateState

Mettez à jour l'état du composant sur le périphérique principal.

Demande

La demande de cette opération comporte les paramètres suivants :

`state`

État à définir. Cette énumération possède `LifecycleState` les valeurs suivantes :

- `RUNNING`
- `ERRORED`

Réponse

Cette opération ne fournit aucune information dans sa réponse.

SubscribeToComponentUpdates

Abonnez-vous pour recevoir des notifications avant que le logiciel AWS IoT Greengrass Core ne mette à jour un composant. La notification indique si le noyau redémarrera ou non dans le cadre de la mise à jour.

Le noyau envoie des notifications de mise à jour uniquement si la politique de mise à jour des composants du déploiement indique de notifier les composants. Le comportement par défaut consiste à notifier les composants. Pour plus d'informations, reportez-vous à la section [Créer des déploiements](#) et à l'[DeploymentComponentUpdatePolicy](#) objet que vous pouvez fournir lorsque vous appelez l'[CreateDeployment](#) opération.

Important

Les déploiements locaux ne notifient pas les composants avant les mises à jour.

Il s'agit d'une opération d'abonnement dans le cadre de laquelle vous vous abonnez à un flux de messages d'événements. Pour utiliser cette opération, définissez un gestionnaire de réponse au flux

avec des fonctions qui gèrent les messages d'événements, les erreurs et la fermeture du flux. Pour de plus amples informations, veuillez consulter [Abonnez-vous aux diffusions d'événements IPC](#).

Type de message d'événement : `ComponentUpdatePolicyEvents`

 Tip

Vous pouvez suivre un didacticiel pour apprendre à développer un composant qui reporte de manière conditionnelle les mises à jour des composants. Pour de plus amples informations, veuillez consulter [Tutoriel : Développement d'un composant Greengrass qui reporte les mises à jour des composants](#).

Demande

La demande de cette opération ne comporte aucun paramètre.

Réponse

La réponse de cette opération contient les informations suivantes :

messages

Le flux de messages de notification. Cet objet contient `ComponentUpdatePolicyEvents` les informations suivantes :

`preUpdateEvent`(Python :`pre_update_event`)

(Facultatif) Événement indiquant que le noyau souhaite mettre à jour un composant. Vous pouvez répondre par une [DeferComponentUpdate](#) opération visant à accuser réception ou à différer la mise à jour jusqu'à ce que le composant soit prêt à redémarrer. Cet objet contient `PreComponentUpdateEvent` les informations suivantes :

`deploymentId`(Python :`deployment_id`)

ID du AWS IoT Greengrass déploiement qui met à jour le composant.

`isGgcRestarting`(Python :`is_ggc_restarting`)

Si le noyau doit ou non redémarrer pour appliquer la mise à jour.

`postUpdateEvent(Python :post_update_event)`

(Facultatif) Événement indiquant que le noyau a mis à jour un composant. Cet objet contient `PostComponentUpdateEvent` les informations suivantes :

`deploymentId(Python :deployment_id)`

ID du AWS IoT Greengrass déploiement qui a mis à jour le composant.

 Note

Cette fonctionnalité nécessite la version 2.7.0 ou ultérieure du composant Greengrass nucleus.

DeferComponentUpdate

Reconnaissez ou reportez une mise à jour d'un composant que vous découvrez avec [SubscribeToComponentUpdates](#). Vous spécifiez le délai d'attente avant que le noyau vérifie à nouveau si votre composant est prêt à être mis à jour. Vous pouvez également utiliser cette opération pour indiquer au noyau que votre composant est prêt pour la mise à jour.

Si un composant ne répond pas à la notification de mise à jour du composant, le noyau attend le délai que vous spécifiez dans la politique de mise à jour des composants du déploiement. Après ce délai, le noyau procède au déploiement. Le délai de mise à jour des composants par défaut est de 60 secondes. Pour plus d'informations, reportez-vous à la section [Créer des déploiements](#) et à l'[DeploymentComponentUpdatePolicy](#) objet que vous pouvez fournir lorsque vous appelez l'[CreateDeployment](#) opération.

 Tip

Vous pouvez suivre un didacticiel pour apprendre à développer un composant qui reporte de manière conditionnelle les mises à jour des composants. Pour de plus amples informations, veuillez consulter [Tutoriel : Développement d'un composant Greengrass qui reporte les mises à jour des composants](#).

Demande

La demande de cette opération comporte les paramètres suivants :

`deploymentId(Python :deployment_id)`

ID du AWS IoT Greengrass déploiement à reporter.

`message`

(Facultatif) Nom du composant dont les mises à jour doivent être différées.

La valeur par défaut est le nom du composant qui fait la demande.

`recheckAfterMs(Python :recheck_after_ms)`

Durée en millisecondes pendant laquelle la mise à jour doit être différée. Le noyau attend ce laps de temps puis envoie un autre avec `PreComponentUpdateEvent` [SubscribeToComponentUpdates](#) lequel vous pouvez le découvrir.

Spécifiez 0 pour accuser réception de la mise à jour. Cela indique au noyau que votre composant est prêt pour la mise à jour.

La valeur par défaut est zéro milliseconde, ce qui signifie que la mise à jour est confirmée.

Réponse

Cette opération ne fournit aucune information dans sa réponse.

PauseComponent

Cette fonctionnalité est disponible pour les versions 2.4.0 et ultérieures du composant [Greengrass](#) nucleus. AWS IoT Greengrass ne prend actuellement pas en charge cette fonctionnalité sur les appareils Windows principaux.

Suspend les processus d'un composant sur le périphérique principal. Pour reprendre un composant, utilisez l'[ResumeComponent](#) opération.

Vous ne pouvez suspendre que les composants génériques. Si vous essayez de suspendre un autre type de composant, cette opération génère un `InvalidRequestError`.

Note

Cette opération ne peut pas suspendre les processus conteneurisés, tels que les conteneurs Docker. Pour suspendre et reprendre un conteneur Docker, vous pouvez utiliser les commandes [docker pause](#) et [docker unpause](#).

Cette opération ne met pas en pause les dépendances des composants ni les composants qui dépendent du composant que vous interrompez. Tenez compte de ce comportement lorsque vous suspendez un composant dépendant d'un autre composant, car le composant dépendant peut rencontrer des problèmes lorsque sa dépendance est suspendue.

Lorsque vous redémarrez ou arrêtez un composant suspendu, par exemple lors d'un déploiement, le noyau Greengrass reprend le composant et exécute son cycle de vie d'arrêt. Pour plus d'informations sur le redémarrage d'un composant, consultez [RestartComponent](#).

Important

Pour utiliser cette opération, vous devez définir une politique d'autorisation qui accorde l'autorisation d'utiliser cette opération. Pour de plus amples informations, veuillez consulter [Autorisation](#).

Versions minimales du SDK

Le tableau suivant répertorie les versions minimales Kit SDK des appareils AWS IoT que vous devez utiliser pour suspendre et reprendre les composants.

SDK	Version minimale	
Kit SDK des appareils AWS IoT pour Java v2	v1.4.3	
Kit SDK des appareils AWS IoT pour Python v2	v1.6.2	
Kit SDK des appareils AWS IoT pour C++ v2	v1.13.1	
Kit SDK des appareils AWS IoT pour JavaScript v2	v1.12.0	

Demande

La demande de cette opération comporte les paramètres suivants :

`componentName(Python :component_name)`

Nom du composant à suspendre, qui doit être un composant générique. Pour de plus amples informations, veuillez consulter [Types de composants](#).

Réponse

Cette opération ne fournit aucune information dans sa réponse.

ResumeComponent

Cette fonctionnalité est disponible pour les versions 2.4.0 et ultérieures du composant [Greengrass](#) nucleus. AWS IoT Greengrass ne prend actuellement pas en charge cette fonctionnalité sur les appareils Windows principaux.

Reprend les processus d'un composant sur le périphérique principal. Pour suspendre un composant, utilisez l'[PauseComponent](#) opération.

Vous ne pouvez reprendre que les composants en pause. Si vous essayez de reprendre un composant qui n'est pas en pause, cette opération lance un `InvalidRequestError`.

Important

Pour utiliser cette opération, vous devez définir une politique d'autorisation qui accorde l'autorisation de le faire. Pour de plus amples informations, veuillez consulter [Autorisation](#).

Versions minimales du SDK

Le tableau suivant répertorie les versions minimales Kit SDK des appareils AWS IoT que vous devez utiliser pour suspendre et reprendre les composants.

SDK	Version minimale	
Kit SDK des appareils AWS IoT pour Java v2	v1.4.3	
Kit SDK des appareils AWS IoT pour Python v2	v1.6.2	

SDK	Version minimale	
Kit SDK des appareils AWS IoT pour C++ v2	v1.13.1	
Kit SDK des appareils AWS IoT pour JavaScript v2	v1.12.0	

Demande

La demande de cette opération comporte les paramètres suivants :

`componentName`(Python :`component_name`)

Nom du composant à reprendre.

Réponse

Cette opération ne fournit aucune information dans sa réponse.

Interagir avec la configuration des composants

Le service IPC de configuration des composants vous permet d'effectuer les opérations suivantes :

- Obtenez et définissez les paramètres de configuration des composants.
- Abonnez-vous aux mises à jour de configuration des composants.
- Validez les mises à jour de configuration des composants avant que le noyau ne les applique.

Rubriques

- [Versions minimales du SDK](#)
- [GetConfiguration](#)
- [UpdateConfiguration](#)
- [SubscribeToConfigurationUpdate](#)
- [SubscribeToValidateConfigurationUpdates](#)
- [SendConfigurationValidityReport](#)

Versions minimales du SDK

Le tableau suivant répertorie les versions minimales Kit SDK des appareils AWS IoT que vous devez utiliser pour interagir avec la configuration des composants.

SDK	Version minimale	
Kit SDK des appareils AWS IoT pour Java v2	v1.2.10	
Kit SDK des appareils AWS IoT pour Python v2	v1.5.3	
Kit SDK des appareils AWS IoT pour C++ v2	v1.17.0	
Kit SDK des appareils AWS IoT pour JavaScript v2	v1.12.0	

GetConfiguration

Obtient une valeur de configuration pour un composant du périphérique principal. Vous spécifiez le chemin clé pour lequel vous souhaitez obtenir une valeur de configuration.

Demande

La demande de cette opération comporte les paramètres suivants :

`componentName`(Python :`component_name`)

(Facultatif) Le nom du composant.

La valeur par défaut est le nom du composant à l'origine de la demande.

`keyPath`(Python :`key_path`)

Le chemin clé vers la valeur de configuration. Spécifiez une liste dans laquelle chaque entrée est la clé d'un seul niveau de l'objet de configuration. Par exemple, spécifiez `["mqtt", "port"]` d'obtenir la valeur de `port` dans la configuration suivante.

```
{
  "mqtt": {
    "port": 443
  }
}
```

Pour obtenir la configuration complète du composant, spécifiez une liste vide.

Réponse

La réponse de cette opération contient les informations suivantes :

`componentName`(Python :`component_name`)

Le nom du composant.

`value`

La configuration demandée sous forme d'objet.

UpdateConfiguration

Met à jour une valeur de configuration pour ce composant sur le périphérique principal.

Demande

La demande de cette opération comporte les paramètres suivants :

`keyPath`(Python :`key_path`)

(Facultatif) Le chemin clé vers le nœud de conteneur (l'objet) à mettre à jour. Spécifiez une liste dans laquelle chaque entrée est la clé d'un seul niveau de l'objet de configuration. Par exemple, spécifiez le chemin clé `["mqtt"]` et la valeur de fusion `{ "port": 443 }` dont vous souhaitez définir la valeur `port` dans la configuration suivante.

```
{
  "mqtt": {
    "port": 443
  }
}
```

Le chemin clé doit spécifier un nœud de conteneur (un objet) dans la configuration. Si le nœud n'existe pas dans la configuration du composant, cette opération le crée et attribue sa valeur à l'objet dans `valueToMerge`.

La valeur par défaut est la racine de l'objet de configuration.

`timestamp`

Durée actuelle de l'époque Unix en millisecondes. Cette opération utilise cet horodatage pour résoudre les mises à jour simultanées de la clé. Si la clé de la configuration du composant possède un horodatage supérieur à celui de la demande, la demande échoue.

`valueToMerge`(Python :`value_to_merge`)

L'objet de configuration à fusionner à l'emplacement que vous spécifiez `keyPath`. Pour de plus amples informations, veuillez consulter [Mettre à jour les configurations des composants](#).

Réponse

Cette opération ne fournit aucune information dans sa réponse.

SubscribeToConfigurationUpdate

Abonnez-vous pour recevoir des notifications lorsque la configuration d'un composant est mise à jour. Lorsque vous vous abonnez à une clé, vous recevez une notification lorsqu'un enfant porteur de cette clé est mis à jour.

Il s'agit d'une opération d'abonnement dans le cadre de laquelle vous vous abonnez à un flux de messages d'événements. Pour utiliser cette opération, définissez un gestionnaire de réponse au flux avec des fonctions qui gèrent les messages d'événements, les erreurs et la fermeture du flux. Pour de plus amples informations, veuillez consulter [Abonnez-vous aux diffusions d'événements IPC](#).

Type de message d'événement : `ConfigurationUpdateEvents`

Demande

La demande de cette opération comporte les paramètres suivants :

`componentName`(Python :`component_name`)

(Facultatif) Le nom du composant.

La valeur par défaut est le nom du composant à l'origine de la demande.

keyPath(Python :key_path)

Le chemin clé vers la valeur de configuration à laquelle vous souhaitez vous abonner. Spécifiez une liste dans laquelle chaque entrée est la clé d'un seul niveau de l'objet de configuration. Par exemple, spécifiez ["mqtt", "port"] d'obtenir la valeur de port dans la configuration suivante.

```
{
  "mqtt": {
    "port": 443
  }
}
```

Pour vous abonner aux mises à jour pour toutes les valeurs de la configuration du composant, spécifiez une liste vide.

Réponse

La réponse de cette opération contient les informations suivantes :

messages

Le flux de messages de notification. Cet objet contient ConfigurationUpdateEvents les informations suivantes :

configurationUpdateEvent(Python :configuration_update_event)

L'événement de mise à jour de configuration. Cet objet contient ConfigurationUpdateEvent les informations suivantes :

componentName(Python :component_name)

Le nom du composant.

keyPath(Python :key_path)

Le chemin clé vers la valeur de configuration mise à jour.

SubscribeToValidateConfigurationUpdates

Abonnez-vous pour recevoir des notifications avant les mises à jour de configuration de ce composant. Cela permet aux composants de valider les mises à jour de leur propre configuration.

Utilisez cette [SendConfigurationValidityReport](#) opération pour indiquer au noyau si la configuration est valide ou non.

 Important

Les déploiements locaux n'informent pas les composants des mises à jour.

Il s'agit d'une opération d'abonnement dans le cadre de laquelle vous vous abonnez à un flux de messages d'événements. Pour utiliser cette opération, définissez un gestionnaire de réponse au flux avec des fonctions qui gèrent les messages d'événements, les erreurs et la fermeture du flux. Pour de plus amples informations, veuillez consulter [Abonnez-vous aux diffusions d'événements IPC](#).

Type de message d'événement : `ValidateConfigurationUpdateEvents`

Demande

La demande de cette opération ne comporte aucun paramètre.

Réponse

La réponse de cette opération contient les informations suivantes :

messages

Le flux de messages de notification. Cet objet contient

`ValidateConfigurationUpdateEvents` les informations suivantes :

`validateConfigurationUpdateEvent`(Python :`validate_configuration_update_event`)

L'événement de mise à jour de configuration. Cet objet contient

`ValidateConfigurationUpdateEvent` les informations suivantes :

`deploymentId`(Python :`deployment_id`)

ID du AWS IoT Greengrass déploiement qui met à jour le composant.

`configuration`

L'objet qui contient la nouvelle configuration.

SendConfigurationValidityReport

Indiquez au noyau si une mise à jour de configuration de ce composant est valide ou non. Le déploiement échoue si vous indiquez au noyau que la nouvelle configuration n'est pas valide. Utilisez l'[SubscribeToValidateConfigurationUpdates](#) opération pour vous abonner afin de valider les mises à jour de configuration.

Si un composant ne répond pas à une notification de mise à jour de configuration validée, le noyau attend le délai que vous spécifiez dans la politique de validation de configuration du déploiement. Après ce délai, le noyau procède au déploiement. Le délai de validation du composant par défaut est de 20 secondes. Pour plus d'informations, reportez-vous à la section [Créer des déploiements](#) et à l'[DeploymentConfigurationValidationPolicy](#) objet que vous pouvez fournir lorsque vous appelez l'[CreateDeployment](#) opération.

Demande

La demande de cette opération comporte les paramètres suivants :

`configurationValidityReport(Python :configuration_validity_report)`

Le rapport qui indique au noyau si la mise à jour de configuration est valide ou non. Cet objet contient `ConfigurationValidityReport` les informations suivantes :

`status`

Le statut de validité. Cette énumération possède `ConfigurationValidityStatus` les valeurs suivantes :

- `ACCEPTED`— La configuration est valide et le noyau peut l'appliquer à ce composant.
- `REJECTED`— La configuration n'est pas valide et le déploiement échoue.

`deploymentId(Python :deployment_id)`

ID du AWS IoT Greengrass déploiement qui a demandé la mise à jour de configuration.

`message`

(Facultatif) Message indiquant pourquoi la configuration n'est pas valide.

Réponse

Cette opération ne fournit aucune information dans sa réponse.

Récupérez les valeurs secrètes

Utilisez le service IPC du gestionnaire de secrets pour récupérer les valeurs secrètes des secrets du périphérique principal. Vous utilisez le [composant gestionnaire de secrets](#) pour déployer des secrets chiffrés sur les appareils principaux. Vous pouvez ensuite utiliser une opération IPC pour déchiffrer le secret et utiliser sa valeur dans vos composants personnalisés.

Rubriques

- [Versions minimales du SDK](#)
- [Autorisation](#)
- [GetSecretValue](#)
- [Exemples](#)

Versions minimales du SDK

Le tableau suivant répertorie les versions minimales Kit SDK des appareils AWS IoT que vous devez utiliser pour récupérer des valeurs secrètes à partir de secrets sur le périphérique principal.

SDK	Version minimale	
Kit SDK des appareils AWS IoT pour Java v2	v1.2.10	
Kit SDK des appareils AWS IoT pour Python v2	v1.5.3	
Kit SDK des appareils AWS IoT pour C++ v2	v1.17.0	
Kit SDK des appareils AWS IoT pour JavaScript v2	v1.12.0	

Autorisation

Pour utiliser le gestionnaire de secrets dans un composant personnalisé, vous devez définir des politiques d'autorisation qui permettent à votre composant d'obtenir la valeur des secrets que

vous stockez sur le périphérique principal. Pour plus d'informations sur la définition des politiques d'autorisation, consultez [Autoriser les composants à effectuer des opérations IPC](#).

Les politiques d'autorisation pour Secret Manager possèdent les propriétés suivantes.

Identifiant du service IPC : `aws.greengrass.SecretManager`

Opération	Description	Ressources
<code>aws.greengrass#GetSecretValue</code> ou <code>*</code>	Permet à un composant d'obtenir la valeur des secrets chiffrés sur le périphérique principal.	Un ARN secret du Secrets Manager, ou <code>*</code> pour autoriser l'accès à tous les secrets.

Exemples de politiques d'autorisation

Vous pouvez vous référer à l'exemple de politique d'autorisation suivant pour vous aider à configurer les politiques d'autorisation pour vos composants.

Exemple de politique d'autorisation

L'exemple de politique d'autorisation suivant permet à un composant d'obtenir la valeur de n'importe quel secret sur le périphérique principal.

Note

Dans un environnement de production, nous vous recommandons de réduire la portée de la politique d'autorisation afin que le composant ne récupère que les secrets qu'il utilise. Vous pouvez remplacer le `*` caractère générique par une liste de secrets ARNs lorsque vous déployez le composant.

```
{
  "accessControl": {
    "aws.greengrass.SecretManager": {
      "com.example.MySecretComponent:secrets:1": {
        "policyDescription": "Allows access to a secret.",
        "operations": [
          "aws.greengrass#GetSecretValue"
        ]
      }
    }
  }
}
```



```
    ],  
    "resources": [  
        "*"   
    ]  
  }  
}
```

GetSecretValue

Obtient la valeur d'un secret que vous stockez sur le périphérique principal.

Cette opération est similaire à l'opération Secrets Manager que vous pouvez utiliser pour obtenir la valeur d'un secret dans le AWS Cloud. Pour plus d'informations, consultez [GetSecretValue](#) dans la Référence d'API AWS Secrets Manager .

Demande

La demande de cette opération comporte les paramètres suivants :

`refresh`(Python :`refresh`)

(facultatif) : s'il faut synchroniser le secret demandé avec sa dernière valeur du AWS Secrets Manager service.

Lorsqu'il est défini sur `true`, le gestionnaire de secrets demande au AWS Secrets Manager service la dernière valeur de l'étiquette secrète spécifiée et renvoie cette valeur en réponse. Dans le cas contraire, la valeur secrète stockée localement sera renvoyée.

Ce paramètre ne fonctionnera pas conjointement avec le `versionId` paramètre de la demande. Ce paramètre fonctionne lorsqu'il est utilisé conjointement avec Nucleus 2.13.0 et versions ultérieures.

`secretId`(Python :`secret_id`)

Le nom du secret à obtenir. Vous pouvez spécifier l'Amazon Resource Name (ARN) ou le nom convivial du secret.

`versionId`(Python :`version_id`)

(Facultatif) L'ID de la version à obtenir.

Vous pouvez spécifier `versionId` ou `versionStage`.

Si vous ne spécifiez pas `versionId` ou `versionStage`, cette opération utilise par défaut la version avec l'AWSCURRENT étiquette.

`versionStage`(Python :`version_stage`)

(Facultatif) L'étiquette de mise en scène de la version à obtenir.

Vous pouvez spécifier `versionId` ou `versionStage`.

Si vous ne spécifiez pas `versionId` ou `versionStage`, cette opération utilise par défaut la version avec l'AWSCURRENT étiquette.

Réponse

La réponse de cette opération contient les informations suivantes :

`secretId`(Python :`secret_id`)

L'identifiant du secret.

`versionId`(Python :`version_id`)

L'ID de cette version du secret.

`versionStage`(Python :`version_stage`)

La liste des labels de mise en scène attachés à cette version du secret.

`secretValue`(Python :`secret_value`)

La valeur de cette version du secret. Cet objet contient `SecretValue` les informations suivantes.

`secretString`(Python :`secret_string`)

Partie déchiffrée des informations secrètes protégées que vous avez fournies à Secrets Manager sous forme de chaîne.

`secretBinary`(Python :`secret_binary`)

(Facultatif) Partie déchiffrée des informations secrètes protégées que vous avez fournies à Secrets Manager sous forme de données binaires sous la forme d'un tableau d'octets. Cette propriété contient les données binaires sous forme de chaîne codée en base64.

Cette propriété n'est pas utilisée si vous avez créé le secret dans la console Secrets Manager.

Exemples

Les exemples suivants montrent comment appeler cette opération dans le code de composant personnalisé.

Java (IPC client V1)

Exemple Exemple : obtenir une valeur secrète

Note

Cet exemple utilise une `IPCUtils` classe pour créer une connexion au service AWS IoT Greengrass Core IPC. Pour de plus amples informations, veuillez consulter [Connectez-vous au service AWS IoT Greengrass Core IPC](#).

```
package com.aws.greengrass.docs.samples.ipc;

import com.aws.greengrass.docs.samples.ipc.util.IPCUtils;
import software.amazon.awssdk.aws.greengrass.GetSecretValueResponseHandler;
import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClient;
import software.amazon.awssdk.aws.greengrass.model.GetSecretValueRequest;
import software.amazon.awssdk.aws.greengrass.model.GetSecretValueResponse;
import software.amazon.awssdk.aws.greengrass.model.UnauthorizedError;
import software.amazon.awssdk.eventstreamrpc.EventStreamRPCConnection;

import java.util.Optional;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;

public class GetSecretValue {

    public static final int TIMEOUT_SECONDS = 10;

    public static void main(String[] args) {
        String secretArn = args[0];
        String versionStage = args[1];
        try (EventStreamRPCConnection eventStreamRPCConnection =
            IPCUtils.getEventStreamRpcConnection()) {
            GreengrassCoreIPCClient ipcClient =
```

```

        new GreengrassCoreIPCClient(eventStreamRPCConnection);
        GetSecretValueResponseHandler responseHandler =
            GetSecretValue.getSecretValue(ipcClient, secretArn,
versionStage);
        CompletableFuture<GetSecretValueResponse> futureResponse =
            responseHandler.getResponse();
        try {
            GetSecretValueResponse response =
futureResponse.get(TIMEOUT_SECONDS, TimeUnit.SECONDS);
            response.getSecretValue().postFromJson();
            String secretString = response.getSecretValue().getSecretString();
            System.out.println("Successfully retrieved secret value: " +
secretString);
        } catch (TimeoutException e) {
            System.err.println("Timeout occurred while retrieving secret: " +
secretArn);
        } catch (ExecutionException e) {
            if (e.getCause() instanceof UnauthorizedError) {
                System.err.println("Unauthorized error while retrieving secret:
" + secretArn);
            } else {
                throw e;
            }
        } catch (InterruptedException e) {
            System.out.println("IPC interrupted.");
        } catch (ExecutionException e) {
            System.err.println("Exception occurred when using IPC.");
            e.printStackTrace();
            System.exit(1);
        }
    }

    public static GetSecretValueResponseHandler
getSecretValue(GreengrassCoreIPCClient greengrassCoreIPCClient, String secretArn,
String versionStage) {
        GetSecretValueRequest getSecretValueRequest = new GetSecretValueRequest();
        getSecretValueRequest.setSecretId(secretArn);
        getSecretValueRequest.setVersionStage(versionStage);
        return greengrassCoreIPCClient.getSecretValue(getSecretValueRequest,
Optional.empty());
    }
}

```

Python (IPC client V1)

Exemple Exemple : obtenir une valeur secrète

Note

Cet exemple suppose que vous utilisez la version 1.5.4 ou ultérieure de Kit SDK des appareils AWS IoT for Python v2.

```
import json

import awsiot.greengrasscoreipc
from awsiot.greengrasscoreipc.model import (
    GetSecretValueRequest,
    GetSecretValueResponse,
    UnauthorizedError
)

secret_id = 'arn:aws:secretsmanager:us-
west-2:123456789012:secret:MyGreengrassSecret-abcdef'
TIMEOUT = 10

ipc_client = awsiot.greengrasscoreipc.connect()

request = GetSecretValueRequest()
request.secret_id = secret_id
request.version_stage = 'AWSCURRENT'
operation = ipc_client.new_get_secret_value()
operation.activate(request)
future_response = operation.get_response()
response = future_response.result(TIMEOUT)
secret_json = json.loads(response.secret_value.secret_string)
# Handle secret value.
```

JavaScript

Exemple Exemple : obtenir une valeur secrète

```
import {
    GetSecretValueRequest,
} from 'aws-iot-device-sdk-v2/dist/greengrasscoreipc/model';
```

```
import * as greengrasscoreipc from "aws-iot-device-sdk-v2/dist/greengrasscoreipc";

class GetSecretValue {
  private readonly secretId : string;
  private readonly versionStage : string;
  private ipcClient : greengrasscoreipc.Client

  constructor() {
    this.secretId = "<define_your_own_secretId>"
    this.versionStage = "<define_your_own_versionStage>"

    this.getSecretValue().then(r => console.log("Started workflow"));
  }

  private async getSecretValue() {
    try {
      this.ipcClient = await getIpcClient();

      const getSecretValueRequest : GetSecretValueRequest = {
        secretId: this.secretId,
        versionStage: this.versionStage,
      };

      const result = await
this.ipcClient.getSecretValue(getSecretValueRequest);
      const secretString = result.secretValue.secretString;
      console.log("Successfully retrieved secret value: " + secretString)
    } catch (e) {
      // parse the error depending on your use cases
      throw e
    }
  }
}

export async function getIpcClient(){
  try {
    const ipcClient = greengrasscoreipc.createClient();
    await ipcClient.connect()
      .catch(error => {
        // parse the error depending on your use cases
        throw error;
      });
    return ipcClient
  } catch (err) {
```

```
        // parse the error depending on your use cases
        throw err
    }
}

const getSecretValue = new GetSecretValue();
```

Exemples

Utilisez les exemples suivants pour apprendre à utiliser le service IPC du gestionnaire de secrets dans vos composants.

Exemple : secret d'impression (Python, client IPC V1)

Cet exemple de composant imprime la valeur d'un secret que vous déployez sur le périphérique principal.

Important

Cet exemple de composant imprime la valeur d'un secret. Ne l'utilisez donc qu'avec les secrets qui stockent des données de test. N'utilisez pas ce composant pour imprimer la valeur d'un secret contenant des informations importantes.

Rubriques

- [Recipe](#)
- [Artefacts](#)
- [Utilisation](#)

Recipe

L'exemple de recette suivant définit un paramètre de configuration ARN secret et permet au composant d'obtenir la valeur de n'importe quel secret sur le périphérique principal.

Note

Dans un environnement de production, nous vous recommandons de réduire la portée de la politique d'autorisation afin que le composant ne récupère que les secrets qu'il utilise.

Vous pouvez remplacer le * caractère générique par une liste de secrets ARNs lorsque vous déployez le composant.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.PrintSecret",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "Prints the value of an AWS Secrets Manager secret.",
  "ComponentPublisher": "Amazon",
  "ComponentDependencies": {
    "aws.greengrass.SecretManager": {
      "VersionRequirement": "^2.0.0",
      "DependencyType": "HARD"
    }
  },
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "SecretArn": "",
      "accessControl": {
        "aws.greengrass.SecretManager": {
          "com.example.PrintSecret:secrets:1": {
            "policyDescription": "Allows access to a secret.",
            "operations": [
              "aws.greengrass#GetSecretValue"
            ],
            "resources": [
              "*"
            ]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "install": "python3 -m pip install --user awsioticsdk",
```



```

      "Run": "python3 -u {artifacts:path}/print_secret.py \"{configuration:/
SecretArn}\""
    }
  },
  {
    "Platform": {
      "os": "windows"
    },
    "Lifecycle": {
      "install": "py -3 -m pip install --user awsiotsdk",
      "Run": "py -3 -u {artifacts:path}/print_secret.py \"{configuration:/
SecretArn}\""
    }
  }
]
}

```

YAML

```

---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.PrintSecret
ComponentVersion: 1.0.0
ComponentDescription: Prints the value of a Secrets Manager secret.
ComponentPublisher: Amazon
ComponentDependencies:
  aws.greengrass.SecretManager:
    VersionRequirement: "^2.0.0"
    DependencyType: HARD
ComponentConfiguration:
  DefaultConfiguration:
    SecretArn: ''
    accessControl:
      aws.greengrass.SecretManager:
        com.example.PrintSecret:secrets:1:
          policyDescription: Allows access to a secret.
          operations:
            - aws.greengrass#GetSecretValue
          resources:
            - "*"
Manifests:
  - Platform:
      os: linux

```

```

Lifecycle:
  install: python3 -m pip install --user awsiotsdk
  Run: python3 -u {artifacts:path}/print_secret.py "{configuration:/SecretArn}"
- Platform:
  os: windows
Lifecycle:
  install: py -3 -m pip install --user awsiotsdk
  Run: py -3 -u {artifacts:path}/print_secret.py "{configuration:/SecretArn}"

```

Artefacts

L'exemple d'application Python suivant montre comment utiliser le service IPC du gestionnaire de secrets pour obtenir la valeur d'un secret sur le périphérique principal.

```

import concurrent.futures
import json
import sys
import traceback

import awsiot.greengrasscoreipc
from awsiot.greengrasscoreipc.model import (
    GetSecretValueRequest,
    GetSecretValueResponse,
    UnauthorizedError
)

TIMEOUT = 10

if len(sys.argv) == 1:
    print('Provide SecretArn in the component configuration.', file=sys.stdout)
    exit(1)

secret_id = sys.argv[1]

try:
    ipc_client = awsiot.greengrasscoreipc.connect()

    request = GetSecretValueRequest()
    request.secret_id = secret_id
    operation = ipc_client.new_get_secret_value()
    operation.activate(request)
    future_response = operation.get_response()

```

```
try:
    response = future_response.result(TIMEOUT)
    secret_json = json.loads(response.secret_value.secret_string)
    print('Successfully got secret: ' + secret_id)
    print('Secret value: ' + str(secret_json))
except concurrent.futures.TimeoutError:
    print('Timeout occurred while getting secret: ' + secret_id, file=sys.stderr)
except UnauthorizedError as e:
    print('Unauthorized error while getting secret: ' + secret_id,
file=sys.stderr)
    raise e
except Exception as e:
    print('Exception while getting secret: ' + secret_id, file=sys.stderr)
    raise e
except Exception:
    print('Exception occurred when using IPC.', file=sys.stderr)
    traceback.print_exc()
    exit(1)
```

Utilisation

Vous pouvez utiliser cet exemple de composant avec le [composant gestionnaire de secrets](#) pour déployer et imprimer la valeur d'un secret sur votre périphérique principal.

Pour créer, déployer et imprimer un secret de test

1. Créez un secret Secrets Manager avec des données de test.

Linux or Unix

```
aws secretsmanager create-secret \
  --name MyTestGreengrassSecret \
  --secret-string '{"my-secret-key": "my-secret-value"}'
```

Windows Command Prompt (CMD)

```
aws secretsmanager create-secret ^
  --name MyTestGreengrassSecret ^
  --secret-string '{"my-secret-key": "my-secret-value"}'
```

PowerShell

```
aws secretsmanager create-secret `
  --name MyTestGreengrassSecret `
  --secret-string '{"my-secret-key": "my-secret-value"}'
```

Enregistrez l'ARN du secret à utiliser dans les étapes suivantes.

Pour plus d'informations, consultez la section [Création d'un secret](#) dans le guide de AWS Secrets Manager l'utilisateur.

2. Déployez le [composant secret manager](#) (`aws.greengrass.SecretManager`) avec la mise à jour de fusion de configuration suivante. Spécifiez l'ARN du secret que vous avez créé précédemment.

```
{
  "cloudSecrets": [
    {
      "arn": "arn:aws:secretsmanager:us-
west-2:123456789012:secret:MyTestGreengrassSecret-abcdef"
    }
  ]
}
```

Pour plus d'informations, consultez [Déployer AWS IoT Greengrass des composants sur des appareils](#) la commande de [déploiement de la CLI Greengrass](#).

3. Créez et déployez le composant d'exemple présenté dans cette section avec la mise à jour de fusion de configuration suivante. Spécifiez l'ARN du secret que vous avez créé précédemment.

```
{
  "SecretArn": "arn:aws:secretsmanager:us-
west-2:123456789012:secret:MyTestGreengrassSecret",
  "accessControl": {
    "aws.greengrass.SecretManager": {
      "com.example.PrintSecret:secrets:1": {
        "policyDescription": "Allows access to a secret.",
        "operations": [
          "aws.greengrass#GetSecretValue"
        ],
        "resources": [
```

```
        "arn:aws:secretsmanager:us-  
west-2:123456789012:secret:MyTestGreengrassSecret-abcdef"  
    ]  
}  
}  
}  
}
```

Pour plus d'informations, consultez [Création de AWS IoT Greengrass composants](#).

4. Consultez les journaux du logiciel AWS IoT Greengrass principal pour vérifier que les déploiements sont réussis, et consultez le journal des `com.example.PrintSecret` composants pour voir la valeur secrète imprimée. Pour de plus amples informations, veuillez consulter [AWS IoT Greengrass Journaux de surveillance](#).

Interagissez avec les ombres locales

Utilisez le service Shadow IPC pour interagir avec les ombres locales sur un appareil. L'appareil avec lequel vous choisissez d'interagir peut être votre appareil principal ou un appareil client connecté.

Pour utiliser ces opérations IPC, incluez le [composant Shadow Manager](#) en tant que dépendance dans votre composant personnalisé. Vous pouvez ensuite utiliser les opérations IPC dans vos composants personnalisés pour interagir avec les ombres locales de votre appareil via le gestionnaire d'ombres. Pour permettre aux composants personnalisés de réagir aux modifications des états fantômes locaux, vous pouvez également utiliser le publish/subscribe IPC service to subscribe to shadow events. For more information about using the publish/subscribe service, voir le [Publier/souscrire des messages locaux](#).

Note

Pour permettre à un périphérique principal d'interagir avec les ombres du périphérique client, vous devez également configurer et déployer le composant de pont MQTT. Pour plus d'informations, voir [Activer le Shadow Manager pour communiquer avec les appareils clients](#).

Rubriques

- [Versions minimales du SDK](#)
- [Autorisation](#)

- [GetThingShadow](#)
- [UpdateThingShadow](#)
- [DeleteThingShadow](#)
- [ListNamedShadowsForThing](#)

Versions minimales du SDK

Le tableau suivant répertorie les versions minimales Kit SDK des appareils AWS IoT que vous devez utiliser pour interagir avec les ombres locales.

SDK	Version minimale	
Kit SDK des appareils AWS IoT pour Java v2	v1.4.0	
Kit SDK des appareils AWS IoT pour Python v2	v1.6.0	
Kit SDK des appareils AWS IoT pour C++ v2	v1.17.0	
Kit SDK des appareils AWS IoT pour JavaScript v2	v1.12.0	

Autorisation

Pour utiliser le service Shadow IPC dans un composant personnalisé, vous devez définir des politiques d'autorisation permettant à votre composant d'interagir avec les ombres. Pour plus d'informations sur la définition des politiques d'autorisation, consultez [Autoriser les composants à effectuer des opérations IPC](#).

Les politiques d'autorisation pour l'interaction parallèle ont les propriétés suivantes.

Identifiant du service IPC : `aws.greengrass.ShadowManager`

Opération	Description	Ressources
<code>aws.greengrass#GetThingShadow</code>	Permet à un composant de récupérer l'ombre d'un objet.	L'une des chaînes suivantes : <code>\$aws/things/<i>thingName</i>/shadow/</code>, pour permettre l'accès à l'ombre classique de l'appareil. <code>\$aws/things/<i>thingName</i>/shadow/<i>shadowName</i></code>, pour autoriser l'accès à une ombre nommée. <code>*</code> pour permettre l'accès à toutes les ombres.
<code>aws.greengrass#UpdateThingShadow</code>	Permet à un composant de mettre à jour l'ombre d'un objet.	L'une des chaînes suivantes : <code>\$aws/things/<i>thingName</i>/shadow/</code>, pour permettre l'accès à l'ombre classique de l'appareil. <code>\$aws/things/<i>thingName</i>/shadow/<i>shadowName</i></code>, pour autoriser l'accès à une ombre nommée. <code>*</code> pour permettre l'accès à toutes les ombres.
<code>aws.greengrass#DeleteThingShadow</code>	Permet à un composant de supprimer l'ombre d'un objet.	L'une des chaînes suivantes :

Opération	Description	Ressources
		• <code>\$aws/thin</code> <code>gs/ <i>thingName</i> /</code> <code>shadow/</code>, pour permettre l'accès au shadow classique de l'appareil • <code>\$aws/thin</code> <code>gs/ <i>thingName</i> /shadow/n</code> <code>ame/ <i>shadowName</i> </code>, pour autoriser l'accès à une ombre nommée • <code>*</code>, pour permettre l'accès à toutes les ombres.
<code>aws.greengrass#ListNamedShadowsForThing</code>	Permet à un composant de récupérer la liste des ombres nommées pour un objet.	Chaîne de nom d'objet qui permet d'accéder à l'objet pour répertorier ses ombres. *À utiliser pour autoriser l'accès à tout.

Identifiant du service IPC : `aws.greengrass.ipc.pubsub`

Opération	Description	Ressources
<code>aws.greengrass#SubscribeToTopic</code>	Permet à un composant de s'abonner à des messages pour les sujets que vous spécifiez.	L'une des chaînes de rubrique suivantes : • <code><i>shadowTopicPrefix</i> /</code> <code>get/accepted</code> • <code><i>shadowTopicPrefix</i> /</code> <code>get/rejected</code> • <code><i>shadowTopicPrefix</i> /</code> <code>delete/accepted</code>

Opération	Description	Ressources
		• <i>shadowTopicPrefix</i> / delete/rejected • <i>shadowTopicPrefix</i> / update/accepted • <i>shadowTopicPrefix</i> / update/delta • <i>shadowTopicPrefix</i> / update/rejected La valeur du préfixe de rubrique <i>shadowTopicPrefix</i> dépend du type d'ombre : • Ombre classique : \$aws/things/ <i>thingName</i> / shadow • Ombre nommée : \$aws/things/ <i>thingName</i> /shadow/n ame/ <i>shadowName</i> *À utiliser pour autoriser l'accès à toutes les rubriques. Dans Greengrass noyau v2.6.0 et versions ultérieures, vous pouvez vous abonner à des sujets contenant des caractères génériques MQTT (et). # + Cette chaîne de rubrique prend en charge les caractères génériques des rubriques MQTT sous

Opération	Description	Ressources
		forme de caractères littéraux . Par exemple, si la politique d'autorisation d'un composant accorde l'accès à <code>test/topic/# test/topic/#</code>, le composant peut s'abonner, mais <code>pastest/topic/filter</code>.

Variables de recette dans les politiques d'autorisation fictives locales

Si vous utilisez la version 2.6.0 ou ultérieure du noyau Greengrass et que vous définissez l'option de `interpolateComponentConfiguration` du noyau Greengrass sur `true`, vous pouvez utiliser la variable de recette dans les politiques d'autorisation. `{iot:thingName}` Cette fonctionnalité vous permet de configurer une politique d'autorisation unique pour un groupe de périphériques principaux, chaque périphérique principal ne pouvant accéder qu'à son propre périphérique fantôme. Par exemple, vous pouvez autoriser un composant à accéder à la ressource suivante pour les opérations IPC parallèles.

```
$aws/things/{iot:thingName}/shadow/
```

Exemples de politiques d'autorisation

Vous pouvez vous référer aux exemples de politiques d'autorisation suivants pour vous aider à configurer les politiques d'autorisation pour vos composants.

Exemple Exemple : autoriser un groupe d'appareils principaux à interagir avec les ombres locales

Important

Cet exemple utilise une fonctionnalité disponible pour les versions 2.6.0 et ultérieures du composant [Greengrass](#) nucleus. Greengrass nucleus v2.6.0 ajoute la prise en charge de la plupart des [variables de recette](#), notamment dans les configurations de composants `{iot:thingName}`. Pour activer cette fonctionnalité, définissez l'option de configuration du noyau de Greengrass sur `interpolateComponentConfiguration` `true` Pour un

exemple qui fonctionne pour toutes les versions du noyau Greengrass, consultez l'[exemple de politique d'autorisation pour un appareil monocœur](#).

L'exemple de politique d'autorisation suivant permet `com.example.MyShadowInteractionComponent` au composant d'interagir avec l'ombre de périphérique classique et l'ombre nommée `myNamedShadow` pour le périphérique principal qui exécute le composant. Cette politique permet également à ce composant de recevoir des messages sur des sujets locaux pour ces ombres.

JSON

```
{
  "accessControl": {
    "aws.greengrass.ShadowManager": {
      "com.example.MyShadowInteractionComponent:shadow:1": {
        "policyDescription": "Allows access to shadows",
        "operations": [
          "aws.greengrass#GetThingShadow",
          "aws.greengrass#UpdateThingShadow",
          "aws.greengrass#DeleteThingShadow"
        ],
        "resources": [
          "$aws/things/{iot:thingName}/shadow",
          "$aws/things/{iot:thingName}/shadow/name/myNamedShadow"
        ]
      },
      "com.example.MyShadowInteractionComponent:shadow:2": {
        "policyDescription": "Allows access to things with shadows",
        "operations": [
          "aws.greengrass#ListNamedShadowsForThing"
        ],
        "resources": [
          "{iot:thingName}"
        ]
      }
    },
    "aws.greengrass.ipc.pubsub": {
      "com.example.MyShadowInteractionComponent:pubsub:1": {
        "policyDescription": "Allows access to shadow pubsub topics",
        "operations": [
          "aws.greengrass#SubscribeToTopic"
        ]
      }
    }
  }
}
```

```

    ],
    "resources": [
        "$aws/things/{iot:thingName}/shadow/get/accepted",
        "$aws/things/{iot:thingName}/shadow/name/myNamedShadow/get/accepted"
    ]
  }
}
}
}

```

YAML

```

accessControl:
  aws.greengrass.ShadowManager:
    'com.example.MyShadowInteractionComponent:shadow:1':
      policyDescription: 'Allows access to shadows'
      operations:
        - 'aws.greengrass#GetThingShadow'
        - 'aws.greengrass#UpdateThingShadow'
        - 'aws.greengrass#DeleteThingShadow'
      resources:
        - $aws/things/{iot:thingName}/shadow
        - $aws/things/{iot:thingName}/shadow/name/myNamedShadow
    'com.example.MyShadowInteractionComponent:shadow:2':
      policyDescription: 'Allows access to things with shadows'
      operations:
        - 'aws.greengrass#ListNamedShadowsForThing'
      resources:
        - '{iot:thingName}'
  aws.greengrass.ipc.pubsub:
    'com.example.MyShadowInteractionComponent:pubsub:1':
      policyDescription: 'Allows access to shadow pubsub topics'
      operations:
        - 'aws.greengrass#SubscribeToTopic'
      resources:
        - $aws/things/{iot:thingName}/shadow/get/accepted
        - $aws/things/{iot:thingName}/shadow/name/myNamedShadow/get/accepted

```

Exemple Exemple : autoriser un groupe d'appareils principaux à interagir avec les ombres des appareils clients

 Important

Cette fonctionnalité nécessite Greengrass nucleus v2.6.0 ou version ultérieure, Shadow Manager v2.2.0 ou version ultérieure, et MQTT bridge v2.2.0 ou version ultérieure. Vous devez configurer le pont MQTT pour [permettre au Shadow Manager de communiquer avec les appareils clients](#).

L'exemple de politique d'autorisation suivant permet au composant `com.example.MyShadowInteractionComponent` d'interagir avec toutes les ombres de périphériques pour les appareils clients dont le nom commence par `MyClientDevice`.

 Note

Pour permettre à un périphérique principal d'interagir avec les ombres du périphérique client, vous devez également configurer et déployer le composant de pont MQTT. Pour plus d'informations, voir [Activer le Shadow Manager pour communiquer avec les appareils clients](#).

JSON

```
{
  "accessControl": {
    "aws.greengrass.ShadowManager": {
      "com.example.MyShadowInteractionComponent:shadow:1": {
        "policyDescription": "Allows access to shadows",
        "operations": [
          "aws.greengrass#GetThingShadow",
          "aws.greengrass#UpdateThingShadow",
          "aws.greengrass#DeleteThingShadow"
        ],
        "resources": [
          "$aws/things/MyClientDevice*/shadow",
          "$aws/things/MyClientDevice*/shadow/name/*"
        ]
      },
      "com.example.MyShadowInteractionComponent:shadow:2": {
```

```

    "policyDescription": "Allows access to things with shadows",
    "operations": [
      "aws.greengrass#ListNamedShadowsForThing"
    ],
    "resources": [
      "MyClientDevice*"
    ]
  }
}
}
}

```

YAML

```

accessControl:
  aws.greengrass.ShadowManager:
    'com.example.MyShadowInteractionComponent:shadow:1':
      policyDescription: 'Allows access to shadows'
      operations:
        - 'aws.greengrass#GetThingShadow'
        - 'aws.greengrass#UpdateThingShadow'
        - 'aws.greengrass#DeleteThingShadow'
      resources:
        - $aws/things/MyClientDevice*/shadow
        - $aws/things/MyClientDevice*/shadow/name/*
    'com.example.MyShadowInteractionComponent:shadow:2':
      policyDescription: 'Allows access to things with shadows'
      operations:
        - 'aws.greengrass#ListNamedShadowsForThing'
      resources:
        - MyClientDevice*

```

Exemple Exemple : autoriser un appareil central à interagir avec les ombres locales

L'exemple de politique d'autorisation suivant permet

`com.example.MyShadowInteractionComponent` au composant d'interagir avec l'ombre classique du périphérique et l'ombre nommée `myNamedShadow` pour le périphérique `MyThingName`. Cette politique permet également à ce composant de recevoir des messages sur des sujets locaux pour ces ombres.

JSON

```
{
  "accessControl": {
    "aws.greengrass.ShadowManager": {
      "com.example.MyShadowInteractionComponent:shadow:1": {
        "policyDescription": "Allows access to shadows",
        "operations": [
          "aws.greengrass#GetThingShadow",
          "aws.greengrass#UpdateThingShadow",
          "aws.greengrass#DeleteThingShadow"
        ],
        "resources": [
          "$aws/things/MyThingName/shadow",
          "$aws/things/MyThingName/shadow/name/myNamedShadow"
        ]
      },
      "com.example.MyShadowInteractionComponent:shadow:2": {
        "policyDescription": "Allows access to things with shadows",
        "operations": [
          "aws.greengrass#ListNamedShadowsForThing"
        ],
        "resources": [
          "MyThingName"
        ]
      }
    },
    "aws.greengrass.ipc.pubsub": {
      "com.example.MyShadowInteractionComponent:pubsub:1": {
        "policyDescription": "Allows access to shadow pubsub topics",
        "operations": [
          "aws.greengrass#SubscribeToTopic"
        ],
        "resources": [
          "$aws/things/MyThingName/shadow/get/accepted",
          "$aws/things/MyThingName/shadow/name/myNamedShadow/get/accepted"
        ]
      }
    }
  }
}
```

YAML

```

accessControl:
  aws.greengrass.ShadowManager:
    'com.example.MyShadowInteractionComponent:shadow:1':
      policyDescription: 'Allows access to shadows'
      operations:
        - 'aws.greengrass#GetThingShadow'
        - 'aws.greengrass#UpdateThingShadow'
        - 'aws.greengrass#DeleteThingShadow'
      resources:
        - $aws/things/MyThingName/shadow
        - $aws/things/MyThingName/shadow/name/myNamedShadow
    'com.example.MyShadowInteractionComponent:shadow:2':
      policyDescription: 'Allows access to things with shadows'
      operations:
        - 'aws.greengrass#ListNamedShadowsForThing'
      resources:
        - MyThingName
  aws.greengrass.ipc.pubsub:
    'com.example.MyShadowInteractionComponent:pubsub:1':
      policyDescription: 'Allows access to shadow pubsub topics'
      operations:
        - 'aws.greengrass#SubscribeToTopic'
      resources:
        - $aws/things/MyThingName/shadow/get/accepted
        - $aws/things/MyThingName/shadow/name/myNamedShadow/get/accepted

```

Exemple Exemple : autoriser un groupe de périphériques principaux à réagir aux modifications de l'état parallèle local

Important

Cet exemple utilise une fonctionnalité disponible pour les versions 2.6.0 et ultérieures du composant [Greengrass](#) nucleus. Greengrass nucleus v2.6.0 ajoute la prise en charge de la plupart des [variables de recette](#), notamment dans les configurations de composants `{iot:thingName}`. Pour activer cette fonctionnalité, définissez l'option de configuration du noyau de Greengrass sur [interpolateComponentConfiguration](#) `true` Pour un

exemple qui fonctionne pour toutes les versions du noyau Greengrass, consultez l'[exemple de politique d'autorisation pour un appareil monocœur](#).

L'exemple de politique de contrôle d'accès suivant permet `com.example.MyShadowReactiveComponent` au client de recevoir des messages sur le `/update/delta` sujet de l'ombre de périphérique classique et de l'ombre nommée `myNamedShadow` sur chaque périphérique principal qui exécute le composant.

JSON

```
{
  "accessControl": {
    "aws.greengrass.ipc.pubsub": {
      "com.example.MyShadowReactiveComponent:pubsub:1": {
        "policyDescription": "Allows access to shadow pubsub topics",
        "operations": [
          "aws.greengrass#SubscribeToTopic"
        ],
        "resources": [
          "$aws/things/{iot:thingName}/shadow/update/delta",
          "$aws/things/{iot:thingName}/shadow/name/myNamedShadow/update/delta"
        ]
      }
    }
  }
}
```

YAML

```
accessControl:
  aws.greengrass.ipc.pubsub:
    "com.example.MyShadowReactiveComponent:pubsub:1":
      policyDescription: Allows access to shadow pubsub topics
      operations:
        - 'aws.greengrass#SubscribeToTopic'
      resources:
        - $aws/things/{iot:thingName}/shadow/update/delta
        - $aws/things/{iot:thingName}/shadow/name/myNamedShadow/update/delta
```

Exemple Exemple : autoriser un périphérique à cœur unique à réagir aux modifications de l'état fantôme local

L'exemple de politique de contrôle d'accès suivant permet `com.example.MyShadowReactiveComponent` au client de recevoir des messages sur le `/update/delta` sujet correspondant à l'ombre classique de l'appareil et à l'ombre nommée `myNamedShadow` pour l'appareil `MyThingName`.

JSON

```
{
  "accessControl": {
    "aws.greengrass.ipc.pubsub": {
      "com.example.MyShadowReactiveComponent:pubsub:1": {
        "policyDescription": "Allows access to shadow pubsub topics",
        "operations": [
          "aws.greengrass#SubscribeToTopic"
        ],
        "resources": [
          "$aws/things/MyThingName/shadow/update/delta",
          "$aws/things/MyThingName/shadow/name/myNamedShadow/update/delta"
        ]
      }
    }
  }
}
```

YAML

```
accessControl:
  aws.greengrass.ipc.pubsub:
    "com.example.MyShadowReactiveComponent:pubsub:1":
      policyDescription: Allows access to shadow pubsub topics
      operations:
        - 'aws.greengrass#SubscribeToTopic'
      resources:
        - $aws/things/MyThingName/shadow/update/delta
        - $aws/things/MyThingName/shadow/name/myNamedShadow/update/delta
```

GetThingShadow

Obtenez l'ombre d'un objet spécifique.

Demande

La demande de cette opération comporte les paramètres suivants :

`thingName(Python :thing_name)`

Nom de l'objet.

Type : `string`

`shadowName(Python :shadow_name)`

Nom du shadow. Pour spécifier l'ombre classique de l'objet, définissez ce paramètre sur une chaîne vide (`""`).

Warning

Le AWS IoT Greengrass service utilise le `AWSThingsShadowV2Deployment` nom shadow pour gérer les déploiements qui ciblent des appareils principaux individuels. Cette ombre nommée est réservée à l'usage du AWS IoT Greengrass service. Ne mettez pas à jour ou ne supprimez pas cette ombre nommée.

Type : `string`

Réponse

La réponse de cette opération contient les informations suivantes :

`payload`

Le document d'état de réponse sous forme de blob.

Type : `object` qui contient les informations suivantes :

`state`

Les informations sur l'état.

Cet objet contient les informations suivantes.

`desired`

Les propriétés d'état et les valeurs dont la mise à jour est demandée dans l'appareil.

Type : map de paires clé-valeur

`reported`

Les propriétés d'état et les valeurs signalées par l'appareil.

Type : map de paires clé-valeur

`delta`

Différence entre les propriétés et valeurs d'état souhaitées et signalées. Cette propriété n'est présente que si les `reported` états `desired` et sont différents.

Type : map de paires clé-valeur

`metadata`

Les horodatages de chaque attribut dans les `reported` sections `desired` et afin que vous puissiez déterminer quand l'état a été mis à jour.

Type : `string`

`timestamp`

L'époque, la date et l'heure auxquelles la réponse a été générée.

Type : `integer`

`clientToken`(Python :`clientToken`)

Le jeton utilisé pour faire correspondre la demande et la réponse correspondante

Type : `string`

`version`

Version du document fantôme local.

Type : `integer`

Erreurs

Cette opération peut renvoyer les erreurs suivantes.

InvalidArgumentsError

Le service parallèle local n'est pas en mesure de valider les paramètres de la demande. Cela peut se produire si la demande contient du JSON mal formé ou des caractères non pris en charge.

ResourceNotFoundError

Le document parallèle local demandé est introuvable.

ServiceError

Une erreur de service interne s'est produite ou le nombre de demandes adressées au service IPC a dépassé les limites spécifiées dans les paramètres de `maxTotalLocalRequestsRate` configuration `maxLocalRequestsPerSecondPerThing` et dans le composant Shadow Manager.

UnauthorizedError

La politique d'autorisation du composant n'inclut pas les autorisations requises pour cette opération.

Exemples

Les exemples suivants montrent comment appeler cette opération dans le code de composant personnalisé.

Java (IPC client V1)

Exemple Exemple : Get a thing Shadow

Note

Cet exemple utilise une `IPCUtils` classe pour créer une connexion au service AWS IoT Greengrass Core IPC. Pour de plus amples informations, veuillez consulter [Connectez-vous au service AWS IoT Greengrass Core IPC](#).

```
package com.aws.greengrass.docs.samples.ipc;
```

```
import com.aws.greengrass.docs.samples.ipc.util.IPCUtils;
import software.amazon.awssdk.aws.greengrass.GetThingShadowResponseHandler;
import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClient;
import software.amazon.awssdk.aws.greengrass.model.GetThingShadowRequest;
import software.amazon.awssdk.aws.greengrass.model.GetThingShadowResponse;
import software.amazon.awssdk.aws.greengrass.model.ResourceNotFoundError;
import software.amazon.awssdk.aws.greengrass.model.UnauthorizedError;
import software.amazon.awssdk.eventstreamrpc.EventStreamRPCConnection;

import java.nio.charset.StandardCharsets;
import java.util.Optional;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;

public class GetThingShadow {

    public static final int TIMEOUT_SECONDS = 10;

    public static void main(String[] args) {
        // Use the current core device's name if thing name isn't set.
        String thingName = args[0].isEmpty() ? System.getenv("AWS_IOT_THING_NAME") :
args[0];
        String shadowName = args[1];
        try (EventStreamRPCConnection eventStreamRPCConnection =
            IPCUtils.getEventStreamRpcConnection()) {
            GreengrassCoreIPCClient ipcClient =
                new GreengrassCoreIPCClient(eventStreamRPCConnection);
            GetThingShadowResponseHandler responseHandler =
                GetThingShadow.getThingShadow(ipcClient, thingName,
shadowName);
            CompletableFuture<GetThingShadowResponse> futureResponse =
                responseHandler.getResponse();
            try {
                GetThingShadowResponse response =
futureResponse.get(TIMEOUT_SECONDS,
                    TimeUnit.SECONDS);
                String shadowPayload = new String(response.getPayload(),
StandardCharsets.UTF_8);
                System.out.printf("Successfully got shadow %s/%s: %s%n", thingName,
shadowName,
                    shadowPayload);
            } catch (TimeoutException e) {
```

```

        System.err.printf("Timeout occurred while getting shadow: %s/%s%n",
thingName,
                        shadowName);
    } catch (ExecutionException e) {
        if (e.getCause() instanceof UnauthorizedError) {
            System.err.printf("Unauthorized error while getting shadow: %s/
%s%n",
                                thingName, shadowName);
        } else if (e.getCause() instanceof ResourceNotFoundError) {
            System.err.printf("Unable to find shadow to get: %s/%s%n",
thingName,
                                shadowName);
        } else {
            throw e;
        }
    }
} catch (InterruptedException e) {
    System.out.println("IPC interrupted.");
} catch (ExecutionException e) {
    System.err.println("Exception occurred when using IPC.");
    e.printStackTrace();
    System.exit(1);
}
}

    public static GetThingShadowResponseHandler
getThingShadow(GreengrassCoreIPCClient greengrassCoreIPCClient, String thingName,
String shadowName) {
    GetThingShadowRequest getThingShadowRequest = new GetThingShadowRequest();
    getThingShadowRequest.setThingName(thingName);
    getThingShadowRequest.setShadowName(shadowName);
    return greengrassCoreIPCClient.getThingShadow(getThingShadowRequest,
Optional.empty());
}
}

```

Python (IPC client V1)

Example Exemple : Get a thing Shadow

```

import awsiot.greengrasscoreipc
import awsiot.greengrasscoreipc.client as client
from awsiot.greengrasscoreipc.model import GetThingShadowRequest

```

```

TIMEOUT = 10

def sample_get_thing_shadow_request(thingName, shadowName):
    try:
        # set up IPC client to connect to the IPC server
        ipc_client = awsiot.greengrasscoreipc.connect()

        # create the GetThingShadow request
        get_thing_shadow_request = GetThingShadowRequest()
        get_thing_shadow_request.thing_name = thingName
        get_thing_shadow_request.shadow_name = shadowName

        # retrieve the GetThingShadow response after sending the request to the IPC
server
        op = ipc_client.new_get_thing_shadow()
        op.activate(get_thing_shadow_request)
        fut = op.get_response()

        result = fut.result(TIMEOUT)
        return result.payload

    except InvalidArgumentsError as e:
        # add error handling
        ...
    # except ResourceNotFoundError | UnauthorizedError | ServiceError

```

JavaScript

Example Exemple : Get a thing Shadow

```

import {
    GetThingShadowRequest
} from 'aws-iot-device-sdk-v2/dist/greengrasscoreipc/model';
import * as greengrasscoreipc from 'aws-iot-device-sdk-v2/dist/greengrasscoreipc';

class GetThingShadow {
    private ipcClient: greengrasscoreipc.Client;
    private thingName: string;
    private shadowName: string;

    constructor() {
        // Define args parameters here
        this.thingName = "<define_your_own_thingName>";
        this.shadowName = "<define_your_own_shadowName>";
    }
}

```



```
    this.bootstrap();
  }

  async bootstrap() {
    try {
      this.ipcClient = await getIpClient();
    } catch (err) {
      // parse the error depending on your use cases
      throw err
    }

    try {
      await this.handleGetThingShadowOperation(this.thingName,
        this.shadowName);
    } catch (err) {
      // parse the error depending on your use cases
      throw err
    }
  }

  async handleGetThingShadowOperation(
    thingName: string,
    shadowName: string
  ) {
    const request: GetThingShadowRequest = {
      thingName: thingName,
      shadowName: shadowName
    };
    const response = await this.ipcClient.getThingShadow(request);
  }
}

export async function getIpClient() {
  try {
    const ipcClient = greengrasscoreipc.createClient();
    await ipcClient.connect()
      .catch(error => {
        // parse the error depending on your use cases
        throw error;
      });
    return ipcClient
  } catch (err) {
    // parse the error depending on your use cases
    throw err
  }
}
```

```
    }  
  }  
  
  const startScript = new GetThingShadow();
```

UpdateThingShadow

Mettez à jour l'ombre pour l'objet spécifié. Si aucune ombre n'existe, une ombre est créée.

Demande

La demande de cette opération comporte les paramètres suivants :

`thingName(Python :thing_name)`

Nom de l'objet.

Type : `string`

`shadowName(Python :shadow_name)`

Nom du shadow. Pour spécifier l'ombre classique de l'objet, définissez ce paramètre sur une chaîne vide (`""`).

Warning

Le AWS IoT Greengrass service utilise le `AWSManagedGreengrassV2Deployment` nom shadow pour gérer les déploiements qui ciblent des appareils principaux individuels. Cette ombre nommée est réservée à l'usage du AWS IoT Greengrass service. Ne mettez pas à jour ou ne supprimez pas cette ombre nommée.

Type : `string`

`payload`

Le document d'état de la demande sous forme de blob.

Type : objet qui contient les informations suivantes :

`state`

Informations d'état à mettre à jour. Cette opération IPC n'affecte que les champs spécifiés.

Cet objet contient les informations suivantes. Généralement, vous utiliserez la `desired` propriété ou la `reported` propriété, mais pas les deux dans la même demande.

`desired`

Les propriétés d'état et les valeurs dont la mise à jour est demandée dans l'appareil.

Type : map de paires clé-valeur

`reported`

Les propriétés d'état et les valeurs signalées par l'appareil.

Type : map de paires clé-valeur

`clientToken`(Python :`client_token`)

(Facultatif) Le jeton utilisé pour faire correspondre la demande et la réponse correspondante du jeton client.

Type : `string`

`version`

(Facultatif) Version du document fantôme local à mettre à jour. Le service parallèle traite la mise à jour uniquement si la version spécifiée correspond à la dernière version dont il dispose.

Type : `integer`

Réponse

La réponse de cette opération contient les informations suivantes :

`payload`

Le document d'état de réponse sous forme de blob.

Type : `object` qui contient les informations suivantes :

`state`

Les informations sur l'état.

Cet objet contient les informations suivantes.

`desired`

Les propriétés d'état et les valeurs dont la mise à jour est demandée dans l'appareil.

Type : map de paires clé-valeur

`reported`

Les propriétés d'état et les valeurs signalées par l'appareil.

Type : map de paires clé-valeur

`delta`

Les propriétés d'état et les valeurs signalées par l'appareil.

Type : map de paires clé-valeur

`metadata`

Les horodatages de chaque attribut dans les `reported` sections `desired` et afin que vous puissiez déterminer quand l'état a été mis à jour.

Type : `string`

`timestamp`

L'époque, la date et l'heure auxquelles la réponse a été générée.

Type : `integer`

`clientToken(Python :client_token)`

Le jeton utilisé pour faire correspondre la demande et la réponse correspondante.

Type : `string`

`version`

Version du document fantôme local une fois la mise à jour terminée.

Type : `integer`

Erreurs

Cette opération peut renvoyer les erreurs suivantes.

ConflictError

Le service fantôme local a rencontré un conflit de version lors de l'opération de mise à jour. Cela se produit lorsque la version de la charge utile de la demande ne correspond pas à la version du dernier document parallèle local disponible.

InvalidArgumentsError

Le service parallèle local n'est pas en mesure de valider les paramètres de la demande. Cela peut se produire si la demande contient du JSON mal formé ou des caractères non pris en charge.

Une valeur valide `payload` possède les propriétés suivantes :

- Le `state` nœud existe et est un objet qui contient les informations `reported` d'état `desired` ou.
- Les `reported` nœuds `desired` et sont soit des objets, soit des nœuds nuls. Au moins l'un de ces objets doit contenir des informations d'état valides.
- La profondeur des `reported` objets `desired` et ne peut pas dépasser huit nœuds.
- La longueur de la `clientToken` valeur ne peut pas dépasser 64 caractères.
- La `version` valeur doit être 1 ou supérieure.

ServiceError

Une erreur de service interne s'est produite ou le nombre de demandes adressées au service IPC a dépassé les limites spécifiées dans les paramètres de `maxTotalLocalRequestsRate` configuration `maxLocalRequestsPerSecondPerThing` et dans le composant Shadow Manager.

UnauthorizedError

La politique d'autorisation du composant n'inclut pas les autorisations requises pour cette opération.

Exemples

Les exemples suivants montrent comment appeler cette opération dans le code de composant personnalisé.

Java (IPC client V1)

Exemple Exemple : mettre à jour un objet (shadow)

Note

Cet exemple utilise une `IPCUtils` classe pour créer une connexion au service AWS IoT Greengrass Core IPC. Pour de plus amples informations, veuillez consulter [Connectez-vous au service AWS IoT Greengrass Core IPC](#).

```
package com.aws.greengrass.docs.samples.ipc;

import com.aws.greengrass.docs.samples.ipc.util.IPCUtils;
import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClient;
import software.amazon.awssdk.aws.greengrass.UpdateThingShadowResponseHandler;
import software.amazon.awssdk.aws.greengrass.model.UnauthorizedError;
import software.amazon.awssdk.aws.greengrass.model.UpdateThingShadowRequest;
import software.amazon.awssdk.aws.greengrass.model.UpdateThingShadowResponse;
import software.amazon.awssdk.eventstreamrpc.EventStreamRPCConnection;

import java.nio.charset.StandardCharsets;
import java.util.Optional;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;

public class UpdateThingShadow {

    public static final int TIMEOUT_SECONDS = 10;

    public static void main(String[] args) {
        // Use the current core device's name if thing name isn't set.
        String thingName = args[0].isEmpty() ? System.getenv("AWS_IOT_THING_NAME") :
args[0];
        String shadowName = args[1];
        byte[] shadowPayload = args[2].getBytes(StandardCharsets.UTF_8);
        try (EventStreamRPCConnection eventStreamRPCConnection =
            IPCUtils.getEventStreamRpcConnection()) {
            GreengrassCoreIPCClient ipcClient =
                new GreengrassCoreIPCClient(eventStreamRPCConnection);
```

```

        UpdateThingShadowResponseHandler responseHandler =
            UpdateThingShadow.updateThingShadow(ipcClient, thingName,
shadowName,
                shadowPayload);
        CompletableFuture<UpdateThingShadowResponse> futureResponse =
            responseHandler.getResponse();
        try {
            futureResponse.get(TIMEOUT_SECONDS, TimeUnit.SECONDS);
            System.out.printf("Successfully updated shadow: %s/%s\n", thingName,
shadowName);
        } catch (TimeoutException e) {
            System.err.printf("Timeout occurred while updating shadow: %s/%s\n",
thingName,
                shadowName);
        } catch (ExecutionException e) {
            if (e.getCause() instanceof UnauthorizedError) {
                System.err.printf("Unauthorized error while updating shadow: %s/
%s\n",
                    thingName, shadowName);
            } else {
                throw e;
            }
        }
        } catch (InterruptedException e) {
            System.out.println("IPC interrupted.");
        } catch (ExecutionException e) {
            System.err.println("Exception occurred when using IPC.");
            e.printStackTrace();
            System.exit(1);
        }
    }

    public static UpdateThingShadowResponseHandler
updateThingShadow(GreengrassCoreIPCCClient greengrassCoreIPCCClient, String
thingName, String shadowName, byte[] shadowPayload) {
        UpdateThingShadowRequest updateThingShadowRequest = new
UpdateThingShadowRequest();
        updateThingShadowRequest.setThingName(thingName);
        updateThingShadowRequest.setShadowName(shadowName);
        updateThingShadowRequest.setPayload(shadowPayload);
        return greengrassCoreIPCCClient.updateThingShadow(updateThingShadowRequest,
            Optional.empty());
    }
}

```

```
}

```

Python (IPC client V1)

Exemple Exemple : mettre à jour un objet (shadow)

```
import awsiot.greengrasscoreipc
import awsiot.greengrasscoreipc.client as client
from awsiot.greengrasscoreipc.model import UpdateThingShadowRequest

TIMEOUT = 10

def sample_update_thing_shadow_request(thingName, shadowName, payload):
    try:
        # set up IPC client to connect to the IPC server
        ipc_client = awsiot.greengrasscoreipc.connect()

        # create the UpdateThingShadow request
        update_thing_shadow_request = UpdateThingShadowRequest()
        update_thing_shadow_request.thing_name = thingName
        update_thing_shadow_request.shadow_name = shadowName
        update_thing_shadow_request.payload = payload

        # retrieve the UpdateThingShadow response after sending the request to the
        # IPC server
        op = ipc_client.new_update_thing_shadow()
        op.activate(update_thing_shadow_request)
        fut = op.get_response()

        result = fut.result(TIMEOUT)
        return result.payload

    except InvalidArgumentsError as e:
        # add error handling
        ...
    # except ConflictError | UnauthorizedError | ServiceError
```

JavaScript

Exemple Exemple : mettre à jour un objet (shadow)

```
import {
    UpdateThingShadowRequest
} from 'aws-iot-device-sdk-v2/dist/greengrasscoreipc/model';
```



```
import * as greengrasscoreipc from 'aws-iot-device-sdk-v2/dist/greengrasscoreipc';

class UpdateThingShadow {
  private ipcClient: greengrasscoreipc.Client;
  private thingName: string;
  private shadowName: string;
  private shadowDocumentStr: string;

  constructor() {
    // Define args parameters here

    this.thingName = "<define_your_own_thingName>";
    this.shadowName = "<define_your_own_shadowName>";
    this.shadowDocumentStr = "<define_your_own_payload>";

    this.bootstrap();
  }

  async bootstrap() {
    try {
      this.ipcClient = await getIpcClient();
    } catch (err) {
      // parse the error depending on your use cases
      throw err
    }

    try {
      await this.handleUpdateThingShadowOperation(
        this.thingName,
        this.shadowName,
        this.shadowDocumentStr);
    } catch (err) {
      // parse the error depending on your use cases
      throw err
    }
  }

  async handleUpdateThingShadowOperation(
    thingName: string,
    shadowName: string,
    payloadStr: string
  ) {
    const request: UpdateThingShadowRequest = {
      thingName: thingName,
```

```

        shadowName: shadowName,
        payload: payloadStr
    }
    // make the UpdateThingShadow request
    const response = await this.ipcClient.updateThingShadow(request);
}
}

export async function getIpcClient() {
    try {
        const ipcClient = greengrasscoreipc.createClient();
        await ipcClient.connect()
            .catch(error => {
                // parse the error depending on your use cases
                throw error;
            });
        return ipcClient
    } catch (err) {
        // parse the error depending on your use cases
        throw err
    }
}

const startScript = new UpdateThingShadow();

```

DeleteThingShadow

Supprime le shadow de l'objet spécifié.

À partir de la version 2.0.4 du gestionnaire d'ombres, la suppression d'une ombre augmente le numéro de version. Par exemple, lorsque vous supprimez l'ombre MyThingShadow dans la version 1, la version de l'ombre supprimée est 2. Si vous recréez ensuite une ombre portant ce nomMyThingShadow, la version de cette ombre est 3.

Demande

La demande de cette opération comporte les paramètres suivants :

thingName(Python :thing_name)

Nom de l'objet.

Type : string

shadowName(Python :shadow_name)

Nom du shadow. Pour spécifier l'ombre classique de l'objet, définissez ce paramètre sur une chaîne vide ("").

 Warning

Le AWS IoT Greengrass service utilise le `AWSManagedGreengrassV2Deployment` nom shadow pour gérer les déploiements qui ciblent des appareils principaux individuels. Cette ombre nommée est réservée à l'usage du AWS IoT Greengrass service. Ne mettez pas à jour ou ne supprimez pas cette ombre nommée.

Type : string

Réponse

La réponse de cette opération contient les informations suivantes :

payload

Document d'état de réponse vide.

Erreurs

Cette opération peut renvoyer les erreurs suivantes.

`InvalidArgumentsError`

Le service parallèle local n'est pas en mesure de valider les paramètres de la demande. Cela peut se produire si la demande contient du JSON mal formé ou des caractères non pris en charge.

`ResourceNotFoundError`

Le document parallèle local demandé est introuvable.

`ServiceError`

Une erreur de service interne s'est produite ou le nombre de demandes adressées au service IPC a dépassé les limites spécifiées dans les paramètres de `maxTotalLocalRequestsRate`

configuration `maxLocalRequestsPerSecondPerThing` et dans le composant `Shadow Manager`.

UnauthorizedError

La politique d'autorisation du composant n'inclut pas les autorisations requises pour cette opération.

Exemples

Les exemples suivants montrent comment appeler cette opération dans le code de composant personnalisé.

Java (IPC client V1)

Exemple Exemple : supprimer une ombre d'objet

Note

Cet exemple utilise une `IPCUtils` classe pour créer une connexion au service AWS IoT Greengrass Core IPC. Pour de plus amples informations, veuillez consulter [Connectez-vous au service AWS IoT Greengrass Core IPC](#).

```
package com.aws.greengrass.docs.samples.ipc;

import com.aws.greengrass.docs.samples.ipc.util.IPCUtils;
import software.amazon.awssdk.aws.greengrass.DeleteThingShadowResponseHandler;
import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClient;
import software.amazon.awssdk.aws.greengrass.model.DeleteThingShadowRequest;
import software.amazon.awssdk.aws.greengrass.model.DeleteThingShadowResponse;
import software.amazon.awssdk.aws.greengrass.model.ResourceNotFoundError;
import software.amazon.awssdk.aws.greengrass.model.UnauthorizedError;
import software.amazon.awssdk.eventstreamrpc.EventStreamRPCConnection;

import java.util.Optional;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;

public class DeleteThingShadow {
```

```

public static final int TIMEOUT_SECONDS = 10;

public static void main(String[] args) {
    // Use the current core device's name if thing name isn't set.
    String thingName = args[0].isEmpty() ? System.getenv("AWS_IOT_THING_NAME") :
args[0];
    String shadowName = args[1];
    try (EventStreamRPCConnection eventStreamRPCConnection =
        IPCUtils.getEventStreamRpcConnection()) {
        GreengrassCoreIPCClient ipcClient =
            new GreengrassCoreIPCClient(eventStreamRPCConnection);
        DeleteThingShadowResponseHandler responseHandler =
            DeleteThingShadow.deleteThingShadow(ipcClient, thingName,
shadowName);
        CompletableFuture<DeleteThingShadowResponse> futureResponse =
            responseHandler.getResponse();
        try {
            futureResponse.get(TIMEOUT_SECONDS, TimeUnit.SECONDS);
            System.out.printf("Successfully deleted shadow: %s/%s%n", thingName,
shadowName);
        } catch (TimeoutException e) {
            System.err.printf("Timeout occurred while deleting shadow: %s/%s%n",
thingName,
                shadowName);
        } catch (ExecutionException e) {
            if (e.getCause() instanceof UnauthorizedError) {
                System.err.printf("Unauthorized error while deleting shadow: %s/
%s%n",
                    thingName, shadowName);
            } else if (e.getCause() instanceof ResourceNotFoundError) {
                System.err.printf("Unable to find shadow to delete: %s/%s%n",
thingName,
                    shadowName);
            } else {
                throw e;
            }
        }
    } catch (InterruptedException e) {
        System.out.println("IPC interrupted.");
    } catch (ExecutionException e) {
        System.err.println("Exception occurred when using IPC.");
        e.printStackTrace();
        System.exit(1);
    }
}

```

```

    }
}

public static DeleteThingShadowResponseHandler
deleteThingShadow(GreengrassCoreIPCClient greengrassCoreIPCClient, String
thingName, String shadowName) {
    DeleteThingShadowRequest deleteThingShadowRequest = new
DeleteThingShadowRequest();
    deleteThingShadowRequest.setThingName(thingName);
    deleteThingShadowRequest.setShadowName(shadowName);
    return greengrassCoreIPCClient.deleteThingShadow(deleteThingShadowRequest,
Optional.empty());
}
}

```

Python (IPC client V1)

Exemple Exemple : supprimer une ombre d'objet

```

import awsiot.greengrasscoreipc
import awsiot.greengrasscoreipc.client as client
from awsiot.greengrasscoreipc.model import DeleteThingShadowRequest

TIMEOUT = 10

def sample_delete_thing_shadow_request(thingName, shadowName):
    try:
        # set up IPC client to connect to the IPC server
        ipc_client = awsiot.greengrasscoreipc.connect()

        # create the DeleteThingShadow request
        delete_thing_shadow_request = DeleteThingShadowRequest()
        delete_thing_shadow_request.thing_name = thingName
        delete_thing_shadow_request.shadow_name = shadowName

        # retrieve the DeleteThingShadow response after sending the request to the
IPC server
        op = ipc_client.new_delete_thing_shadow()
        op.activate(delete_thing_shadow_request)
        fut = op.get_response()

        result = fut.result(TIMEOUT)
        return result.payload

```

```

except InvalidArgumentsError as e:
    # add error handling
...
# except ResourceNotFoundError | UnauthorizedError | ServiceError

```

JavaScript

Exemple Exemple : supprimer une ombre d'objet

```

import {
    DeleteThingShadowRequest
} from 'aws-iot-device-sdk-v2/dist/greengrasscoreipc/model';
import * as greengrasscoreipc from 'aws-iot-device-sdk-v2/dist/greengrasscoreipc';

class DeleteThingShadow {
    private ipcClient: greengrasscoreipc.Client;
    private thingName: string;
    private shadowName: string;

    constructor() {
        // Define args parameters here
        this.thingName = "<define_your_own_thingName>";
        this.shadowName = "<define_your_own_shadowName>";
        this.bootstrap();
    }

    async bootstrap() {
        try {
            this.ipcClient = await getIpcClient();
        } catch (err) {
            // parse the error depending on your use cases
            throw err
        }

        try {
            await this.handleDeleteThingShadowOperation(this.thingName,
this.shadowName)
        } catch (err) {
            // parse the error depending on your use cases
            throw err
        }
    }

    async handleDeleteThingShadowOperation(thingName: string, shadowName: string) {

```

```
        const request: DeleteThingShadowRequest = {
            thingName: thingName,
            shadowName: shadowName
        }
        // make the DeleteThingShadow request
        const response = await this.ipcClient.deleteThingShadow(request);
    }
}

export async function getIpClient() {
    try {
        const ipcClient = greengrasscoreipc.createClient();
        await ipcClient.connect()
            .catch(error => {
                // parse the error depending on your use cases
                throw error;
            });
        return ipcClient
    } catch (err) {
        // parse the error depending on your use cases
        throw err
    }
}

const startScript = new DeleteThingShadow();
```

ListNamedShadowsForThing

Répertoriez les ombres nommées pour l'objet spécifié.

Demande

La demande de cette opération comporte les paramètres suivants :

`thingName(Python :thing_name)`

Nom de l'objet.

Type : string

`pageSize(Python :page_size)`

(Facultatif) Le nombre de noms fictifs à renvoyer lors de chaque appel.

Type : integer

Par défaut: 25

Maximum : 100

nextToken(Python :next_token)

(Facultatif) Le jeton permettant de récupérer le prochain ensemble de résultats. Cette valeur est renvoyée sur les résultats paginés et est utilisée dans l'appel qui renvoie la page suivante.

Type : string

Réponse

La réponse de cette opération contient les informations suivantes :

results

La liste des noms des ombres.

Type : array

timestamp

(Facultatif) Date et heure auxquelles la réponse a été générée.

Type : integer

nextToken(Python :next_token)

(Facultatif) La valeur du jeton à utiliser dans les demandes paginées pour récupérer la page suivante de la séquence. Ce jeton n'est pas présent lorsqu'il n'y a plus de noms d'ombres à renvoyer.

Type : string

Note

Si le format de page demandé correspond exactement au nombre de noms d'ombres dans la réponse, ce jeton est présent ; toutefois, lorsqu'il est utilisé, il renvoie une liste vide.

Erreurs

Cette opération peut renvoyer les erreurs suivantes.

InvalidArgumentsError

Le service parallèle local n'est pas en mesure de valider les paramètres de la demande. Cela peut se produire si la demande contient du JSON mal formé ou des caractères non pris en charge.

ResourceNotFoundError

Le document parallèle local demandé est introuvable.

ServiceError

Une erreur de service interne s'est produite ou le nombre de demandes adressées au service IPC a dépassé les limites spécifiées dans les paramètres de `maxTotalLocalRequestsRate` configuration `maxLocalRequestsPerSecondPerThing` et dans le composant Shadow Manager.

UnauthorizedError

La politique d'autorisation du composant n'inclut pas les autorisations requises pour cette opération.

Exemples

Les exemples suivants montrent comment appeler cette opération dans le code de composant personnalisé.

Java (IPC client V1)

Exemple Exemple : répertorier un objet nommé ombres

Note

Cet exemple utilise une `IPCUtils` classe pour créer une connexion au service AWS IoT Greengrass Core IPC. Pour de plus amples informations, veuillez consulter [Connectez-vous au service AWS IoT Greengrass Core IPC](#).

```
package com.aws.greengrass.docs.samples.ipc;
```

```
import com.aws.greengrass.docs.samples.ipc.util.IPCUtils;
import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClient;
import
    software.amazon.awssdk.aws.greengrass.ListNamedShadowsForThingResponseHandler;
import software.amazon.awssdk.aws.greengrass.model.ListNamedShadowsForThingRequest;
import
    software.amazon.awssdk.aws.greengrass.model.ListNamedShadowsForThingResponse;
import software.amazon.awssdk.aws.greengrass.model.ResourceNotFoundError;
import software.amazon.awssdk.aws.greengrass.model.UnauthorizedError;
import software.amazon.awssdk.eventstreamrpc.EventStreamRPCConnection;

import java.util.ArrayList;
import java.util.List;
import java.util.Optional;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;

public class ListNamedShadowsForThing {

    public static final int TIMEOUT_SECONDS = 10;

    public static void main(String[] args) {
        // Use the current core device's name if thing name isn't set.
        String thingName = args[0].isEmpty() ? System.getenv("AWS_IOT_THING_NAME") :
args[0];
        try (EventStreamRPCConnection eventStreamRPCConnection =
            IPCUtils.getEventStreamRpcConnection()) {
            GreengrassCoreIPCClient ipcClient =
                new GreengrassCoreIPCClient(eventStreamRPCConnection);
            List<String> namedShadows = new ArrayList<>();
            String nextToken = null;
            try {
                // Send additional requests until there's no pagination token in the
response.
                do {
                    ListNamedShadowsForThingResponseHandler responseHandler =

ListNamedShadowsForThing.listNamedShadowsForThing(ipcClient, thingName,
                                                    nextToken, 25);
                    CompletableFuture<ListNamedShadowsForThingResponse>
futureResponse =
                        responseHandler.getResponse();
```

```

        ListNamedShadowsForThingResponse response =
            futureResponse.get(TIMEOUT_SECONDS, TimeUnit.SECONDS);
        List<String> responseNamedShadows = response.getResults();
        namedShadows.addAll(responseNamedShadows);
        nextToken = response.getNextToken();
    } while (nextToken != null);
    System.out.printf("Successfully got named shadows for thing %s: %s
%n", thingName,
        String.join(", ", namedShadows));
    } catch (TimeoutException e) {
        System.err.println("Timeout occurred while listing named shadows for
thing: " + thingName);
    } catch (ExecutionException e) {
        if (e.getCause() instanceof UnauthorizedError) {
            System.err.println("Unauthorized error while listing named
shadows for " +
                "thing: " + thingName);
        } else if (e.getCause() instanceof ResourceNotFoundError) {
            System.err.println("Unable to find thing to list named shadows:
" + thingName);
        } else {
            throw e;
        }
    }
    } catch (InterruptedException e) {
        System.out.println("IPC interrupted.");
    } catch (ExecutionException e) {
        System.err.println("Exception occurred when using IPC.");
        e.printStackTrace();
        System.exit(1);
    }
}

public static ListNamedShadowsForThingResponseHandler
listNamedShadowsForThing(GreengrassCoreIPCClient greengrassCoreIPCClient, String
thingName, String nextToken, int pageSize) {
    ListNamedShadowsForThingRequest listNamedShadowsForThingRequest =
        new ListNamedShadowsForThingRequest();
    listNamedShadowsForThingRequest.setThingName(thingName);
    listNamedShadowsForThingRequest.setNextToken(nextToken);
    listNamedShadowsForThingRequest.setPageSize(pageSize);
    return
greengrassCoreIPCClient.listNamedShadowsForThing(listNamedShadowsForThingRequest,
Optional.empty());
}

```

```
}
}
```

Python (IPC client V1)

Exemple Exemple : répertorier un objet nommé ombres

```
import awsiot.greengrasscoreipc
import awsiot.greengrasscoreipc.client as client
from awsiot.greengrasscoreipc.model import ListNamedShadowsForThingRequest

TIMEOUT = 10

def sample_list_named_shadows_for_thing_request(thingName, nextToken, pageSize):
    try:
        # set up IPC client to connect to the IPC server
        ipc_client = awsiot.greengrasscoreipc.connect()

        # create the ListNamedShadowsForThingRequest request
        list_named_shadows_for_thing_request = ListNamedShadowsForThingRequest()
        list_named_shadows_for_thing_request.thing_name = thingName
        list_named_shadows_for_thing_request.next_token = nextToken
        list_named_shadows_for_thing_request.page_size = pageSize

        # retrieve the ListNamedShadowsForThingRequest response after sending the
        request to the IPC server
        op = ipc_client.new_list_named_shadows_for_thing()
        op.activate(list_named_shadows_for_thing_request)
        fut = op.get_response()

        list_result = fut.result(TIMEOUT)

        # additional returned fields
        timestamp = list_result.timestamp
        next_token = result.next_token
        named_shadow_list = list_result.results

        return named_shadow_list, next_token, timestamp

    except InvalidArgumentsError as e:
        # add error handling
        ...
    # except ResourceNotFoundError | UnauthorizedError | ServiceError
```

JavaScript

Exemple Exemple : répertorier un objet nommé ombres

```
import {
    ListNamedShadowsForThingRequest
} from 'aws-iot-device-sdk-v2/dist/greengrasscoreipc/model';
import * as greengrasscoreipc from 'aws-iot-device-sdk-v2/dist/greengrasscoreipc';

class listNamedShadowsForThing {
    private ipcClient: greengrasscoreipc.Client;
    private thingName: string;
    private pageSizeStr: string;
    private nextToken: string;

    constructor() {
        // Define args parameters here
        this.thingName = "<define_your_own_thingName>";
        this.pageSizeStr = "<define_your_own_pageSize>";
        this.nextToken = "<define_your_own_token>";
        this.bootstrap();
    }

    async bootstrap() {
        try {
            this.ipcClient = await getIpcClient();
        } catch (err) {
            // parse the error depending on your use cases
            throw err
        }

        try {
            await this.handleListNamedShadowsForThingOperation(this.thingName,
                this.nextToken, this.pageSizeStr);
        } catch (err) {
            // parse the error depending on your use cases
            throw err
        }
    }

    async handleListNamedShadowsForThingOperation(
        thingName: string,
        nextToken: string,
        pageSizeStr: string
```

```
    ) {
      let request: ListNamedShadowsForThingRequest = {
        thingName: thingName,
        nextToken: nextToken,
      };
      if (pageSizeStr) {
        request.pageSize = parseInt(pageSizeStr);
      }
      // make the ListNamedShadowsForThing request
      const response = await this.ipcClient.listNamedShadowsForThing(request);
      const shadowNames = response.results;
    }
  }

export async function getIpcClient(){
  try {
    const ipcClient = greengrasscoreipc.createClient();
    await ipcClient.connect()
      .catch(error => {
        // parse the error depending on your use cases
        throw error;
      });
    return ipcClient
  } catch (err) {
    // parse the error depending on your use cases
    throw err
  }
}

const startScript = new listNamedShadowsForThing();
```

Gérez les déploiements et les composants locaux

Note

Cette fonctionnalité est disponible pour les versions 2.6.0 et ultérieures du composant [Greengrass nucleus](#).

Utilisez le service IPC Greengrass CLI pour gérer les déploiements locaux et les composants Greengrass sur le périphérique principal.

Pour utiliser ces opérations IPC, incluez la version 2.6.0 ou ultérieure du composant Greengrass [CLI en tant que dépendance dans votre composant](#) personnalisé. Vous pouvez ensuite utiliser les opérations IPC dans vos composants personnalisés pour effectuer les opérations suivantes :

- Créez des déploiements locaux pour modifier et configurer les composants Greengrass sur le périphérique principal.
- Redémarrez et arrêtez les composants Greengrass sur le périphérique principal.
- Générez un mot de passe que vous pouvez utiliser pour vous connecter à la [console de débogage locale](#).

Rubriques

- [Versions minimales du SDK](#)
- [Autorisation](#)
- [CreateLocalDeployment](#)
- [ListLocalDeployments](#)
- [GetLocalDeploymentStatus](#)
- [ListComponents](#)
- [GetComponentDetails](#)
- [RestartComponent](#)
- [StopComponent](#)
- [CreateDebugPassword](#)

Versions minimales du SDK

Le tableau suivant répertorie les versions minimales du Kit SDK des appareils AWS IoT que vous devez utiliser pour interagir avec le service IPC Greengrass CLI.

SDK	Version minimale	
Kit SDK des appareils AWS IoT pour Java v2	v1.2.10	
Kit SDK des appareils AWS IoT pour Python v2	v1.5.3	

SDK	Version minimale	
Kit SDK des appareils AWS IoT pour C++ v2	v1.17.0	
Kit SDK des appareils AWS IoT pour JavaScript v2	v1.12.0	

Autorisation

Pour utiliser le service IPC Greengrass CLI dans un composant personnalisé, vous devez définir des politiques d'autorisation qui permettent à votre composant de gérer les déploiements et les composants locaux. Pour plus d'informations sur la définition des politiques d'autorisation, consultez [Autoriser les composants à effectuer des opérations IPC](#).

Les politiques d'autorisation de la Greengrass CLI présentent les propriétés suivantes.

Identifiant du service IPC : `aws.greengrass.Cli`

Opération	Description	Ressources
<code>aws.greengrass#CreateLocalDeployment</code>	Permet à un composant de créer un déploiement local sur le périphérique principal.	*
<code>aws.greengrass#ListLocalDeployments</code>	Permet à un composant de répertorier les déploiements locaux sur le périphérique principal.	*
<code>aws.greengrass#GetLocalDeploymentStatus</code>	Permet à un composant d'obtenir l'état d'un déploiement local sur le périphérique principal.	Un identifiant de déploiement local, ou * pour autoriser l'accès à tous les déploiements locaux.
<code>aws.greengrass#ListComponents</code>	Permet à un composant de répertorier les composants du périphérique principal.	*

Opération	Description	Ressources
<code>aws.greengrass#GetComponentDetails</code>	Permet à un composant d'obtenir des informations sur un composant du périphérique principal.	Un nom de composant, tel que <code>com.example.HelloWorld</code> , ou <code>*</code> pour autoriser l'accès à tous les composants.
<code>aws.greengrass#RestartComponent</code>	Permet à un composant de redémarrer un composant sur le périphérique principal.	Un nom de composant, tel que <code>com.example.HelloWorld</code> , ou <code>*</code> pour autoriser l'accès à tous les composants.
<code>aws.greengrass#StopComponent</code>	Permet à un composant d'arrêter un composant du périphérique principal.	Un nom de composant, tel que <code>com.example.HelloWorld</code> , ou <code>*</code> pour autoriser l'accès à tous les composants.
<code>aws.greengrass#CreateDebugPassword</code>	Permet à un composant de générer un mot de passe à utiliser pour se connecter au composant de console de débogage local .	<code>*</code>

Exemple Exemple de politique d'autorisation

Les exemples de politiques d'autorisation suivants permettent à un composant de créer des déploiements locaux, de visualiser tous les déploiements et composants locaux, ainsi que de redémarrer et d'arrêter un composant nommé `com.example.HelloWorld`

```
{
  "accessControl": {
    "aws.greengrass.Cli": {
      "com.example.MyLocalManagerComponent:cli:1": {
        "policyDescription": "Allows access to create local deployments and view deployments and components.",
        "operations": [
          "aws.greengrass#CreateLocalDeployment",
          "aws.greengrass#ListLocalDeployments",
          "aws.greengrass#GetLocalDeploymentStatus",

```

```

        "aws.greengrass#ListComponents",
        "aws.greengrass#GetComponentDetails"
    ],
    "resources": [
        "*"
    ]
}
},
"aws.greengrass.Cli": {
    "com.example.MyLocalManagerComponent:cli:2": {
        "policyDescription": "Allows access to restart and stop the Hello World
component.",
        "operations": [
            "aws.greengrass#RestartComponent",
            "aws.greengrass#StopComponent"
        ],
        "resources": [
            "com.example.HelloWorld"
        ]
    }
}
}
}
}

```

CreateLocalDeployment

Créez ou mettez à jour un déploiement local à l'aide de recettes de composants, d'artefacts et d'arguments d'exécution spécifiés.

Cette opération fournit les mêmes fonctionnalités que la [commande de création de déploiement](#) dans la CLI Greengrass.

Demande

La demande de cette opération comporte les paramètres suivants :

`recipeDirectoryPath`(Python :`recipe_directory_path`)

(Facultatif) Le chemin absolu vers le dossier contenant les fichiers de recettes de composants.

`artifactDirectoryPath`(Python :`artifact_directory_path`)

(Facultatif) Le chemin absolu vers le dossier contenant les fichiers d'artefacts à inclure dans le déploiement. Le dossier des artefacts doit contenir la structure de dossiers suivante :

```
/path/to/artifact/folder/component-name/component-version/artifacts
```

`rootComponentVersionsToAdd(Python :root_component_versions_to_add)`

(Facultatif) Les versions des composants à installer sur le périphérique principal. Cet objet est une carte qui contient les paires clé-valeur suivantes : `ComponentToVersionMap`

key

Le nom du composant.

value

Version du composant.

`rootComponentsToRemove(Python :root_components_to_remove)`

(Facultatif) Les composants à désinstaller du périphérique principal. Spécifiez une liste dans laquelle chaque entrée est le nom d'un composant.

`componentToConfiguration(Python :component_to_configuration)`

(Facultatif) Les mises à jour de configuration pour chaque composant du déploiement. Cet objet est une carte qui contient les paires clé-valeur suivantes : `ComponentToConfiguration`

key

Le nom du composant.

value

L'objet JSON de mise à jour de configuration pour le composant. L'objet JSON doit avoir le format suivant.

```
{
  "MERGE": {
    "config-key": "config-value"
  },
  "RESET": [
    "path/to/reset/"
  ]
}
```

Pour plus d'informations sur les mises à jour de configuration, consultez [Mettre à jour les configurations des composants](#).

`componentToRunWithInfo(Python :component_to_run_with_info)`

(Facultatif) La configuration d'exécution pour chaque composant du déploiement. Cette configuration inclut l'utilisateur du système qui possède les processus de chaque composant et les limites du système à appliquer à chaque composant. Cet objet est une carte qui contient les paires clé-valeur suivantes : `ComponentToRunWithInfo`

`key`

Le nom du composant.

`value`

Configuration d'exécution du composant. Si vous omettez un paramètre de configuration d'exécution, le logiciel AWS IoT Greengrass Core utilise les valeurs par défaut que vous configurez sur le noyau [Greengrass](#). Cet objet contient `RunWithInfo` les informations suivantes :

`posixUser(Python :posix_user)`

(Facultatif) L'utilisateur du système POSIX et, éventuellement, le groupe à utiliser pour exécuter ce composant sur les périphériques principaux Linux. L'utilisateur et le groupe, s'ils sont spécifiés, doivent exister sur chaque périphérique principal Linux. Spécifiez l'utilisateur et le groupe en les séparant par deux points (:) au format suivant : `user:group`. Le groupe est facultatif. Si vous ne spécifiez aucun groupe, le logiciel AWS IoT Greengrass Core utilise le groupe principal pour l'utilisateur. Pour de plus amples informations, veuillez consulter [Configurer l'utilisateur qui exécute les composants](#).

`windowsUser(Python :windows_user)`

(Facultatif) Utilisateur Windows à utiliser pour exécuter ce composant sur les appareils principaux de Windows. L'utilisateur doit exister sur chaque appareil principal de Windows, et son nom et son mot de passe doivent être stockés dans l'instance Credentials Manager du LocalSystem compte. Pour de plus amples informations, veuillez consulter [Configurer l'utilisateur qui exécute les composants](#).

`systemResourceLimits(Python :system_resource_limits)`

(Facultatif) Les limites de ressources système à appliquer aux processus de ce composant. Vous pouvez appliquer des limites de ressources système aux composants Lambda génériques et non conteneurisés. Pour de plus amples informations, veuillez consulter [Configuration des limites de ressources système pour les composants](#).

AWS IoT Greengrass ne prend actuellement pas en charge cette fonctionnalité sur les appareils Windows principaux.

Cet objet contient `SystemResourceLimits` les informations suivantes :

`cpus`

(Facultatif) Durée maximale du processeur que les processus de ce composant peuvent utiliser sur le périphérique principal. Le temps processeur total d'un appareil principal est équivalent au nombre de cœurs processeurs de l'appareil. Par exemple, sur un périphérique principal doté de 4 cœurs de processeur, vous pouvez définir cette valeur 2 pour limiter les processus de ce composant à 50 % d'utilisation de chaque cœur de processeur. Sur un appareil doté d'un cœur de processeur, vous pouvez définir cette valeur 0.25 pour limiter les processus de ce composant à 25 % d'utilisation du processeur. Si vous définissez cette valeur sur un nombre supérieur au nombre de cœurs de processeur, le logiciel AWS IoT Greengrass Core ne limite pas l'utilisation du processeur par le composant.

`memory`

(Facultatif) La quantité maximale de RAM (en kilo-octets) que les processus de ce composant peuvent utiliser sur le périphérique principal.

`groupName`(Python :`group_name`)

(Facultatif) Nom du groupe d'objets à cibler dans le cadre de ce déploiement.

Réponse

La réponse de cette opération contient les informations suivantes :

`deploymentId`(Python :`deployment_id`)

ID du déploiement local créé par la demande.

ListLocalDeployments

Obtient le statut des 10 derniers déploiements locaux.

Cette opération fournit les mêmes fonctionnalités que la [commande de liste de déploiement](#) dans la CLI Greengrass.

Demande

La demande de cette opération ne comporte aucun paramètre.

Réponse

La réponse de cette opération contient les informations suivantes :

`localDeployments`(Python :`local_deployments`)

La liste des déploiements locaux. Chaque objet de cette liste est un `LocalDeployment` objet qui contient les informations suivantes :

`deploymentId`(Python :`deployment_id`)

ID du déploiement local.

`status`

État du déploiement local. Cette énumération possède `DeploymentStatus` les valeurs suivantes :

- `QUEUED`
- `IN_PROGRESS`
- `SUCCEEDED`
- `FAILED`

GetLocalDeploymentStatus

Obtient le statut d'un déploiement local.

Cette opération fournit les mêmes fonctionnalités que la [commande d'état du déploiement](#) dans la CLI Greengrass.

Demande

La demande de cette opération comporte les paramètres suivants :

`deploymentId`(Python :`deployment_id`)

ID du déploiement local à obtenir.

Réponse

La réponse de cette opération contient les informations suivantes :

deployment

Le déploiement local. Cet objet contient LocalDeployment les informations suivantes :

deploymentId(Python :deployment_id)

ID du déploiement local.

status

État du déploiement local. Cette énumération possède DeploymentStatus les valeurs suivantes :

- QUEUED
- IN_PROGRESS
- SUCCEEDED
- FAILED

ListComponents

Obtient le nom, la version, l'état et la configuration de chaque composant racine sur le périphérique principal. Un composant racine est un composant que vous spécifiez dans un déploiement. Cette réponse n'inclut pas les composants installés en tant que dépendances d'autres composants.

Cette opération fournit les mêmes fonctionnalités que la [commande de liste de composants](#) dans la CLI Greengrass.

Demande

La demande de cette opération ne comporte aucun paramètre.

Réponse

La réponse de cette opération contient les informations suivantes :

components

Liste des composants racine du périphérique principal. Chaque objet de cette liste est un ComponentDetails objet qui contient les informations suivantes :

`componentName(Python :component_name)`

Le nom du composant.

`version`

Version du composant.

`state`

État du composant. Cet état peut être l'un des suivants :

- BROKEN
- ERRORED
- FINISHED
- INSTALLED
- NEW
- RUNNING
- STARTING
- STOPPING

`configuration`

Configuration du composant en tant qu'objet JSON.

GetComponentDetails

Obtient la version, le statut et la configuration d'un composant sur le périphérique principal.

Cette opération fournit les mêmes fonctionnalités que la [commande component details](#) de la CLI Greengrass.

Demande

La demande de cette opération comporte les paramètres suivants :

`componentName(Python :component_name)`

Le nom du composant à obtenir.

Réponse

La réponse de cette opération contient les informations suivantes :

`componentDetails(Python :component_details)`

Les détails du composant. Cet objet contient `ComponentDetails` les informations suivantes :

`componentName(Python :component_name)`

Le nom du composant.

`version`

Version du composant.

`state`

État du composant. Cet état peut être l'un des suivants :

- `BROKEN`
- `ERRORED`
- `FINISHED`
- `INSTALLED`
- `NEW`
- `RUNNING`
- `STARTING`
- `STOPPING`

`configuration`

Configuration du composant en tant qu'objet JSON.

RestartComponent

Redémarre un composant sur le périphérique principal.

Note

Bien que vous puissiez redémarrer n'importe quel composant, nous vous recommandons de ne redémarrer que [les composants génériques](#).

Cette opération fournit les mêmes fonctionnalités que la [commande de redémarrage du composant](#) dans la CLI Greengrass.

Demande

La demande de cette opération comporte les paramètres suivants :

`componentName`(Python :`component_name`)

Le nom du composant.

Réponse

La réponse de cette opération contient les informations suivantes :

`restartStatus`(Python :`restart_status`)

État de la demande de redémarrage. Le statut de la demande peut être l'un des suivants :

- SUCCEEDED
- FAILED

`message`

Un message expliquant pourquoi le composant n'a pas pu redémarrer, en cas d'échec de la demande.

StopComponent

Arrête les processus d'un composant sur le périphérique principal.

Note

Bien que vous puissiez arrêter n'importe quel composant, nous vous recommandons de n'arrêter que [les composants génériques](#).

Cette opération fournit les mêmes fonctionnalités que la [commande d'arrêt du composant](#) dans la CLI Greengrass.

Demande

La demande de cette opération comporte les paramètres suivants :

`componentName(Python :component_name)`

Le nom du composant.

Réponse

La réponse de cette opération contient les informations suivantes :

`stopStatus(Python :stop_status)`

État de la demande d'arrêt. Le statut de la demande peut être l'un des suivants :

- SUCCEEDED
- FAILED

`message`

Un message expliquant pourquoi le composant n'a pas pu s'arrêter, en cas d'échec de la demande.

CreateDebugPassword

Génère un mot de passe aléatoire que vous pouvez utiliser pour vous connecter au [composant de console de débogage local](#). Le mot de passe expire 8 heures après sa création.

Cette opération fournit les mêmes fonctionnalités que la [get-debug-password commande](#) de la CLI Greengrass.

Demande

La demande de cette opération ne comporte aucun paramètre.

Réponse

La réponse de cette opération contient les informations suivantes :

`username`

Le nom d'utilisateur à utiliser pour se connecter.

password

Le mot de passe à utiliser pour se connecter.

passwordExpiration(Python :password_expiration)

Heure à laquelle le mot de passe expire.

certificateSHA256Hash(Python :certificate_sha256_hash)

L'empreinte SHA-256 du certificat autosigné utilisé par la console de débogage locale lorsque le protocole HTTPS est activé. Lorsque vous ouvrez la console de débogage locale, utilisez cette empreinte pour vérifier que le certificat est légitime et que la connexion est sécurisée.

certificateSHA1Hash(Python :certificate_sha1_hash)

L'empreinte SHA-1 du certificat autosigné utilisé par la console de débogage locale lorsque le protocole HTTPS est activé. Lorsque vous ouvrez la console de débogage locale, utilisez cette empreinte pour vérifier que le certificat est légitime et que la connexion est sécurisée.

Authentifier et autoriser les appareils clients

Note

Cette fonctionnalité est disponible pour les versions 2.6.0 et ultérieures du composant [Greengrass](#) nucleus.

Utilisez le service IPC d'authentification des appareils clients pour développer un composant de courtier local personnalisé auquel les appareils IoT locaux, tels que les appareils clients, peuvent se connecter.

Pour utiliser ces opérations IPC, incluez la version 2.2.0 ou ultérieure du composant d'[authentification du périphérique client en tant que dépendance dans votre composant](#) personnalisé. Vous pouvez ensuite utiliser les opérations IPC dans vos composants personnalisés pour effectuer les opérations suivantes :

- Vérifiez l'identité des appareils clients qui se connectent au périphérique principal.
- Créez une session pour qu'un appareil client se connecte au périphérique principal.
- Vérifiez si un appareil client est autorisé à effectuer une action.
- Recevez une notification lorsque le certificat de serveur de l'appareil principal change.

Rubriques

- [Versions minimales du SDK](#)
- [Autorisation](#)
- [VerifyClientDeviceIdentity](#)
- [GetClientDeviceAuthToken](#)
- [AuthorizeClientDeviceAction](#)
- [SubscribeToCertificateUpdates](#)

Versions minimales du SDK

Le tableau suivant répertorie les versions minimales du service IPC Kit SDK des appareils AWS IoT que vous devez utiliser pour interagir avec le service IPC d'authentification de l'appareil client.

SDK	Version minimale	
Kit SDK des appareils AWS IoT pour Java v2	v1.9.3	
Kit SDK des appareils AWS IoT pour Python v2	v1.11.3	
Kit SDK des appareils AWS IoT pour C++ v2	v1.18.3	
Kit SDK des appareils AWS IoT pour JavaScript v2	v1.12.0	

Autorisation

Pour utiliser le service IPC d'authentification de l'appareil client dans un composant personnalisé, vous devez définir des politiques d'autorisation permettant à votre composant d'effectuer ces opérations. Pour plus d'informations sur la définition des politiques d'autorisation, consultez [Autoriser les composants à effectuer des opérations IPC](#).

Les politiques d'autorisation pour l'authentification et l'autorisation des appareils clients présentent les propriétés suivantes.

Identifiant du service IPC : `aws.greengrass.clientdevices.Auth`

Opération	Description	Ressources
<code>aws.greengrass#VerifyClientDeviceIdentity</code>	Permet à un composant de vérifier l'identité d'un appareil client.	*
<code>aws.greengrass#GetClientDeviceAuthToken</code>	Permet à un composant de valider les informations d'identification d'un appareil client et de créer une session pour cet appareil client.	*
<code>aws.greengrass#AuthorizeClientDeviceAction</code>	Permet à un composant de vérifier si un appareil client est autorisé à effectuer une action.	*
<code>aws.greengrass#SubscribeToCertificateUpdates</code>	Permet à un composant de recevoir des notifications lorsque le certificat de serveur de l'appareil principal change.	*
*	Permet à un composant d'effectuer toutes les opérations de service IPC d'authentification du périphérique client.	*

Exemples de politiques d'autorisation

Vous pouvez vous référer à l'exemple de politique d'autorisation suivant pour vous aider à configurer les politiques d'autorisation pour vos composants.

Exemple Exemple de politique d'autorisation

L'exemple de politique d'autorisation suivant permet à un composant d'effectuer toutes les opérations IPC d'authentification du périphérique client.

```
{
  "accessControl": {
    "aws.greengrass.clientdevices.Auth": {
      "com.example.MyLocalBrokerComponent:clientdevices:1": {
        "policyDescription": "Allows access to authenticate and authorize client
devices.",
        "operations": [
          "aws.greengrass#VerifyClientDeviceIdentity",
          "aws.greengrass#GetClientDeviceAuthToken",
          "aws.greengrass#AuthorizeClientDeviceAction",
          "aws.greengrass#SubscribeToCertificateUpdates"
        ],
        "resources": [
          "*"
        ]
      }
    }
  }
}
```

VerifyClientDeviceIdentity

Vérifiez l'identité d'un appareil client. Cette opération vérifie si le périphérique client est valide AWS IoT .

Demande

La demande de cette opération comporte les paramètres suivants :

`credential`

Les informations d'identification de l'appareil client. Cet objet contient

`ClientDeviceCredential` les informations suivantes :

`clientDeviceCertificate`(Python :`client_device_certificate`)

Le certificat de périphérique X.509 de l'appareil client.

Réponse

La réponse de cette opération contient les informations suivantes :

`isValidClientDevice(Python :is_valid_client_device)`

Si l'identité de l'appareil client est valide.

GetClientDeviceAuthToken

Valide les informations d'identification d'un appareil client et crée une session pour l'appareil client. Cette opération renvoie un jeton de session que vous pouvez utiliser dans les demandes suivantes pour [autoriser les actions de l'appareil client](#).

Pour connecter correctement un appareil client, le [composant d'authentification du périphérique client](#) doit accorder l'`mqtt:connect` autorisation pour l'ID client utilisé par le périphérique client.

Demande

La demande de cette opération comporte les paramètres suivants :

`credential`

Les informations d'identification de l'appareil client. Cet objet contient `CredentialDocument` les informations suivantes :

`mqttCredential(Python :mqtt_credential)`

Les informations d'identification MQTT de l'appareil client. Spécifiez l'ID client et le certificat que l'appareil client utilise pour se connecter. Cet objet contient `MQTTCredential` les informations suivantes :

`clientId(Python :client_id)`

L'ID client à utiliser pour se connecter.

`certificatePem(Python :certificate_pem)`

Le certificat de périphérique X.509 à utiliser pour la connexion.

`username`

Note

Cette propriété n'est pas utilisée actuellement.

password

 Note

Cette propriété n'est pas utilisée actuellement.

Réponse

La réponse de cette opération contient les informations suivantes :

`clientDeviceAuthToken(Python :client_device_auth_token)`

Le jeton de session pour l'appareil client. Vous pouvez utiliser ce jeton de session dans les demandes suivantes pour autoriser les actions de cet appareil client.

AuthorizeClientDeviceAction

Vérifiez si un appareil client est autorisé à effectuer une action sur une ressource. Les politiques d'autorisation des appareils clients spécifient les autorisations que les appareils clients peuvent exécuter lorsqu'ils sont connectés à un périphérique principal. Vous définissez les politiques d'autorisation de l'appareil client lorsque vous configurez le [composant d'authentification du périphérique client](#).

Demande

La demande de cette opération comporte les paramètres suivants :

`clientDeviceAuthToken(Python :client_device_auth_token)`

Le jeton de session pour l'appareil client.

`operation`

L'opération à autoriser.

`resource`

Ressource sur laquelle le dispositif client effectue l'opération.

Réponse

La réponse de cette opération contient les informations suivantes :

`isAuthorized(Python :is_authorized)`

Si le dispositif client est autorisé à effectuer l'opération sur la ressource.

SubscribeToCertificateUpdates

Abonnez-vous pour recevoir le nouveau certificat de serveur de l'appareil principal à chaque rotation. Lorsque le certificat de serveur change, les courtiers doivent le recharger à l'aide du nouveau certificat de serveur.

Le [composant d'authentification de l'appareil client](#) effectue une rotation des certificats de serveur tous les 7 jours par défaut. Vous pouvez configurer l'intervalle de rotation entre 2 et 10 jours.

Il s'agit d'une opération d'abonnement dans le cadre de laquelle vous vous abonnez à un flux de messages d'événements. Pour utiliser cette opération, définissez un gestionnaire de réponse au flux avec des fonctions qui gèrent les messages d'événements, les erreurs et la fermeture du flux. Pour de plus amples informations, veuillez consulter [Abonnez-vous aux diffusions d'événements IPC](#).

Type de message d'événement : `CertificateUpdateEvent`

Demande

La demande de cette opération comporte les paramètres suivants :

`certificateOptions(Python :certificate_options)`

Les types de mises à jour de certificats auxquels vous devez vous abonner. Cet objet contient `CertificateOptions` les informations suivantes :

`certificateType(Python :certificate_type)`

Type de mises à jour de certificat auxquelles vous devez vous abonner. Choisissez l'option suivante :

- `SERVER`

Réponse

La réponse de cette opération contient les informations suivantes :

messages

Le flux de messages. Cet objet contient `CertificateUpdateEvent` les informations suivantes :

`certificateUpdate(Python :certificate_update)`

Les informations relatives au nouveau certificat. Cet objet contient `CertificateUpdate` les informations suivantes :

`certificate`

Certificat.

`privateKey(Python :private_key)`

La clé privée du certificat.

`publicKey(Python :public_key)`

La clé publique du certificat.

`caCertificates(Python :ca_certificates)`

Liste des certificats de l'autorité de certification (CA) de la chaîne de certificats de l'autorité de certification du certificat.

Interagissez avec les appareils IoT locaux

Les appareils clients sont des appareils IoT locaux qui se connectent et communiquent avec un appareil principal de Greengrass via MQTT. Vous pouvez connecter les appareils clients aux appareils principaux pour effectuer les opérations suivantes :

- Interagissez avec les messages MQTT dans les composants Greengrass.
- Relayer les messages et les données entre les appareils clients et AWS IoT Core.
- Interagissez avec les ombres des appareils clients dans les composants Greengrass.
- Synchronisez les ombres des appareils clients avec AWS IoT Core.
- À utiliser IPv6 pour la messagerie locale.

Pour se connecter à un appareil principal, les appareils clients peuvent utiliser le cloud discovery. Les appareils clients se connectent au service AWS IoT Greengrass cloud pour récupérer des informations sur les principaux appareils auxquels ils peuvent se connecter. Ils peuvent ensuite se connecter à un appareil principal pour traiter leurs messages et synchroniser leurs données avec le service AWS IoT Core cloud.

Vous pouvez suivre un didacticiel qui explique comment configurer un appareil principal pour se connecter et communiquer avec un AWS IoT objet. Ce didacticiel explique également comment développer un composant Greengrass personnalisé qui interagit avec les appareils clients. Pour de plus amples informations, veuillez consulter [Tutoriel : Interagissez avec des appareils IoT locaux via MQTT](#).

Rubriques

- [AWS-composants de l'appareil client fournis](#)
- [Connect les appareils clients aux appareils principaux](#)
- [Relayer les messages MQTT entre les appareils clients et AWS IoT Core](#)
- [Interagir avec les appareils clients dans les composants](#)
- [Interagissez avec les ombres des appareils clients et synchronisez-les](#)
- [Utilisation IPv6 pour la messagerie locale](#)
- [Résolution des problèmes liés aux appareils clients](#)

AWS-composants de l'appareil client fournis

AWS IoT Greengrass fournit les composants publics suivants que vous pouvez déployer sur les appareils principaux. Ces composants permettent aux appareils clients de se connecter et de communiquer avec un périphérique principal.

Note

Plusieurs composants AWS fournis dépendent de versions mineures spécifiques du noyau Greengrass. En raison de cette dépendance, vous devez mettre à jour ces composants lorsque vous mettez à jour le noyau Greengrass vers une nouvelle version mineure. Pour plus d'informations sur les versions spécifiques du noyau dont dépend chaque composant, consultez la rubrique correspondante sur les composants. Pour plus d'informations sur la mise à jour du noyau, consultez [Mettre à jour le logiciel AWS IoT Greengrass principal \(OTA\)](#).

Lorsqu'un composant possède un type de composant à la fois générique et Lambda, la version actuelle du composant est le type générique et une version précédente du composant est le type Lambda.

Composant	Description	Type de composant	Systèmes d'exploitation pris en charge	Open source
Authentification de l'appareil client	Permet aux appareils IoT locaux, appelés appareils clients, de se connecter à l'appareil principal.	Plugin	Linux, Windows	Oui
Détecteur IP	Transmet les informations	Plugin	Linux, Windows	Oui

Composant	Description	Type de composant	Systèmes d'exploitation pris en charge	Open source
	de connectivité au AWS IoT Greengrass courtier MQTT afin que les appareils clients puissent découvrir comment se connecter.			
Pont MQTT	Relaie les messages MQTT entre les appareils clients, AWS IoT Greengrass publie/abonnement locaux, et. AWS IoT Core	Plugin	Linux, Windows	Oui
Courtier MQTT 3.1.1 (Moquette)	Exécute un broker MQTT 3.1.1 qui gère les messages entre les appareils clients et le périphérique principal.	Plugin	Linux, Windows	Oui

Composant	Description	Type de composant	Systèmes d'exploitation pris en charge	Open source
Courtier MQTT 5 (EMQX)	Exécute un broker MQTT 5 qui gère les messages entre les appareils clients et le périphérique principal.	Générique	Linux, Windows	Non
Shadow Manager	Permet l'interaction avec les ombres sur le périphérique principal. Il gère le stockage des documents cachés ainsi que la synchronisation des états d'ombre locaux avec le service AWS IoT Device Shadow.	Plugin	Linux, Windows	Oui

Connect les appareils clients aux appareils principaux

Vous pouvez configurer Cloud Discovery pour connecter les appareils clients aux appareils principaux. Lorsque vous configurez la découverte du cloud, les appareils clients peuvent se connecter au service AWS IoT Greengrass cloud pour récupérer des informations sur les principaux appareils auxquels ils peuvent se connecter. Les appareils clients peuvent ensuite tenter de se connecter à chaque périphérique principal jusqu'à ce qu'ils se connectent correctement.

Pour utiliser Cloud Discovery, vous devez effectuer les opérations suivantes :

- Associez les appareils clients aux appareils principaux auxquels ils peuvent se connecter.
- Spécifiez les points de terminaison du broker MQTT auxquels les appareils clients peuvent se connecter à chaque périphérique principal.
- Déployez des composants sur le périphérique principal qui permettent la prise en charge des appareils clients.

Vous pouvez également déployer des composants facultatifs pour effectuer les opérations suivantes :

- Relayez les messages entre les appareils clients, les composants de Greengrass et le service AWS IoT Core cloud.
- Gérez automatiquement pour vous les points de terminaison du broker MQTT du périphérique principal.
- Gérez les ombres des appareils clients locaux et synchronisez les ombres avec le service AWS IoT Core cloud.

Vous devez également revoir et mettre à jour la AWS IoT politique de l'appareil principal afin de vérifier qu'il dispose des autorisations requises pour connecter les appareils clients. Pour de plus amples informations, veuillez consulter [Prérequis](#).

Après avoir configuré Cloud Discovery, vous pouvez tester les communications entre un appareil client et un appareil principal. Pour de plus amples informations, veuillez consulter [Tester les communications entre appareils clients](#).

Rubriques

- [Prérequis](#)
- [Composants Greengrass pour le support des appareils clients](#)

- [Configuration de la découverte du cloud \(console\)](#)
- [Configurer la découverte du cloud \(AWS CLI\)](#)
- [Associer des appareils clients](#)
- [Authentification des clients hors ligne](#)
- [Gérez les principaux points de terminaison des appareils](#)
- [Choisissez un courtier MQTT](#)
- [Connexion des appareils clients à un périphérique AWS IoT Greengrass Core avec un courtier MQTT](#)
- [Tester les communications entre appareils clients](#)
- [API de découverte Greengrass RESTful](#)

Prérequis

Pour connecter des appareils clients à un appareil principal, vous devez disposer des éléments suivants :

- Le périphérique principal doit exécuter [Greengrass nucleus](#) v2.2.0 ou version ultérieure.
- Le rôle de service Greengrass qui vous est associé AWS IoT Greengrass Compte AWS dans la AWS région où fonctionne l'appareil principal. Pour de plus amples informations, veuillez consulter [Configurer le rôle de service Greengrass](#).
- La AWS IoT politique de l'appareil principal doit autoriser les autorisations suivantes :
 - `greengrass:PutCertificateAuthorities`
 - `greengrass:VerifyClientDeviceIdentity`
 - `greengrass:VerifyClientDeviceIoTCertificateAssociation`
 - `greengrass:GetConnectivityInfo`
 - `greengrass:UpdateConnectivityInfo`— (Facultatif) Cette autorisation est requise pour utiliser le [composant de détection IP](#), qui transmet les informations de connectivité réseau de l'appareil principal au service AWS IoT Greengrass cloud.
 - `iot:GetThingShadow`, `iot:UpdateThingShadow`, et `iot:DeleteThingShadow` — (Facultatif) Ces autorisations sont requises pour utiliser le [composant Shadow Manager](#) afin de synchroniser les ombres du périphérique client avec AWS IoT Core. [Cette fonctionnalité nécessite Greengrass nucleus v2.6.0 ou version ultérieure, Shadow Manager v2.2.0 ou version ultérieure et MQTT bridge v2.2.0 ou version ultérieure.](#)

Pour de plus amples informations, veuillez consulter [Configuration de la AWS IoT politique des objets](#).

 Note

Si vous avez utilisé la AWS IoT politique par défaut lors de [l'installation du logiciel AWS IoT Greengrass Core](#), le périphérique principal dispose d'une AWS IoT politique qui autorise l'accès à toutes les AWS IoT Greengrass actions (`greengrass:*`).

- AWS IoT objets que vous pouvez connecter en tant qu'appareils clients. Pour plus d'informations, consultez la section [Création de AWS IoT ressources](#) dans le guide du AWS IoT Core développeur.
- L'appareil client doit se connecter à l'aide d'un ID client. Un identifiant client est un nom d'objet. Aucun autre identifiant client ne sera accepté.
- La AWS IoT politique de chaque appareil client doit autoriser `l'greengrass:Discover` autorisation. Pour de plus amples informations, veuillez consulter [AWS IoT Politique minimale pour les appareils clients](#).

Rubriques

- [Configurer le rôle de service Greengrass](#)
- [Configuration de la AWS IoT politique des objets](#)

Configurer le rôle de service Greengrass

Le rôle de service Greengrass est un rôle de service Gestion des identités et des accès AWS (IAM) qui autorise l'accès AWS IoT Greengrass aux ressources des AWS services en votre nom. Ce rôle permet de vérifier l'identité AWS IoT Greengrass des appareils clients et de gérer les informations de connectivité de base des appareils.

Si vous n'avez pas encore configuré le [rôle de service Greengrass](#) dans cette région, vous devez associer un rôle de service Greengrass à votre Compte AWS rôle dans cette région. AWS IoT Greengrass

Lorsque vous utilisez la page Configurer la découverte des appareils principaux de la [AWS IoT Greengrass console](#), vous AWS IoT Greengrass configurez le rôle de service Greengrass pour vous. Sinon, vous pouvez le configurer manuellement à l'aide de la [AWS IoT console](#) ou de AWS IoT Greengrass l'API.

Dans cette section, vous allez vérifier si le rôle de service Greengrass est configuré. S'il n'est pas configuré, vous créez un nouveau rôle de service Greengrass auquel vous pourrez vous associer AWS IoT Greengrass Compte AWS dans cette région.

Configuration du rôle de service Greengrass (console)

1. Vérifiez si le rôle de service Greengrass vous est associé AWS IoT Greengrass Compte AWS dans cette région. Procédez comme suit :

- a. Accédez à la [console AWS IoT](#).
- b. Dans le panneau de navigation, sélectionnez Settings (Paramètres).
- c. Dans la section Rôle de service Greengrass, recherchez Rôle de service actuel pour voir si un rôle de service Greengrass est associé.

Si un rôle de service Greengrass est associé, vous répondez à cette exigence pour utiliser le composant de détection IP. Passez à [Configuration de la AWS IoT politique des objets](#).

2. Si le rôle de service Greengrass n'est pas associé à votre AWS IoT Greengrass compte Compte AWS dans cette région, créez un rôle de service Greengrass et associez-le. Procédez comme suit :

- a. Accédez à la [Console IAM](#).
- b. Sélectionnez Roles.
- c. Choisissez Créer un rôle.
- d. Sur la page Créer un rôle, procédez comme suit :
 - i. Sous Type d'entité de confiance, sélectionnez Service AWS.
 - ii. Sous Cas d'utilisation, Cas d'utilisation pour les autres Services AWS, choisissez Greengrass, sélectionnez Greengrass. Cette option indique d'ajouter AWS IoT Greengrass en tant qu'entité de confiance capable d'assumer ce rôle.
 - iii. Choisissez Suivant.
 - iv. Sous Politiques d'autorisations, sélectionnez le `AWSGreengrassResourceAccessRolePolicy` à associer au rôle.
 - v. Choisissez Suivant.
 - vi. Dans le champ Nom du rôle, saisissez un nom unique pour votre rôle, par exemple **Greengrass_ServiceRole**.
 - vii. Choisissez Créer un rôle.

- e. Accédez à la [console AWS IoT](#).
- f. Dans le panneau de navigation, sélectionnez Settings (Paramètres).
- g. Dans la section Rôle de service Greengrass, choisissez Attacher un rôle.
- h. Dans le mode Update Greengrass Service Role, sélectionnez le rôle IAM que vous avez créé, puis choisissez Attacher un rôle.

Configurer le rôle de service Greengrass ()AWS CLI

1. Vérifiez si le rôle de service Greengrass vous est associé AWS IoT Greengrass Compte AWS dans cette région.

```
aws greengrassv2 get-service-role-for-account
```

Si le rôle de service Greengrass est associé, l'opération renvoie une réponse contenant des informations sur le rôle.

Si un rôle de service Greengrass est associé, vous répondez à cette exigence pour utiliser le composant de détection IP. Passez à [Configuration de la AWS IoT politique des objets](#).

2. Si le rôle de service Greengrass n'est pas associé à votre AWS IoT Greengrass compte Compte AWS dans cette région, créez un rôle de service Greengrass et associez-le. Procédez comme suit :
 - a. Créez un rôle avec une politique de confiance qui permet 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 clés `aws:SourceArn` contextuelles et les clés de contexte de condition `aws:SourceAccount` globale dans votre politique de confiance afin d'éviter tout problème de sécurité secondaire confus. Les clés contextuelles de condition limitent l'accès pour autoriser uniquement les demandes provenant du compte spécifié et de l'espace de travail Greengrass. Pour de plus amples informations sur le problème de l'adjoint confus, veuillez consulter [Prévention du problème de l'adjoint confus entre services](#).

Linux or Unix

```
aws iam create-role --role-name Greengrass_ServiceRole --assume-role-policy-document '{
  "Version": "2012-10-17",
```

```

"Statement": [
  {
    "Effect": "Allow",
    "Principal": {
      "Service": "greengrass.amazonaws.com"
    },
    "Action": "sts:AssumeRole",
    "Condition": {
      "ArnLike": {
        "aws:SourceArn": "arn:aws:greengrass:region:account-id:*"
      },
      "StringEquals": {
        "aws:SourceAccount": "account-id"
      }
    }
  }
]
}'

```

Windows Command Prompt (CMD)

```

aws iam create-role --role-name Greengrass_ServiceRole --assume-role-policy-
document "{\"Version\":\"2012-10-17\", \"Statement\": [{\"Effect
\": \"Allow\", \"Principal\": {\"Service\": \"greengrass.amazonaws.com
\"}, \"Action\": \"sts:AssumeRole\", \"Condition\": {\"ArnLike\":
{\"aws:SourceArn\": \"arn:aws:greengrass:region:account-id:*\"}, \
\"StringEquals\": {\"aws:SourceAccount\": \"account-id\"}}}]}"

```

PowerShell

```

aws iam create-role --role-name Greengrass_ServiceRole --assume-role-policy-
document '{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "greengrass.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "ArnLike": {
          "aws:SourceArn": "arn:aws:greengrass:region:account-id:*"
        }
      }
    }
  ]
}'

```

```
    },
    "StringEquals": {
      "aws:SourceAccount": "account-id"
    }
  }
}
```

- b. Copiez l'ARN de rôle des métadonnées dans la sortie. Vous utilisez l'ARN pour associer le rôle à votre compte.
- c. 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
```

- d. Associez le rôle de service Greengrass à AWS IoT Greengrass votre. Compte AWS Remplacez *role-arn* par l'ARN du rôle de service.

```
aws greengrassv2 associate-service-role-to-account --role-arn role-arn
```

L'opération renvoie la réponse suivante en cas de succès.

```
{
  "associatedAt": "timestamp"
}
```

Configuration de la AWS IoT politique des objets

Les appareils principaux utilisent des certificats de périphérique X.509 pour autoriser les connexions à AWS. Vous associez AWS IoT des politiques aux certificats d'appareil afin de définir les autorisations pour un appareil principal. Pour plus d'informations, consultez [AWS IoT politiques relatives aux opérations du plan de données](#) et [AWS IoT Politique minimale de prise en charge des appareils clients](#).

Pour connecter des appareils clients à un appareil principal, la AWS IoT politique du périphérique principal doit autoriser les autorisations suivantes :

- `greengrass:PutCertificateAuthorities`

- `greengrass:VerifyClientDeviceIdentity`
- `greengrass:VerifyClientDeviceIoTCertificateAssociation`
- `greengrass:GetConnectivityInfo`
- `greengrass:UpdateConnectivityInfo`— (Facultatif) Cette autorisation est requise pour utiliser le [composant de détection IP](#), qui transmet les informations de connectivité réseau de l'appareil principal au service AWS IoT Greengrass cloud.
- `iot:GetThingShadow`, `iot:UpdateThingShadow`, et `iot:DeleteThingShadow` — (Facultatif) Ces autorisations sont requises pour utiliser le [composant Shadow Manager](#) afin de synchroniser les ombres du périphérique client avec AWS IoT Core. [Cette fonctionnalité nécessite Greengrass noyau v2.6.0 ou version ultérieure, Shadow Manager v2.2.0 ou version ultérieure et MQTT bridge v2.2.0 ou version ultérieure.](#)

Dans cette section, vous passez en revue les AWS IoT politiques de votre appareil principal et ajoutez les autorisations requises manquantes. Si vous avez utilisé le [programme d'installation du logiciel AWS IoT Greengrass Core pour provisionner des ressources](#), votre appareil principal dispose d'une AWS IoT politique qui autorise l'accès à toutes les AWS IoT Greengrass actions (`greengrass:*`). Dans ce cas, vous devez mettre à jour la AWS IoT politique uniquement si vous prévoyez de déployer le composant Shadow Manager avec lequel synchroniser les ombres des appareils AWS IoT Core. Sinon, vous pouvez ignorer cette section.

Configuration de la AWS IoT politique des objets (console)

1. Dans le menu de navigation de la [AWS IoT Greengrass console](#), choisissez Core devices.
2. Sur la page des appareils principaux, choisissez le périphérique principal à mettre à jour.
3. Sur la page de détails de l'appareil principal, choisissez le lien vers l'objet de l'appareil principal. Ce lien ouvre la page des détails de l'objet dans la AWS IoT console.
4. Sur la page des détails de l'objet, sélectionnez Certificats.
5. Dans l'onglet Certificats, choisissez le certificat actif de l'objet.
6. Sur la page des détails du certificat, sélectionnez Politiques.
7. Dans l'onglet Politiques, choisissez la AWS IoT politique à revoir et à mettre à jour. Vous pouvez ajouter les autorisations requises à n'importe quelle politique associée au certificat actif de l'appareil principal.

 Note

Si vous avez utilisé le [programme d'installation du logiciel AWS IoT Greengrass Core pour provisionner des ressources](#), vous avez deux AWS IoT règles. Nous vous recommandons de choisir la politique nommée GreengrassV2IoTThingPolicy, si elle existe. Les appareils principaux que vous créez avec le programme d'installation rapide utilisent ce nom de politique par défaut. Si vous ajoutez des autorisations à cette politique, vous les accordez également aux autres appareils principaux qui utilisent cette politique.

8. Dans l'aperçu des politiques, choisissez Modifier la version active.
9. Passez en revue la politique relative aux autorisations requises et ajoutez les autorisations requises manquantes.
 - `greengrass:PutCertificateAuthorities`
 - `greengrass:VerifyClientDeviceIdentity`
 - `greengrass:VerifyClientDeviceIoTCertificateAssociation`
 - `greengrass:GetConnectivityInfo`
 - `greengrass:UpdateConnectivityInfo`— (Facultatif) Cette autorisation est requise pour utiliser le [composant de détection IP](#), qui transmet les informations de connectivité réseau de l'appareil principal au service AWS IoT Greengrass cloud.
 - `iot:GetThingShadow`, `iot:UpdateThingShadow`, et `iot:DeleteThingShadow` — (Facultatif) Ces autorisations sont requises pour utiliser le [composant Shadow Manager](#) afin de synchroniser les ombres du périphérique client avec AWS IoT Core. [Cette fonctionnalité nécessite Greengrass nucleus v2.6.0 ou version ultérieure, Shadow Manager v2.2.0 ou version ultérieure et MQTT bridge v2.2.0 ou version ultérieure.](#)
10. (Facultatif) Pour autoriser le périphérique principal à synchroniser les ombres avec AWS IoT Core, ajoutez l'instruction suivante à la politique. Si vous prévoyez d'interagir avec les ombres de l'appareil client, mais que vous ne les synchronisez pas AWS IoT Core, ignorez cette étape. Remplacez *region* et *account-id* par la région que vous utilisez et votre Compte AWS numéro.
 - Cet exemple d'instruction permet d'accéder aux ombres de tous les appareils. Pour suivre les meilleures pratiques de sécurité, vous pouvez restreindre l'accès uniquement au périphérique principal et aux appareils clients que vous connectez au périphérique principal. Pour de plus

amples informations, veuillez consulter [AWS IoT Politique minimale de prise en charge des appareils clients](#).

```
{
  "Effect": "Allow",
  "Action": [
    "iot:GetThingShadow",
    "iot:UpdateThingShadow",
    "iot:DeleteThingShadow"
  ],
  "Resource": [
    "arn:aws:iot:region:account-id:thing/*"
  ]
}
```

Après avoir ajouté cette déclaration, le document de politique peut ressembler à l'exemple suivant.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Connect",
        "iot:Publish",
        "iot:Subscribe",
        "iot:Receive",
        "greengrass:*"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:GetThingShadow",
        "iot:UpdateThingShadow",
        "iot:DeleteThingShadow"
      ],

```

```

    "Resource": [
      "arn:aws:iot:us-east-1:123456789012:thing/*"
    ]
  }
]
}

```

11. 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.
12. Sélectionnez Enregistrer comme nouvelle version.

Configurer la politique AWS IoT des objets (AWS CLI)

1. Énumérez les principes de base de l'appareil AWS IoT . Les principaux de l'objet peuvent être des certificats de périphériques X.509 ou d'autres identifiants. Exécutez la commande suivante et *MyGreengrassCore* remplacez-la par le nom du périphérique principal.

```
aws iot list-thing-principals --thing-name MyGreengrassCore
```

L'opération renvoie une réponse répertoriant les principaux éléments du périphérique principal.

```

{
  "principals": [
    "arn:aws:iot:us-west-2:123456789012:cert/certificateId"
  ]
}

```

2. Identifiez le certificat actif de l'appareil principal. Exécutez la commande suivante et remplacez-la *certificateId* par l'ID de chaque certificat de l'étape précédente jusqu'à ce que vous trouviez le certificat actif. L'ID du certificat est la chaîne hexadécimale à la fin de l'ARN du certificat. L'--query argument indique de n'afficher que le statut du certificat.

```
aws iot describe-certificate --certificate-id certificateId --query
'certificateDescription.status'
```

L'opération renvoie le statut du certificat sous forme de chaîne. Par exemple, si le certificat est actif, cette opération produit des résultats "ACTIVE".

3. AWS IoT Répertoriez les politiques associées au certificat. Exécutez la commande suivante et remplacez l'ARN du certificat par l'ARN du certificat.

```
aws iot list-principal-policies --principal arn:aws:iot:us-west-2:123456789012:cert/certificateId
```

L'opération renvoie une réponse répertoriant les AWS IoT politiques associées au certificat.

```
{
  "policies": [
    {
      "policyName":
        "GreengrassTESCertificatePolicyMyGreengrassCoreTokenExchangeRoleAlias",
      "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassTESCertificatePolicyMyGreengrassCoreTokenExchangeRoleAlias"
    },
    {
      "policyName": "GreengrassV2IoTThingPolicy",
      "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassV2IoTThingPolicy"
    }
  ]
}
```

4. Choisissez la politique à consulter et à mettre à jour.

Note

Si vous avez utilisé le [programme d'installation du logiciel AWS IoT Greengrass Core pour provisionner des ressources](#), vous avez deux AWS IoT règles. Nous vous recommandons de choisir la politique nommée GreengrassV2IoTThingPolicy, si elle existe. Les appareils principaux que vous créez avec le programme d'installation rapide utilisent ce nom de politique par défaut. Si vous ajoutez des autorisations à cette politique, vous les accordez également aux autres appareils principaux qui utilisent cette politique.

5. Obtenez le document de la politique. Exécutez la commande suivante et *GreengrassV2IoTThingPolicy* remplacez-la par le nom de la politique.

```
aws iot get-policy --policy-name GreengrassV2IoTThingPolicy
```

L'opération renvoie une réponse contenant le document de la politique et d'autres informations relatives à la politique. Le document de politique est un objet JSON sérialisé sous forme de chaîne.

```
{
  "policyName": "GreengrassV2IoTThingPolicy",
  "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/GreengrassV2IoTThingPolicy",
  "policyDocument": "{\\
    \\\"Version\\\": \\\"2012-10-17\\\",\\
    \\\"Statement\\\": [\\
      {\\
        \\\"Effect\\\": \\\"Allow\\\",\\
        \\\"Action\\\": [\\
          \\\"iot:Connect\\\",\\
          \\\"iot:Publish\\\",\\
          \\\"iot:Subscribe\\\",\\
          \\\"iot:Receive\\\",\\
          \\\"greengrass:*\\\"\\
        ],\\
        \\\"Resource\\\": \\\"*\\\"\\
      }\\
    ]\\
  }\",
  "defaultVersionId": "1",
  "creationDate": "2021-02-05T16:03:14.098000-08:00",
  "lastModifiedDate": "2021-02-05T16:03:14.098000-08:00",
  "generationId":
    "f19144b798534f52c619d44f771a354f1b957dfa2b850625d9f1d0fde530e75f"
}
```

6. Utilisez un convertisseur en ligne ou un autre outil pour convertir la chaîne du document de politique en objet JSON, puis enregistrez-la dans un fichier nommé `iot-policy.json`.

Par exemple, si l'outil [jq](#) est installé, vous pouvez exécuter la commande suivante pour obtenir le document de politique, le convertir en objet JSON et enregistrer le document de politique en tant qu'objet JSON.

```
aws iot get-policy --policy-name GreengrassV2IoTThingPolicy --query
'policyDocument' | jq fromjson >> iot-policy.json
```

7. Passez en revue la politique relative aux autorisations requises et ajoutez les autorisations requises manquantes.

Par exemple, sur un système basé sur Linux, vous pouvez exécuter la commande suivante pour utiliser GNU nano pour ouvrir le fichier.

```
nano iot-policy.json
```

- `greengrass:PutCertificateAuthorities`
 - `greengrass:VerifyClientDeviceIdentity`
 - `greengrass:VerifyClientDeviceIoTCertificateAssociation`
 - `greengrass:GetConnectivityInfo`
 - `greengrass:UpdateConnectivityInfo`— (Facultatif) Cette autorisation est requise pour utiliser le [composant de détection IP](#), qui transmet les informations de connectivité réseau de l'appareil principal au service AWS IoT Greengrass cloud.
 - `iot:GetThingShadow`, `iot:UpdateThingShadow`, et `iot:DeleteThingShadow` — (Facultatif) Ces autorisations sont requises pour utiliser le [composant Shadow Manager](#) afin de synchroniser les ombres du périphérique client avec AWS IoT Core. [Cette fonctionnalité nécessite Greengrass nucleus v2.6.0 ou version ultérieure, Shadow Manager v2.2.0 ou version ultérieure et MQTT bridge v2.2.0 ou version ultérieure.](#)
8. Enregistrez les modifications en tant que nouvelle version de la politique. Exécutez la commande suivante et *GreengrassV2IoTThingPolicy* remplacez-la par le nom de la politique.

```
aws iot create-policy-version --policy-name GreengrassV2IoTThingPolicy --policy-document file://iot-policy.json --set-as-default
```

L'opération renvoie une réponse similaire à l'exemple suivant en cas de succès.

```
{
  "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/GreengrassV2IoTThingPolicy",
  "policyDocument": "{\n  \\\\\"Version\\\\\\": \\\\\"2012-10-17\\\\\\",\n  \\\\\"Statement\\\\\\": [\n    {\n      \\\\\"Effect\\\\\\": \\\\\"Allow\\\\\\",\n      \\\\\"Action\\\\\\": [\n
```

```

\\t\\t\\t\\"iot:Connect\\",\
\\t\\t\\t\\"iot:Publish\\",\
\\t\\t\\t\\"iot:Subscribe\\",\
\\t\\t\\t\\"iot:Receive\\",\
\\t\\t\\t\\"greengrass:*\\",\
    ],\
    \\\"Resource\\\": \\\"*\\\"\\
  ]\
]\
},\
  \"policyVersionId\": \"2\",
  \"isDefaultVersion\": true
}

```

Composants Greengrass pour le support des appareils clients

Important

Le périphérique principal doit exécuter [Greengrass nucleus](#) v2.2.0 ou version ultérieure pour prendre en charge les appareils clients.

Pour permettre aux appareils clients de se connecter et de communiquer avec un appareil principal, vous déployez les composants Greengrass suivants sur le périphérique principal :

- [Authentification de l'appareil client](#) (`aws.greengrass.clientdevices.Auth`)

Déployez le composant d'authentification des appareils clients pour authentifier les appareils clients et autoriser les actions des appareils clients. Ce composant permet à vos AWS IoT objets de se connecter à un appareil principal.

Ce composant nécessite une certaine configuration pour pouvoir être utilisé. Vous devez spécifier les groupes de périphériques clients et les opérations que chaque groupe est autorisé à effectuer, telles que la connexion et la communication via MQTT. Pour plus d'informations, consultez la section [Configuration du composant d'authentification de l'appareil client](#).

- [Courtier MQTT 3.1.1 \(Moquette\)](#) (`aws.greengrass.clientdevices.mqtt.Moquette`)

Déployez le composant de courtier MQTT Moquette pour exécuter un courtier MQTT léger. Le broker Moquette MQTT est conforme à MQTT 3.1.1 et inclut un support local pour QoS 0,

QoS 1, QoS 2, les messages conservés, les messages de dernière volonté et les abonnements persistants.

Vous n'êtes pas obligé de configurer ce composant pour l'utiliser. Cependant, vous pouvez configurer le port sur lequel ce composant fait fonctionner le broker MQTT. Par défaut, il utilise le port 8883.

- [Courtier MQTT 5 \(EMQX\)](#) (`aws.greengrass.clientdevices.mqtt.EMQX`)

 Note

Pour utiliser le broker EMQX MQTT 5, vous devez utiliser [Greengrass noyau v2.6.0 ou version ultérieure et client device](#) auth v2.2.0 ou version ultérieure.

Déployez le composant broker EMQX MQTT pour utiliser les fonctionnalités MQTT 5.0 dans la communication entre les appareils clients et le périphérique principal. Le broker MQTT EMQX est compatible avec MQTT 5.0 et inclut la prise en charge des intervalles d'expiration des sessions et des messages, des propriétés utilisateur, des abonnements partagés, des alias de rubrique, etc.

Vous n'êtes pas obligé de configurer ce composant pour l'utiliser. Cependant, vous pouvez configurer le port sur lequel ce composant fait fonctionner le broker MQTT. Par défaut, il utilise le port 8883.

- [Pont MQTT](#) (`aws.greengrass.clientdevices.mqtt.Bridge`)

(Facultatif) Déployez le composant pont MQTT pour relayer les messages entre les appareils clients (MQTT local), publication/abonnement locaux et MQTT. AWS IoT Core Configurez ce composant pour synchroniser les appareils clients avec les appareils clients AWS IoT Core et interagir avec eux à partir des composants Greengrass.

Ce composant nécessite une configuration pour être utilisé. Vous devez spécifier les mappages de rubriques dans lesquels ce composant relaie les messages. Pour plus d'informations, consultez la section [Configuration des composants du pont MQTT](#).

- [Détecteur IP](#) (`aws.greengrass.clientdevices.IPDetector`)

(Facultatif) Déployez le composant de détection IP pour signaler automatiquement les points de terminaison du broker MQTT du périphérique principal au service AWS IoT Greengrass cloud.

Vous ne pouvez pas utiliser ce composant si votre configuration réseau est complexe, par exemple si un routeur transmet le port du broker MQTT au périphérique principal.

Vous n'êtes pas obligé de configurer ce composant pour l'utiliser.

- [Shadow Manager](#) (`aws.greengrass.ShadowManager`)

 Note

Pour gérer les ombres des appareils clients, vous devez utiliser Greengrass nucleus v2.6.0 ou version ultérieure, Shadow Manager v2.2.0 ou version ultérieure et MQTT bridge v2.2.0 ou version ultérieure.

(Facultatif) Déployez le composant Shadow Manager pour gérer les ombres du périphérique client sur le périphérique principal. Les composants Greengrass peuvent obtenir, mettre à jour et supprimer les ombres des appareils clients afin d'interagir avec les appareils clients. Vous pouvez également configurer le composant Shadow Manager pour synchroniser les ombres du périphérique client avec le service AWS IoT Core cloud.

Pour utiliser ce composant avec les ombres des périphériques clients, vous devez configurer le composant du pont MQTT pour relayer les messages entre les appareils clients et le gestionnaire des ombres, qui utilise la publication et l'abonnement locaux. Dans le cas contraire, ce composant ne nécessite pas de configuration pour être utilisé, mais il nécessite une configuration pour synchroniser les ombres de l'appareil.

 Note

Nous vous recommandons de ne déployer qu'un seul composant de broker MQTT. Le [pont MQTT](#) et les composants du [détecteur IP](#) ne fonctionnent qu'avec un seul composant de courtier MQTT à la fois. Si vous déployez plusieurs composants du broker MQTT, vous devez les configurer pour utiliser différents ports.

Configuration de la découverte du cloud (console)

Vous pouvez utiliser la AWS IoT Greengrass console pour associer des appareils clients, gérer les points de terminaison des appareils principaux et déployer des composants afin de permettre la prise en charge des appareils clients. Pour de plus amples informations, veuillez consulter [Étape 2 : activer la prise en charge des appareils clients](#).

Configurer la découverte du cloud (AWS CLI)

Vous pouvez utiliser le AWS Command Line Interface (AWS CLI) pour associer des appareils clients, gérer les points de terminaison des appareils principaux et déployer des composants afin de permettre la prise en charge des appareils clients. Pour plus d'informations, consultez les ressources suivantes :

- [Gérer les associations entre appareils clients \(AWS CLI\)](#)
- [Gérez les principaux points de terminaison des appareils](#)
- [AWS-composants de l'appareil client fournis](#)
- [Créer des déploiements](#)

Associer des appareils clients

Pour utiliser la découverte du cloud, associez les appareils clients à un appareil principal afin qu'ils puissent découvrir le périphérique principal. Ils peuvent ensuite utiliser l'[API de découverte Greengrass](#) pour récupérer les informations de connectivité et les certificats pour leurs principaux appareils associés.

De même, dissociez les appareils clients d'un périphérique principal pour les empêcher de découvrir le périphérique principal.

Rubriques

- [Gérer les associations entre appareils clients \(console\)](#)
- [Gérer les associations entre appareils clients \(AWS CLI\)](#)
- [Gérer les associations client-appareil \(API\)](#)

Gérer les associations entre appareils clients (console)

Vous pouvez utiliser la AWS IoT Greengrass console pour afficher, ajouter et supprimer des associations client-appareil.

Pour afficher les associations de périphériques clients pour un périphérique principal (console)

1. Accédez à la [console AWS IoT Greengrass](#).
2. Choisissez les appareils Core.

3. Choisissez l'appareil principal à gérer.
4. Sur la page de détails de l'appareil principal, choisissez l'onglet Appareils clients.
5. Dans la section Appareils clients associés, vous pouvez voir quels appareils clients (AWS IoT objets) sont associés au périphérique principal.

Pour associer des appareils clients à un périphérique principal (console)

1. Accédez à la [console AWS IoT Greengrass](#).
2. Choisissez les appareils Core.
3. Choisissez l'appareil principal à gérer.
4. Sur la page de détails de l'appareil principal, choisissez l'onglet Appareils clients.
5. Dans la section Appareils clients associés, choisissez Associer les appareils clients.
6. Dans le mode Associer les appareils clients au périphérique principal, procédez comme suit pour chaque appareil client à associer :
 - a. Entrez le nom de l' AWS IoT objet à associer en tant qu'appareil client.
 - b. Choisissez Ajouter.
7. Choisissez Associer.

Les appareils clients que vous avez associés peuvent désormais utiliser l'API de découverte Greengrass pour découvrir cet appareil principal.

Pour dissocier les appareils clients d'un périphérique principal (console)

1. Accédez à la [console AWS IoT Greengrass](#).
2. Choisissez les appareils Core.
3. Choisissez l'appareil principal à gérer.
4. Sur la page de détails de l'appareil principal, choisissez l'onglet Appareils clients.
5. Dans la section Appareils clients associés, sélectionnez chaque appareil client à dissocier.
6. Choisissez Dissocier.
7. Dans le mode de confirmation, choisissez Dissocier.

Les appareils clients que vous avez dissociés ne peuvent plus utiliser l'API de découverte Greengrass pour découvrir cet appareil principal.

Gérer les associations entre appareils clients (AWS CLI)

Vous pouvez utiliser le AWS Command Line Interface (AWS CLI) pour gérer les associations d'appareils clients pour un appareil principal.

Pour afficher les associations de périphériques clients pour un périphérique principal (AWS CLI)

- Utilisez la commande suivante : [list-client-devices-associated- with-core-device](#).

Pour associer des appareils clients à un périphérique principal (AWS CLI)

- Utilisez la commande suivante : [batch-associate-client-device- with-core-device](#).

Pour dissocier les appareils clients d'un périphérique principal (AWS CLI)

- Utilisez la commande suivante : [batch-disassociate-client-device- from-core-device](#).

Gérer les associations client-appareil (API)

Vous pouvez utiliser l' AWS API pour gérer les associations d'appareils clients pour un appareil principal.

Pour afficher les associations entre appareils clients pour un appareil principal (AWS API)

- Utilisez l'opération suivante : [ListClientDevicesAssociatedWithCoreDevice](#)

Pour associer des appareils clients à un périphérique principal (AWS API)

- Utilisez l'opération suivante : [BatchAssociateClientDeviceWithCoreDevice](#)

Pour dissocier les appareils clients d'un périphérique principal (AWS API)

- Utilisez l'opération suivante : [BatchDisassociateClientDeviceFromCoreDevice](#)

Authentification des clients hors ligne

Avec l'authentification hors ligne, vous pouvez configurer votre appareil AWS IoT Greengrass Core afin que les appareils clients puissent se connecter à un appareil principal, même lorsque celui-

ci n'est pas connecté au cloud. Lorsque vous utilisez l'authentification hors ligne, vos appareils Greengrass peuvent continuer à fonctionner dans un environnement partiellement hors ligne.

Pour utiliser l'authentification hors ligne pour un appareil client connecté au cloud, vous devez disposer des éléments suivants :

- Un appareil AWS IoT Greengrass Core sur lequel le [Authentification de l'appareil client](#) composant est déployé. Vous devez utiliser la version 2.3.0 ou supérieure pour l'authentification hors ligne.
- Une connexion au cloud pour le périphérique principal lors de la connexion initiale des appareils clients.

Stockage des informations d'identification client

Lorsqu'un dispositif client se connecte à un dispositif central pour la première fois, celui-ci appelle le AWS IoT Greengrass service. Lorsqu'il est appelé, Greengrass valide l'enregistrement de l'appareil client comme un objet. AWS IoT Cela permet également de vérifier que l'appareil possède un certificat valide. Le dispositif central stocke ensuite ces informations localement.

La prochaine fois que l'appareil se connecte, le périphérique principal de Greengrass tente de valider l'appareil client auprès du AWS IoT Greengrass service. S'il ne parvient pas à se connecter AWS IoT Greengrass, le périphérique principal utilise ses informations de périphérique stockées localement pour valider le périphérique client.

Vous pouvez configurer la durée pendant laquelle le périphérique principal Greengrass stocke les informations d'identification. [Vous pouvez définir le délai d'une minute à 2 147 483 647 minutes en définissant l'option de configuration dans la `clientDeviceTrustDurationMinutes` configuration du composant d'authentification du périphérique client.](#) La durée par défaut est d'une minute, ce qui désactive effectivement l'authentification hors ligne. Lorsque vous définissez ce délai, nous vous recommandons de prendre en compte vos besoins en matière de sécurité. Vous devez également tenir compte de la durée pendant laquelle vous prévoyez que les appareils principaux fonctionneront lorsqu'ils sont déconnectés du cloud.

Le périphérique principal met à jour son stockage d'informations d'identification à trois reprises :

1. Lorsqu'un appareil se connecte au périphérique principal pour la première fois.
2. Si le périphérique principal est connecté au cloud, lorsqu'un appareil client se reconnecte au périphérique principal.

3. Si l'appareil principal est connecté au cloud, actualisez l'intégralité de la banque d'informations d'identification une fois par jour.

Lorsque le périphérique principal Greengrass actualise son magasin d'informations d'identification, il utilise l'opération. [ListClientDevicesAssociatedWithCoreDevice](#) Greengrass actualise uniquement les appareils renvoyés par cette opération. Pour associer un appareil client à un périphérique principal, consultez [Associer des appareils clients](#).

Pour utiliser l'`ListClientDevicesAssociatedWithCoreDevice` opération, vous devez ajouter une autorisation pour l'opération au rôle Gestion des identités et des accès AWS (IAM) associé à l'opération Compte AWS qui s'exécute AWS IoT Greengrass. Pour de plus amples informations, veuillez consulter [Autoriser les appareils principaux à interagir avec les AWS services](#).

Gérez les principaux points de terminaison des appareils

Lorsque vous utilisez la découverte du cloud, vous stockez les points de terminaison du broker MQTT pour les appareils principaux dans le service AWS IoT Greengrass cloud. Les appareils clients se connectent AWS IoT Greengrass pour récupérer ces points de terminaison et d'autres informations relatives à leurs périphériques principaux associés.

Pour chaque appareil principal, vous pouvez gérer les points de terminaison automatiquement ou manuellement.

- Gérez automatiquement les points de terminaison avec le détecteur IP

Vous pouvez déployer le [composant de détection IP](#) pour gérer automatiquement les points de terminaison des appareils principaux à votre place si votre configuration réseau n'est pas complexe, par exemple si les appareils clients se trouvent sur le même réseau que le périphérique principal. Vous ne pouvez pas utiliser le composant de détection IP si le périphérique principal se trouve derrière un routeur qui transmet le port du broker MQTT au périphérique principal, par exemple.

Le composant de détection IP est également utile si vous déployez dans des groupes d'objets, car il gère les points de terminaison de tous les appareils principaux du groupe d'objets. Pour de plus amples informations, veuillez consulter [Utiliser le détecteur IP pour gérer automatiquement les points de terminaison](#).

- Gérer manuellement les points de terminaison

Si vous ne pouvez pas utiliser le composant de détection IP, vous devez gérer manuellement les principaux points de terminaison de l'appareil. Vous pouvez mettre à jour ces points de terminaison à l'aide de la console ou de l'API. Pour de plus amples informations, veuillez consulter [Gérer manuellement les points de terminaison](#).

Rubriques

- [Utiliser le détecteur IP pour gérer automatiquement les points de terminaison](#)
- [Gérer manuellement les points de terminaison](#)

Utiliser le détecteur IP pour gérer automatiquement les points de terminaison

Si vous disposez d'une configuration réseau simple, telle que les appareils clients sur le même réseau que le périphérique principal, vous pouvez déployer le [composant de détection IP](#) pour effectuer les opérations suivantes :

- Surveillez les informations de connectivité au réseau local de l'appareil central Greengrass. Ces informations incluent les points de terminaison réseau du périphérique principal et le port sur lequel fonctionne le broker MQTT.
- Signalez les informations de connectivité de l'appareil principal au service AWS IoT Greengrass cloud.

Le composant du détecteur IP remplace les points de terminaison que vous avez définis manuellement.

Important

La AWS IoT politique de l'appareil principal doit `greengrass:UpdateConnectivityInfo` autoriser l'utilisation du composant de détection IP. Pour plus d'informations, consultez [AWS IoT politiques relatives aux opérations du plan de données](#) et [Configuration de la AWS IoT politique des objets](#).

Vous pouvez effectuer l'une des opérations suivantes pour déployer le composant de détection IP :

- Utilisez la page Configurer la découverte de la console. Pour de plus amples informations, veuillez consulter [Configuration de la découverte du cloud \(console\)](#).

- Créez et révisiez les déploiements pour inclure le détecteur IP. Vous pouvez utiliser la console ou l' AWS CLI AWS API pour gérer les déploiements. Pour de plus amples informations, veuillez consulter [Créer des déploiements](#).

Déployer le composant de détection IP (console)

1. Dans le menu de navigation de la [AWS IoT Greengrass console](#), sélectionnez Composants.
2. Sur la page Composants, choisissez l'onglet Composants publics, puis choisissez `aws.greengrass.clientdevices.IPDetector`.
3. Dans la page `aws.greengrass.clientdevices.IPDetector`, choisissez Déployer.
4. Dans Ajouter au déploiement, choisissez un déploiement existant à réviser ou choisissez de créer un nouveau déploiement, puis choisissez Suivant.
5. Si vous avez choisi de créer un nouveau déploiement, choisissez le périphérique principal ou le groupe d'objets cible pour le déploiement. Sur la page Spécifier la cible, sous Cible de déploiement, choisissez un périphérique principal ou un groupe d'objets, puis cliquez sur Suivant.
6. Sur la page Sélectionner les composants, vérifiez que `aws.greengrass.clientdevices.IPDetector` le composant est sélectionné, choisissez Next.
7. Sur la page Configurer les composants, sélectionnez `aws.greengrass.clientdevices.IPDetector`, puis effectuez les opérations suivantes :
 - a. Choisissez Configure component (Configurer un composant).
 - b. Dans la configuration `aws.greengrass.clientdevices.IPDetector` modal, sous Mise à jour de la configuration, dans Configuration à fusionner, vous pouvez saisir une mise à jour de configuration pour configurer le composant du détecteur IP. Vous pouvez définir l'une des options de configuration suivantes :
 - `defaultPort`— (Facultatif) Le port du broker MQTT à signaler lorsque ce composant détecte des adresses IP. Vous devez spécifier ce paramètre si vous configurez le broker MQTT pour utiliser un port différent du port par défaut 8883.
 - `includeIPv4LoopbackAddrs`— (Facultatif) Vous pouvez activer cette option pour détecter et signaler les adresses de IPv4 boucle. Il s'agit d'adresses IP, par exemple `localhost`, où un appareil peut communiquer avec lui-même. Utilisez cette option dans les environnements de test dans lesquels le périphérique principal et le périphérique client s'exécutent sur le même système.

- `includeIPv4LinkLocalAddr`— (Facultatif) Vous pouvez activer cette option pour détecter et signaler les adresses IPv4 [locales liées aux liens](#). Utilisez cette option si le réseau du périphérique principal ne dispose pas du protocole DHCP (Dynamic Host Configuration Protocol) ou d'adresses IP attribuées de manière statique.
- `includeIPv6LoopbackAddr`— (Facultatif) Vous pouvez activer cette option pour détecter et signaler les adresses de IPv6 boucle. Il s'agit d'adresses IP, par exemple `localhost`, où un appareil peut communiquer avec lui-même. Utilisez cette option dans les environnements de test dans lesquels le périphérique principal et le périphérique client s'exécutent sur le même système. Vous devez `includeIPv4Addr` définir sur `false` et `includeIPv6Addr` sur `true` pour utiliser cette option. Vous devez disposer du détecteur IP v2.2.0 ou version ultérieure pour utiliser cette option.
- `includeIPv6LinkLocalAddr`— (Facultatif) Vous pouvez activer cette option pour détecter et signaler les adresses IPv6 [locales liées aux liens](#). Utilisez cette option si le réseau du périphérique principal ne dispose pas du protocole DHCP (Dynamic Host Configuration Protocol) ou d'adresses IP attribuées de manière statique. Vous devez `includeIPv4Addr` définir sur `false` et `includeIPv6Addr` sur `true` pour utiliser cette option. Vous devez disposer du détecteur IP v2.2.0 ou version ultérieure pour utiliser cette option.
- `includeIPv4Addr`— (Facultatif) La valeur par défaut est définie sur `true`. Vous pouvez activer cette option pour publier IPv4 les adresses présentes sur l'appareil principal. Vous devez disposer du détecteur IP v2.2.0 ou version ultérieure pour utiliser cette option.
- `includeIPv6Addr`— (Facultatif) Vous pouvez activer cette option pour publier IPv6 les adresses trouvées sur l'appareil principal. Réglez `false` sur `includeIPv4Addr` pour utiliser cette option. Vous devez disposer du détecteur IP v2.2.0 ou version ultérieure pour utiliser cette option.

La mise à jour de configuration peut ressembler à l'exemple suivant.

```
{
  "defaultPort": "8883",
  "includeIPv4LoopbackAddr": false,
  "includeIPv4LinkLocalAddr": false
}
```

- c. Choisissez Confirmer pour fermer le modal, puis cliquez sur Suivant.

8. Sur la page Configure advanced settings (Configurer les paramètres avancés), conservez les paramètres de configuration par défaut et choisissez Next (Suivant).
9. Sur la page Review (Révision), choisissez Deploy (Déployer).

Le déploiement peut prendre jusqu'à une minute.

Déployer le composant du détecteur IP (AWS CLI)

Pour déployer le composant du détecteur IP, créez un document de déploiement qui inclut `aws.greengrass.clientdevices.IPDetector` l'componentsobjet et spécifiez la mise à jour de configuration du composant. Suivez les instructions [Créer des déploiements](#) pour créer un nouveau déploiement ou modifier un déploiement existant.

Vous pouvez spécifier l'une des options suivantes pour configurer le composant du détecteur IP lorsque vous créez le document de déploiement :

- `defaultPort`— (Facultatif) Le port du broker MQTT à signaler lorsque ce composant détecte des adresses IP. Vous devez spécifier ce paramètre si vous configurez le broker MQTT pour utiliser un port différent du port par défaut 8883.
- `includeIPv4LoopbackAddr`— (Facultatif) Vous pouvez activer cette option pour détecter et signaler les adresses de IPv4 boucle. Il s'agit d'adresses IP, par exemple `localhost`, où un appareil peut communiquer avec lui-même. Utilisez cette option dans les environnements de test dans lesquels le périphérique principal et le périphérique client s'exécutent sur le même système.
- `includeIPv4LinkLocalAddr`— (Facultatif) Vous pouvez activer cette option pour détecter et signaler les adresses IPv4 [locales liées aux liens](#). Utilisez cette option si le réseau du périphérique principal ne dispose pas du protocole DHCP (Dynamic Host Configuration Protocol) ou d'adresses IP attribuées de manière statique.
- `includeIPv6LoopbackAddr`— (Facultatif) Vous pouvez activer cette option pour détecter et signaler les adresses de IPv6 boucle. Il s'agit d'adresses IP, par exemple `localhost`, où un appareil peut communiquer avec lui-même. Utilisez cette option dans les environnements de test dans lesquels le périphérique principal et le périphérique client s'exécutent sur le même système. Vous devez `includeIPv4Addr` définir sur `false` et `includeIPv6Addr` sur `true` pour utiliser cette option. Vous devez disposer du détecteur IP v2.2.0 ou version ultérieure pour utiliser cette option.
- `includeIPv6LinkLocalAddr`— (Facultatif) Vous pouvez activer cette option pour détecter et signaler les adresses IPv6 [locales liées aux liens](#). Utilisez cette option si le réseau du périphérique

principal ne dispose pas du protocole DHCP (Dynamic Host Configuration Protocol) ou d'adresses IP attribuées de manière statique. Vous devez `includeIPv4Addrs` définir sur `false` et `includeIPv6Addrs` sur `true` pour utiliser cette option. Vous devez disposer du détecteur IP v2.2.0 ou version ultérieure pour utiliser cette option.

- `includeIPv4Addrs`— (Facultatif) La valeur par défaut est définie sur `true`. Vous pouvez activer cette option pour publier IPv4 les adresses présentes sur l'appareil principal. Vous devez disposer du détecteur IP v2.2.0 ou version ultérieure pour utiliser cette option.
- `includeIPv6Addrs`— (Facultatif) Vous pouvez activer cette option pour publier IPv6 les adresses trouvées sur l'appareil principal. Réglez `false` sur `includeIPv4Addrs` pour utiliser cette option. Vous devez disposer du détecteur IP v2.2.0 ou version ultérieure pour utiliser cette option.

L'exemple de document de déploiement partiel suivant indique de signaler le port 8883 comme port du broker MQTT.

```
{
  ...,
  "components": {
    ...,
    "aws.greengrass.clientdevices.IPDetector": {
      "componentVersion": "2.1.1",
      "configurationUpdate": {
        "merge": "{\"defaultPort\":\"8883\"}"
      }
    }
  }
}
```

Gérer manuellement les points de terminaison

Vous pouvez gérer manuellement les points de terminaison du broker MQTT pour les appareils principaux.

Chaque point de terminaison du broker MQTT possède les informations suivantes :

Point de terminaison (HostAddress)

Adresse IP ou adresse DNS à laquelle les appareils clients peuvent se connecter à un courtier MQTT sur le périphérique principal.

Port (PortNumber)

Port sur lequel le broker MQTT fonctionne sur le périphérique principal.

Vous pouvez configurer ce port sur le [composant du broker Moquette MQTT](#), qui utilise par défaut le port 8883.

Métadonnées (Metadata)

Métadonnées supplémentaires à fournir aux appareils clients qui se connectent à ce point de terminaison.

Rubriques

- [Gérer les points de terminaison \(console\)](#)
- [Gérer les points de terminaison \(\)AWS CLI](#)
- [Gérer les points de terminaison \(API\)](#)

Gérer les points de terminaison (console)

Vous pouvez utiliser la AWS IoT Greengrass console pour afficher, mettre à jour et supprimer les points de terminaison d'un appareil principal.

Pour gérer les points de terminaison d'un appareil principal (console)

1. Accédez à la [console AWS IoT Greengrass](#).
2. Choisissez les appareils Core.
3. Choisissez l'appareil principal à gérer.
4. Sur la page de détails de l'appareil principal, choisissez l'onglet Appareils clients.
5. Dans la section Points de terminaison du broker MQTT, vous pouvez voir les points de terminaison du broker MQTT du périphérique principal. Choisissez Gérer les points de terminaison.
6. Dans le mode Gérer les points de terminaison, ajoutez ou supprimez des points de terminaison du broker MQTT pour le périphérique principal.
7. Choisissez Mettre à jour.

Gérer les points de terminaison (AWS CLI)

Vous pouvez utiliser le AWS Command Line Interface (AWS CLI) pour gérer les points de terminaison d'un appareil principal.

Note

La prise en charge des appareils clients AWS IoT Greengrass V2 étant rétrocompatible avec AWS IoT Greengrass V1, vous pouvez utiliser les opérations AWS IoT Greengrass V2 d'AWS IoT Greengrass V1 API pour gérer les principaux points de terminaison des appareils.

Pour obtenir des points de terminaison pour un appareil principal (AWS CLI)

- Utilisez l'une des commandes suivantes :
 - [Greengrass V2 : get-connectivity-info](#)
 - [herbe verte : get-connectivity-info](#)

Pour mettre à jour les points de terminaison d'un appareil principal (AWS CLI)

- Utilisez l'une des commandes suivantes :
 - [Greengrass V2 : update-connectivity-info](#)
 - [herbe verte : update-connectivity-info](#)

Gérer les points de terminaison (API)

Vous pouvez utiliser l'AWS API pour gérer les points de terminaison d'un appareil principal.

Note

La prise en charge des appareils clients AWS IoT Greengrass V2 étant rétrocompatible avec AWS IoT Greengrass V1, vous pouvez utiliser les opérations AWS IoT Greengrass V2 d'AWS IoT Greengrass V1 API pour gérer les principaux points de terminaison des appareils.

Pour obtenir des points de terminaison pour un appareil principal (AWS API)

- Utilisez l'une des opérations suivantes :

- [V2 : GetConnectivityInfo](#)
- [V1 : GetConnectivityInfo](#)

Pour mettre à jour les points de terminaison d'un appareil principal (AWS API)

- Utilisez l'une des opérations suivantes :
 - [V2 : UpdateConnectivityInfo](#)
 - [V1 : UpdateConnectivityInfo](#)

Choisissez un courtier MQTT

AWS IoT Greengrass fournit des options vous permettant de choisir le broker MQTT local à exécuter sur vos appareils principaux. Les appareils clients se connectent au courtier MQTT qui s'exécute sur un périphérique principal. Choisissez donc un courtier MQTT compatible avec les appareils clients que vous souhaitez connecter.

Note

Nous vous recommandons de ne déployer qu'un seul composant de broker MQTT. Le [pont MQTT](#) et les composants du [détecteur IP](#) fonctionnent avec un seul composant de courtier MQTT à la fois. Si vous déployez plusieurs composants du broker MQTT, vous devez les configurer pour utiliser différents ports.

Vous pouvez choisir parmi les courtiers MQTT suivants :

- [Courtier MQTT 3.1.1 \(Moquette\)](#) — `aws.greengrass.clientdevices.mqtt.Moquette`

Choisissez cette option pour un broker MQTT léger conforme à la norme MQTT 3.1.1. Le courtier AWS IoT Core MQTT est également conforme à la norme MQTT 3.1.1. Vous pouvez donc utiliser ces fonctionnalités pour créer une application qui utilise MQTT 3.1.1 sur vos appareils et le Kit SDK des appareils AWS IoT AWS Cloud

- [Courtier MQTT 5 \(EMQX\)](#) — `aws.greengrass.clientdevices.mqtt.EMQX`

Choisissez cette option pour utiliser les fonctionnalités de MQTT 5 dans la communication entre les appareils principaux et les appareils clients. Ce composant utilise plus de ressources que le broker Moquette MQTT 3.1.1, et sur les appareils principaux de Linux, il nécessite Docker.

MQTT 5 est rétrocompatible avec MQTT 3.1.1, vous pouvez donc connecter des appareils clients utilisant MQTT 3.1.1 à ce broker. Si vous exécutez le broker Moquette MQTT 3.1.1, vous pouvez le remplacer par le broker EMQX MQTT 5, et les appareils clients pourront continuer à se connecter et à fonctionner normalement.

- Implémenter un courtier personnalisé

Choisissez cette option pour créer un composant de courtier local personnalisé pour communiquer avec les appareils clients. Vous pouvez créer un courtier local personnalisé qui utilise un protocole autre que MQTT. AWS IoT Greengrass fournit un SDK de composants que vous pouvez utiliser pour authentifier et autoriser les appareils clients. Pour plus d'informations, consultez [Utilisez le Kit SDK des appareils AWS IoT pour communiquer avec le noyau de Greengrass, les autres composants et AWS IoT Core](#) et [Authentifier et autoriser les appareils clients](#).

Connexion des appareils clients à un périphérique AWS IoT Greengrass Core avec un courtier MQTT

Lorsque vous utilisez un courtier MQTT sur votre appareil AWS IoT Greengrass principal, celui-ci utilise une autorité de certification (CA) principale propre à l'appareil pour délivrer un certificat au courtier afin d'établir des connexions TLS mutuelles avec les clients.

AWS IoT Greengrass générera automatiquement une autorité de certification du périphérique principal, ou vous pouvez fournir la vôtre. Le périphérique principal auprès duquel l'autorité de certification est enregistrée AWS IoT Greengrass lorsque le [Authentification de l'appareil client](#) composant est connecté. L'autorité de certification du périphérique principal générée automatiquement est persistante, le périphérique continuera à utiliser la même autorité de certification tant que le composant d'authentification du périphérique client est configuré.

Lorsque le broker MQTT démarre, il demande un certificat. Le composant d'authentification du périphérique client émet un certificat X.509 à l'aide de l'autorité de certification du périphérique principal. Le certificat fait l'objet d'une rotation lorsque le broker démarre, lorsqu'il expire ou lorsque les informations de connectivité telles que l'adresse IP changent. Pour de plus amples informations, veuillez consulter [Rotation des certificats sur le broker MQTT local](#).

Pour connecter un client au broker MQTT, vous avez besoin des éléments suivants :

- L'appareil client doit disposer de l'autorité de certification du périphérique AWS IoT Greengrass Core. Vous pouvez obtenir cette autorité de certification par le biais de la découverte du cloud ou en la fournissant manuellement. Pour de plus amples informations, veuillez consulter [Utiliser votre propre autorité de certification](#).
- Le nom de domaine complet (FQDN) ou l'adresse IP du périphérique principal doivent figurer dans le certificat de courtier émis par l'autorité de certification du périphérique principal. Vous vous en assurez à l'aide du [Détecteur IP](#) composant ou en configurant manuellement l'adresse IP. Pour de plus amples informations, veuillez consulter [Gérez les principaux points de terminaison des appareils](#).
- Le composant d'authentification du périphérique client doit autoriser l'appareil client à se connecter au périphérique principal de Greengrass. Pour de plus amples informations, veuillez consulter [Authentification de l'appareil client](#).

Utiliser votre propre autorité de certification

Si vos appareils clients ne peuvent pas accéder au cloud pour découvrir votre appareil principal, vous pouvez fournir une autorité de certification (CA) principale. Votre appareil principal Greengrass utilise l'autorité de certification du périphérique principal pour émettre des certificats pour votre courtier MQTT. Une fois que vous avez configuré le périphérique principal et que vous avez doté votre appareil client de son autorité de certification, vos appareils clients peuvent se connecter au point de terminaison et vérifier la prise de contact TLS à l'aide de l'autorité de certification du périphérique principal (propre autorité de certification fournie ou générée automatiquement).

Pour configurer le [Authentification de l'appareil client](#) composant afin qu'il utilise l'autorité de certification de votre appareil principal, définissez le paramètre de `certificateAuthority` configuration lorsque vous déployez le composant. Vous devez fournir les informations suivantes lors de la configuration :

- Emplacement du certificat CA d'un périphérique principal.
- La clé privée du certificat CA du périphérique principal.
- (Facultatif) La chaîne de certificats vers le certificat racine si l'autorité de certification du périphérique principal est une autorité de certification intermédiaire.

Si vous fournissez une autorité de certification de périphérique principale, AWS IoT Greengrass enregistre l'autorité de certification dans le cloud.

Vous pouvez stocker vos certificats dans un module de sécurité matériel ou dans le système de fichiers. L'exemple suivant montre une `certificateAuthority` configuration pour une autorité de certification intermédiaire stockée à l'aide de HSM/TPM. Notez que la chaîne de certificats ne peut être stockée que sur disque.

```
"certificateAuthority": {  
  "certificateUri": "pkcs11:object=CustomerIntermediateCA;type=cert",  
  "privateKeyUri": "pkcs11:object=CustomerIntermediateCA;type=private"  
  "certificateChainUri": "file:///home/ec2-user/creds/certificateChain.pem",  
}
```

Dans cet exemple, le paramètre `certificateAuthority` de configuration configure le composant d'authentification de l'appareil client pour utiliser une autorité de certification intermédiaire du système de fichiers :

```
"certificateAuthority": {  
  "certificateUri": "file:///home/ec2-user/creds/intermediateCA.pem",  
  "privateKeyUri": "file:///home/ec2-user/creds/intermediateCA.privateKey.pem",  
  "certificateChainUri": "file:///home/ec2-user/creds/certificateChain.pem",  
}
```

Pour connecter les appareils à votre appareil AWS IoT Greengrass Core, procédez comme suit :

1. Créez une autorité de certification (CA) intermédiaire pour le périphérique principal de Greengrass à l'aide de l'autorité de certification racine de votre organisation. Nous vous recommandons d'utiliser une autorité de certification intermédiaire comme meilleure pratique en matière de sécurité.
2. Fournissez le certificat de l'autorité de certification intermédiaire, la clé privée et la chaîne de certificats de votre autorité de certification racine au périphérique principal Greengrass. Pour de plus amples informations, veuillez consulter [Authentification de l'appareil client](#). L'autorité de certification intermédiaire devient l'autorité de certification principale du périphérique principal de Greengrass, et le périphérique enregistre l'autorité de certification auprès de celle-ci. AWS IoT Greengrass
3. Enregistrez l'appareil client en tant qu'AWS IoT objet. Pour plus d'informations, consultez la section [Création d'un objet](#) dans le Guide du AWS IoT Core développeur. Ajoutez la clé privée, la clé publique, le certificat de périphérique et le certificat d'autorité de certification racine à votre appareil client. La manière dont vous ajoutez les informations dépend de votre appareil et de votre logiciel.

Une fois que vous avez configuré votre appareil, vous pouvez utiliser le certificat et la chaîne de clés publiques pour vous connecter au périphérique principal de Greengrass. Votre logiciel est chargé de trouver les principaux points de terminaison de l'appareil. Vous pouvez définir le point de terminaison manuellement pour le périphérique principal. Pour de plus amples informations, veuillez consulter [Gérer manuellement les points de terminaison](#).

Tester les communications entre appareils clients

Les appareils clients peuvent utiliser le Kit SDK des appareils AWS IoT pour découvrir, se connecter et communiquer avec un appareil principal. Vous pouvez utiliser le client de découverte Greengrass Kit SDK des appareils AWS IoT pour utiliser l'[API de découverte Greengrass](#), qui renvoie des informations sur les principaux appareils auxquels un appareil client peut se connecter. La réponse de l'API inclut les points de terminaison du broker MQTT à connecter et les certificats à utiliser pour vérifier l'identité de chaque périphérique principal. Ensuite, l'appareil client peut essayer chaque point de terminaison jusqu'à ce qu'il se connecte avec succès à un périphérique principal.

Les appareils clients ne peuvent découvrir que les appareils principaux auxquels vous les associez. Avant de tester les communications entre un appareil client et un périphérique principal, vous devez associer l'appareil client au périphérique principal. Pour de plus amples informations, veuillez consulter [Associer des appareils clients](#).

L'API de découverte Greengrass renvoie les points de terminaison du broker MQTT du périphérique principal que vous spécifiez. Vous pouvez utiliser le [composant de détection IP](#) pour gérer ces points de terminaison à votre place, ou vous pouvez les gérer manuellement pour chaque périphérique principal. Pour de plus amples informations, veuillez consulter [Gérez les principaux points de terminaison des appareils](#).

Note

Pour utiliser l'API de découverte Greengrass, un appareil client doit disposer de cette autorisation. `greengrass:Discover` Pour de plus amples informations, veuillez consulter [AWS IoT Politique minimale pour les appareils clients](#).

Kit SDK des appareils AWS IoT Il est disponible dans plusieurs langages de programmation. Pour plus d'informations, consultez la section [AWS IoT Appareil SDKs](#) dans le guide du AWS IoT Core développeur.

Rubriques

- [Tester les communications \(Python\)](#)
- [Tester les communications \(C++\)](#)
- [Tester les communications \(JavaScript\)](#)
- [Tester les communications \(Java\)](#)

Tester les communications (Python)

Dans cette section, vous allez utiliser l'exemple de découverte de Greengrass dans la [Kit SDK des appareils AWS IoT version v2 pour Python](#) afin de tester les communications entre un appareil client et un périphérique principal.

Important

Pour utiliser la Kit SDK des appareils AWS IoT version v2 pour Python, un appareil doit exécuter Python 3.6 ou une version ultérieure.

Pour tester les communications (Kit SDK des appareils AWS IoT v2 pour Python)

1. Téléchargez et installez la [Kit SDK des appareils AWS IoT version 2 pour Python](#) sur l' AWS IoT appareil à connecter en tant que périphérique client.

Sur l'appareil client, effectuez les opérations suivantes :

- a. Clonez le dépôt Kit SDK des appareils AWS IoT v2 pour Python pour le télécharger.

```
git clone https://github.com/aws/aws-iot-device-sdk-python-v2.git
```

- b. Installez la Kit SDK des appareils AWS IoT version 2 pour Python.

```
python3 -m pip install --user ./aws-iot-device-sdk-python-v2
```

2. Accédez au dossier d'échantillons dans la Kit SDK des appareils AWS IoT version 2 pour Python.

```
cd aws-iot-device-sdk-python-v2/samples/greengrass
```

3. Exécutez l'exemple d'application de découverte Greengrass. Cette application attend des arguments qui spécifient le nom de l'objet du périphérique client, le sujet MQTT et le message

à utiliser, ainsi que les certificats qui authentifient et sécurisent la connexion. L'exemple suivant envoie un message Hello World à la `clients/MyClientDevice1/hello/world` rubrique.

- *MyClientDevice1* Remplacez-le par le nom de l'objet de l'appareil client.
- *~/certs/AmazonRootCA1.pem* Remplacez-le par le chemin d'accès au certificat racine de l'autorité de certification Amazon sur l'appareil client.
- Remplacez *~/certs/device.pem.crt* par le chemin d'accès au certificat de l'appareil sur l'appareil client.
- Remplacez *~/certs/private.pem.key* par le chemin d'accès au fichier de clé privée sur l'appareil client.
- Remplacez *us-east-1* par la AWS région dans laquelle fonctionnent votre appareil client et votre appareil principal.

```
python3 basic_discovery.py \\  
--thing_name MyClientDevice1 \\  
--topic 'clients/MyClientDevice1/hello/world' \\  
--message 'Hello World!' \\  
--ca_file ~/certs/AmazonRootCA1.pem \\  
--cert ~/certs/device.pem.crt \\  
--key ~/certs/private.pem.key \\  
--region us-east-1 \\  
--verbosity Warn
```

L'exemple d'application de découverte envoie le message 10 fois et se déconnecte. Il s'abonne également au même sujet où il publie des messages. Si le résultat indique que l'application a reçu des messages MQTT sur le sujet, le dispositif client peut communiquer avec succès avec le dispositif principal.

```
Performing greengrass discovery...
awsiot.greengrass_discovery.DiscoverResponse(gg_groups=[awsiot.greengrass_discovery.GGGroup
coreDevice-MyGreengrassCore',
  cores=[awsiot.greengrass_discovery.GGCore(thing_arn='arn:aws:iot:us-
east-1:123456789012:thing/MyGreengrassCore',
  connectivity=[awsiot.greengrass_discovery.ConnectivityInfo(id='203.0.113.0',
  host_address='203.0.113.0', metadata='', port=8883)]),
  certificate_authorities=['-----BEGIN CERTIFICATE-----\
MIICiT...EXAMPLE=\
-----END CERTIFICATE-----\
```

```
']]])
Trying core arn:aws:iot:us-east-1:123456789012:thing/MyGreengrassCore at host
203.0.113.0 port 8883
Connected!
Published topic clients/MyClientDevice1/hello/world: {"message": "Hello World!",
"sequence": 0}

Publish received on topic clients/MyClientDevice1/hello/world
b'{"message": "Hello World!", "sequence": 0}'
Published topic clients/MyClientDevice1/hello/world: {"message": "Hello World!",
"sequence": 1}

Publish received on topic clients/MyClientDevice1/hello/world
b'{"message": "Hello World!", "sequence": 1}'

...

Published topic clients/MyClientDevice1/hello/world: {"message": "Hello World!",
"sequence": 9}

Publish received on topic clients/MyClientDevice1/hello/world
b'{"message": "Hello World!", "sequence": 9}'
```

Si l'application génère plutôt une erreur, consultez la section [Résolution des problèmes de découverte de Greengrass](#).

Vous pouvez également consulter les journaux Greengrass sur l'appareil principal pour vérifier si le périphérique client se connecte et envoie des messages avec succès. Pour de plus amples informations, veuillez consulter [AWS IoT Greengrass Journaux de surveillance](#).

Tester les communications (C++)

Dans cette section, vous allez utiliser l'exemple de découverte de Greengrass dans la [Kit SDK des appareils AWS IoT version v2 pour C++ pour](#) tester les communications entre un appareil client et un périphérique principal.

Pour créer la Kit SDK des appareils AWS IoT version 2 pour C++, un périphérique doit disposer des outils suivants :

- C++ 11 ou version ultérieure
- CMake 3.1 ou version ultérieure

- L'un des compilateurs suivants :
 - GCC 4.8 ou version ultérieure
 - Clang 3.9 ou version ultérieure
 - MSVC 2015 ou version ultérieure

Pour tester les communications (Kit SDK des appareils AWS IoT v2 pour C++)

1. Téléchargez et compilez la [Kit SDK des appareils AWS IoT v2 pour C++](#) sur l' AWS IoT appareil à connecter en tant que périphérique client.

Sur l'appareil client, effectuez les opérations suivantes :

- a. Créez un dossier pour l'espace de travail Kit SDK des appareils AWS IoT v2 pour C++ et modifiez-le.

```
cd
mkdir iot-device-sdk-cpp
cd iot-device-sdk-cpp
```

- b. Clonez le dépôt Kit SDK des appareils AWS IoT v2 pour C++ pour le télécharger. L'`--recursive` indique de télécharger les sous-modules.

```
git clone --recursive https://github.com/aws/aws-iot-device-sdk-cpp-v2.git
```

- c. Créez un dossier pour la sortie de compilation Kit SDK des appareils AWS IoT v2 pour C++ et modifiez-le.

```
mkdir aws-iot-device-sdk-cpp-v2-build
cd aws-iot-device-sdk-cpp-v2-build
```

- d. Compilez la Kit SDK des appareils AWS IoT v2 pour C++.

```
cmake -DCMAKE_INSTALL_PREFIX=~/.iot-device-sdk-cpp" -
DCMAKE_BUILD_TYPE="Release" ../aws-iot-device-sdk-cpp-v2
cmake --build . --target install
```

2. Créez l'exemple d'application Greengrass Discovery dans la Kit SDK des appareils AWS IoT version 2 pour C++. Procédez comme suit :

- a. Accédez au dossier d'échantillons Greengrass Discovery dans la Kit SDK des appareils AWS IoT version 2 pour C++.

```
cd ../aws-iot-device-sdk-cpp-v2/samples/greengrass/basic_discovery
```

- b. Créez un dossier pour l'exemple de sortie de build de Greengrass Discovery et modifiez-le.

```
mkdir build  
cd build
```

- c. Créez l'exemple d'application Greengrass Discovery.

```
cmake -DCMAKE_PREFIX_PATH=~/.iot-device-sdk-cpp" -  
DCMAKE_BUILD_TYPE="Release" ..  
cmake --build . --config "Release"
```

3. Exécutez l'exemple d'application de découverte Greengrass. Cette application attend des arguments qui spécifient le nom de l'objet du périphérique client, le sujet MQTT à utiliser et les certificats qui authentifient et sécurisent la connexion. L'exemple suivant s'abonne à la `clients/MyClientDevice1/hello/world` rubrique et publie un message que vous entrez sur la ligne de commande pour la même rubrique.

- `MyClientDevice1` Remplacez-le par le nom de l'objet de l'appareil client.
- `~/certs/AmazonRootCA1.pem` Remplacez-le par le chemin d'accès au certificat racine de l'autorité de certification Amazon sur l'appareil client.
- Remplacez `~/certs/device.pem.crt` par le chemin d'accès au certificat de l'appareil sur l'appareil client.
- Remplacez `~/certs/private.pem.key` par le chemin d'accès au fichier de clé privée sur l'appareil client.
- Remplacez `us-east-1` par la AWS région dans laquelle fonctionnent votre appareil client et votre appareil principal.

```
./basic-discovery \  
--thing_name MyClientDevice1 \  
--topic 'clients/MyClientDevice1/hello/world' \  
--ca_file ~/certs/AmazonRootCA1.pem \  
--cert ~/certs/device.pem.crt \  

```

```
--key ~/certs/private.pem.key \  
--region us-east-1
```

L'exemple d'application Discovery s'abonne au sujet et vous invite à saisir un message à publier.

```
Connecting to group greengrassV2-coreDevice-MyGreengrassCore with thing arn  
arn:aws:iot:us-east-1:123456789012:thing/MyGreengrassCore, using endpoint  
203.0.113.0:8883  
Connected to group greengrassV2-coreDevice-MyGreengrassCore, using connection to  
203.0.113.0:8883  
Successfully subscribed to clients/MyClientDevice1/hello/world  
Enter the message you want to publish to topic clients/MyClientDevice1/hello/world  
and press enter. Enter 'exit' to exit this program.
```

Si l'application génère plutôt une erreur, consultez la section [Résolution des problèmes de découverte de Greengrass](#).

4. Entrez un message, tel que **Hello World!**.

```
Enter the message you want to publish to topic clients/MyClientDevice1/hello/world  
and press enter. Enter 'exit' to exit this program.  
Hello World!
```

Si le résultat indique que l'application a reçu le message MQTT sur le sujet, le dispositif client peut communiquer avec succès avec le dispositif principal.

```
Operation on packetId 2 Succeeded  
Publish received on topic clients/MyClientDevice1/hello/world  
Message:  
Hello World!
```

Vous pouvez également consulter les journaux Greengrass sur l'appareil principal pour vérifier si le périphérique client se connecte et envoie des messages avec succès. Pour de plus amples informations, veuillez consulter [AWS IoT Greengrass Journaux de surveillance](#).

Tester les communications (JavaScript)

Dans cette section, vous allez utiliser l'exemple de découverte de Greengrass dans la [Kit SDK des appareils AWS IoT version v2 JavaScript pour](#) tester les communications entre un appareil client et un appareil principal.

⚠ Important

Pour utiliser la Kit SDK des appareils AWS IoT version v2 pour JavaScript, un appareil doit exécuter Node v10.0 ou version ultérieure.

Pour tester les communications (Kit SDK des appareils AWS IoT version 2 pour JavaScript)

1. Téléchargez et installez la [Kit SDK des appareils AWS IoT version 2 JavaScript pour](#) AWS IoT que l'appareil se connecte en tant que périphérique client.

Sur l'appareil client, effectuez les opérations suivantes :

- a. Clonez la Kit SDK des appareils AWS IoT v2 pour JavaScript le dépôt afin de la télécharger.

```
git clone https://github.com/aws/aws-iot-device-sdk-js-v2.git
```

- b. Installez la Kit SDK des appareils AWS IoT v2 pour JavaScript.

```
cd aws-iot-device-sdk-js-v2
npm install
```

2. Accédez au dossier d'exemples de découverte Greengrass dans la Kit SDK des appareils AWS IoT version v2 pour JavaScript

```
cd samples/node/greengrass/basic_discovery
```

3. Installez l'exemple d'application Greengrass Discovery.

```
npm install
```

4. Exécutez l'exemple d'application de découverte Greengrass. Cette application attend des arguments qui spécifient le nom de l'objet du périphérique client, le sujet MQTT et le message à utiliser, ainsi que les certificats qui authentifient et sécurisent la connexion. L'exemple suivant envoie un message Hello World à la `clients/MyClientDevice1/hello/world` rubrique.

- `MyClientDevice1` Remplacez-le par le nom de l'objet de l'appareil client.
- `~/certs/AmazonRootCA1.pem` Remplacez-le par le chemin d'accès au certificat racine de l'autorité de certification Amazon sur l'appareil client.

- Remplacez `~/certs/device.pem.crt` par le chemin d'accès au certificat de l'appareil sur l'appareil client.
- Remplacez `~/certs/private.pem.key` par le chemin d'accès au fichier de clé privée sur l'appareil client.
- Remplacez `us-east-1` par la AWS région dans laquelle fonctionnent votre appareil client et votre appareil principal.

```
node dist/index.js \
  --thing_name MyClientDevice1 \
  --topic 'clients/MyClientDevice1/hello/world' \
  --message 'Hello World!' \
  --ca_file ~/certs/AmazonRootCA1.pem \
  --cert ~/certs/device.pem.crt \
  --key ~/certs/private.pem.key \
  --region us-east-1 \
  --verbose warn
```

L'exemple d'application de découverte envoie le message 10 fois et se déconnecte. Il s'abonne également au même sujet où il publie des messages. Si le résultat indique que l'application a reçu des messages MQTT sur le sujet, le dispositif client peut communiquer avec succès avec le dispositif principal.

```
Discovery Response:
{"gg_groups":[{"gg_group_id":"greengrassV2-coreDevice-MyGreengrassCore","cores":[{"thing_arn":"arn:aws:iot:us-east-1:123456789012:thing/MyGreengrassCore","connectivity":[{"id":"203.0.113.0","host_address":"203.0.113.0","port":8883,"metadata":""}]}],"certificate":["-----BEGIN CERTIFICATE-----\nMIICiT...EXAMPLE=\n-----END CERTIFICATE-----\n"]}]}
Trying
endpoint={"id":"203.0.113.0","host_address":"203.0.113.0","port":8883,"metadata":""}
[WARN] [2021-06-12T00:46:45Z] [00007f90c0e8d700] [socket] - id=0x7f90b8018710
fd=26: setsockopt() for NO_SIGNAL failed with errno 92. If you are having SIGPIPE
signals thrown, you may want to install a signal trap in your application layer.
Connected to
endpoint={"id":"203.0.113.0","host_address":"203.0.113.0","port":8883,"metadata":""}
Publish received. topic:"clients/MyClientDevice1/hello/world" dup:false qos:0
retain:false
{"message":"Hello World!","sequence":1}
```

```
Publish received. topic:"clients/MyClientDevice1/hello/world" dup:false qos:0
retain:false
{"message":"Hello World!","sequence":2}
Publish received. topic:"clients/MyClientDevice1/hello/world" dup:false qos:0
retain:false
{"message":"Hello World!","sequence":3}
Publish received. topic:"clients/MyClientDevice1/hello/world" dup:false qos:0
retain:false
{"message":"Hello World!","sequence":4}
Publish received. topic:"clients/MyClientDevice1/hello/world" dup:false qos:0
retain:false
{"message":"Hello World!","sequence":5}
Publish received. topic:"clients/MyClientDevice1/hello/world" dup:false qos:0
retain:false
{"message":"Hello World!","sequence":6}
Publish received. topic:"clients/MyClientDevice1/hello/world" dup:false qos:0
retain:false
{"message":"Hello World!","sequence":7}
Publish received. topic:"clients/MyClientDevice1/hello/world" dup:false qos:0
retain:false
{"message":"Hello World!","sequence":8}
Publish received. topic:"clients/MyClientDevice1/hello/world" dup:false qos:0
retain:false
{"message":"Hello World!","sequence":9}
Publish received. topic:"clients/MyClientDevice1/hello/world" dup:false qos:0
retain:false
{"message":"Hello World!","sequence":10}
Complete!
```

Si l'application génère plutôt une erreur, consultez la section [Résolution des problèmes de découverte de Greengrass](#).

Vous pouvez également consulter les journaux Greengrass sur l'appareil principal pour vérifier si le périphérique client se connecte et envoie des messages avec succès. Pour de plus amples informations, veuillez consulter [AWS IoT Greengrass Journaux de surveillance](#).

Tester les communications (Java)

Dans cette section, vous allez utiliser l'exemple de découverte de Greengrass dans la [Kit SDK des appareils AWS IoT version v2 pour Java pour](#) tester les communications entre un appareil client et un appareil principal.

⚠ Important

Pour créer la Kit SDK des appareils AWS IoT v2 pour Java, un appareil doit disposer des outils suivants :

- Java 8 ou version ultérieure, en JAVA_HOME pointant vers le dossier Java.
- Apache Maven

Pour tester les communications (Kit SDK des appareils AWS IoT v2 pour Java)

1. Téléchargez et compilez la [Kit SDK des appareils AWS IoT v2 pour Java](#) sur l' AWS IoT appareil à connecter en tant que périphérique client.

Sur l'appareil client, effectuez les opérations suivantes :

- a. Clonez le dépôt Kit SDK des appareils AWS IoT v2 pour Java pour le télécharger.

```
git clone https://github.com/aws/aws-iot-device-sdk-java-v2.git
```

- b. Accédez au dossier Kit SDK des appareils AWS IoT v2 pour Java.
- c. Compilez la Kit SDK des appareils AWS IoT v2 pour Java.

```
cd aws-iot-device-sdk-java-v2
mvn versions:use-latest-versions -Dincludes="software.amazon.awssdk.crt*"
mvn clean install
```

2. Exécutez l'exemple d'application de découverte Greengrass. Cette application attend des arguments qui spécifient le nom de l'objet du périphérique client, le sujet MQTT à utiliser et les certificats qui authentifient et sécurisent la connexion. L'exemple suivant s'abonne à la `clients/MyClientDevice1/hello/world` rubrique et publie un message que vous entrez sur la ligne de commande pour la même rubrique.
 - Remplacez les deux instances de `MyClientDevice1` par le nom de l'objet de l'appareil client.
 - `$HOME/certs/AmazonRootCA1.pem` Remplacez-le par le chemin d'accès au certificat racine de l'autorité de certification Amazon sur l'appareil client.
 - Remplacez `$HOME/certs/device.pem.crt` par le chemin d'accès au certificat de l'appareil sur l'appareil client.

- Remplacez `$HOME/certs/private.pem.key` par le chemin d'accès au fichier de clé privée sur l'appareil client.
- `us-east-1` Remplacez-le par la Région AWS endroit où fonctionnent votre appareil client et votre appareil principal.

```
DISCOVERY_SAMPLE_ARGS="--thing_name MyClientDevice1 \  
  --topic 'clients/MyClientDevice1/hello/world' \  
  --ca_file $HOME/certs/AmazonRootCA1.pem \  
  --cert $HOME/certs/device.pem.crt \  
  --key $HOME/certs/private.pem.key \  
  --region us-east-1"  
  
mvn exec:java -pl samples/Greengrass/Discovery \  
  -Dexec.mainClass=greengrass.BasicDiscovery \  
  -Dexec.args="$DISCOVERY_SAMPLE_ARGS"
```

L'exemple d'application Discovery s'abonne au sujet et vous invite à saisir un message à publier.

```
Connecting to group ID greengrassV2-coreDevice-MyGreengrassCore, with thing  
arn arn:aws:iot:us-east-1:123456789012:thing/MyGreengrassCore, using endpoint  
203.0.113.0:8883  
Started a clean session  
Enter the message you want to publish to topic clients/MyClientDevice1/hello/world  
and press Enter. Type 'exit' or 'quit' to exit this program:
```

Si l'application génère plutôt une erreur, consultez la section [Résolution des problèmes de découverte de Greengrass](#).

3. Entrez un message, tel que **Hello World!**.

```
Enter the message you want to publish to topic clients/MyClientDevice1/hello/world  
and press Enter. Type 'exit' or 'quit' to exit this program:  
Hello World!
```

Si le résultat indique que l'application a reçu le message MQTT sur le sujet, le dispositif client peut communiquer avec succès avec le dispositif principal.

```
Message received on topic clients/MyClientDevice1/hello/world: Hello World!
```

Vous pouvez également consulter les journaux Greengrass sur l'appareil principal pour vérifier si le périphérique client se connecte et envoie des messages avec succès. Pour de plus amples informations, veuillez consulter [AWS IoT Greengrass Journaux de surveillance](#).

API de découverte Greengrass RESTful

AWS IoT Greengrass fournit l'opération *Discover* d'API que les appareils clients peuvent utiliser pour identifier les appareils principaux de Greengrass auxquels ils peuvent se connecter. Les appareils clients utilisent cette opération de plan de données pour récupérer les informations requises pour se connecter aux appareils principaux de Greengrass et les associer à l'opération d'[BatchAssociateClientDeviceWithCoreDevice](#) API. Lorsqu'un appareil client est en ligne, il peut se connecter au service AWS IoT Greengrass cloud et utiliser l'API de découverte pour trouver :

- L'adresse IP et le port de chaque appareil principal Greengrass associé.
- Le certificat CA du périphérique principal, que les appareils clients peuvent utiliser pour authentifier le périphérique principal Greengrass.

Note

Les appareils clients peuvent également utiliser le client de découverte dans le Kit SDK des appareils AWS IoT pour découvrir les informations de connectivité des appareils principaux de Greengrass. Le client de découverte utilise l'API de découverte. Pour plus d'informations, consultez les ressources suivantes :

- [Tester les communications entre appareils clients](#)
- [RESTful API Greengrass Discovery](#) dans le guide du AWS IoT Greengrass Version 1 développeur.

Pour utiliser cette opération d'API, envoyez des requêtes HTTP à l'API de découverte sur le point de terminaison du plan de données Greengrass. Le format de ce point de terminaison d'API est le suivant.

```
https://greengrass-ats.iot.region.amazonaws.com:port/greengrass/discover/thing/thing-name
```

Pour obtenir la liste des points de terminaison Régions AWS et des points de terminaison pris en charge par l'API de AWS IoT Greengrass découverte, consultez la section [AWS IoT Greengrass V2 Points de terminaison et quotas](#) dans le. Références générales AWS Cette opération d'API n'est disponible que sur le point de terminaison du plan de données Greengrass. Le point de terminaison du plan de contrôle que vous utilisez pour gérer les composants et les déploiements est différent du point de terminaison du plan de données.

Note

L'API de découverte est la même pour AWS IoT Greengrass V1 et AWS IoT Greengrass V2. Si vos appareils clients se connectent à un AWS IoT Greengrass V1 cœur, vous pouvez les connecter aux appareils AWS IoT Greengrass V2 principaux sans modifier le code sur les appareils clients. Pour plus d'informations, consultez l' [RESTful API Greengrass Discovery](#) dans le guide du AWS IoT Greengrass Version 1 développeur.

Rubriques

- [Authentification et autorisation de découverte](#)
- [Demande](#)
- [Réponse](#)
- [Testez l'API de découverte avec cURL](#)

Authentification et autorisation de découverte

Pour utiliser l'API de découverte afin de récupérer des informations de connectivité, un appareil client doit utiliser l'authentification mutuelle TLS avec un certificat client X.509 pour s'authentifier. Pour plus d'informations, consultez la section [Certificats client X.509](#) dans le Guide du AWS IoT Core développeur.

Un appareil client doit également être autorisé à effectuer l'`greengrass:Discover`. L'exemple de AWS IoT politique suivant permet à un AWS IoT objet nommé `MyClientDevice1Discover` de fonctionner par lui-même.

JSON

```
{  
  "Version": "2012-10-17",  
}
```

```
"Statement": [  
  {  
    "Effect": "Allow",  
    "Action": "greengrass:Discover",  
    "Resource": [  
      "arn:aws:iot:us-west-2:123456789012:thing/MyClientDevice1"  
    ]  
  }  
]
```

Important

Les variables de politique des objets (`iot:Connection.Thing.*`) ne sont pas prises en charge dans AWS IoT les politiques relatives aux appareils principaux ou aux opérations du plan de données Greengrass. Vous pouvez plutôt utiliser un caractère générique correspondant à plusieurs appareils portant des noms similaires. Par exemple, vous pouvez spécifier `MyGreengrassDevice*` de correspondre `MyGreengrassDevice1` `MyGreengrassDevice2`, et ainsi de suite.

Pour plus d'informations, consultez [AWS IoT Core les politiques](#) dans le guide du AWS IoT Core développeur.

Demande

La demande contient les en-têtes HTTP standard et est envoyée au point de terminaison Greengrass Discovery, comme indiqué dans les exemples suivants.

Le numéro de port varie selon que le périphérique principal est configuré pour envoyer du trafic HTTPS via le port 8443 ou le port 443. Pour de plus amples informations, veuillez consulter [the section called "Connexion au port 443 ou via un proxy réseau"](#).

Note

Ces exemples utilisent le point de terminaison Amazon Trust Services (ATS), qui fonctionne avec les certificats CA racine ATS recommandés. Les points de terminaison doivent correspondre au type de certificat CA racine.

Port 8443

```
HTTP GET https://greengrass-ats.iot.region.amazonaws.com:8443/greengrass/discover/thing/thing-name
```

Port 443

```
HTTP GET https://greengrass-ats.iot.region.amazonaws.com:443/greengrass/discover/thing/thing-name
```

Note

Les clients qui se connectent sur le port 443 doivent implémenter l'extension TLS [ALPN \(Application Layer Protocol Negotiation\)](#) et la transmettre x-amzn-http-ca comme ProtocolName dans le. ProtocolNameList Pour plus d'informations, consultez la section [Protocoles](#) du guide du AWS IoT développeur.

Réponse

En cas de succès, l'en-tête de réponse inclut le code d'état HTTP 200 et le corps de la réponse contient le document de réponse Discover.

Note

Parce qu'elle AWS IoT Greengrass V2 utilise la même API de découverte que AWS IoT Greengrass V1 la réponse organise les informations selon des AWS IoT Greengrass V1 concepts, tels que les groupes Greengrass. La réponse contient une liste de groupes Greengrass. Dans AWS IoT Greengrass V2, chaque périphérique principal appartient à son propre groupe, où le groupe contient uniquement ce périphérique principal et ses informations de connectivité.

Exemple de documents de réponse à une découverte

Le document suivant montre la réponse d'un appareil client associé à un périphérique principal de Greengrass. Le périphérique principal possède un point de terminaison et un certificat CA.

```
{
```

```

"GGGroups": [
  {
    "GGGroupId": "greengrassV2-coreDevice-core-device-01-thing-name",
    "Cores": [
      {
        "thingArn": "core-device-01-thing-arn",
        "Connectivity": [
          {
            "id": "core-device-01-connection-id",
            "hostAddress": "core-device-01-address",
            "portNumber": core-device-01-port,
            "metadata": "core-device-01-description"
          }
        ]
      }
    ],
    "CAs": [
      "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----"
    ]
  }
]
}

```

Le document suivant montre la réponse d'un appareil client associé à deux périphériques principaux. Les appareils principaux ont plusieurs points de terminaison et plusieurs certificats de groupe CA.

```

{
  "GGGroups": [
    {
      "GGGroupId": "greengrassV2-coreDevice-core-device-01-thing-name",
      "Cores": [
        {
          "thingArn": "core-device-01-thing-arn",
          "Connectivity": [
            {
              "id": "core-device-01-connection-id",
              "hostAddress": "core-device-01-address",
              "portNumber": core-device-01-port,
              "metadata": "core-device-01-connection-1-description"
            },
            {
              "id": "core-device-01-connection-id-2",
              "hostAddress": "core-device-01-address-2",

```

```

        "portNumber": core-device-01-port-2,
        "metadata": "core-device-01-connection-2-description"
    }
]
},
"CAs": [
    "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----",
    "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----",
    "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----"
]
},
{
    "GGGroupId": "greengrassV2-coreDevice-core-device-02-thing-name",
    "Cores": [
        {
            "thingArn": "core-device-02-thing-arn",
            "Connectivity" : [
                {
                    "id": "core-device-02-connection-id",
                    "hostAddress": "core-device-02-address",
                    "portNumber": core-device-02-port,
                    "metadata": "core-device-02-connection-1-description"
                }
            ]
        }
    ],
    "CAs": [
        "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----",
        "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----",
        "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----"
    ]
}
]
}

```

Testez l'API de découverte avec cURL

Si vous l'avez cURL installée, vous pouvez tester l'API de découverte. L'exemple suivant indique les certificats d'un appareil client pour authentifier une demande auprès du point de terminaison de l'API de découverte Greengrass.

```
curl -i \
```

```
--cert 1a23bc4d56.cert.pem \
--key 1a23bc4d56.private.key \
https://greengrass-ats.iot.us-west-2.amazonaws.com:8443/greengrass/discover/
thing/MyClientDevice1
```

Note

L'argument indique de générer des en-têtes de réponse HTTP. Vous pouvez utiliser cette option pour identifier les erreurs.

Si la demande aboutit, cette commande produit une réponse similaire à l'exemple suivant.

```
{
  "GGGroups": [
    {
      "GGGroupId": "greengrassV2-coreDevice-MyGreengrassCore",
      "Cores": [
        {
          "thingArn": "arn:aws:iot:us-west-2:123456789012:thing/MyGreengrassCore",
          "Connectivity": [
            {
              "Id": "AUTOIP_192.168.1.4_1",
              "HostAddress": "192.168.1.5",
              "PortNumber": 8883,
              "Metadata": ""
            }
          ]
        }
      ],
      "CAs": [
        "-----BEGIN CERTIFICATE-----\ncert-contents\n-----END CERTIFICATE-----\n"
      ]
    }
  ]
}
```

Si la commande génère une erreur, voir [Résolution des problèmes de découverte de Greengrass](#).

Relayer les messages MQTT entre les appareils clients et AWS IoT Core

Vous pouvez relayer des messages MQTT et d'autres données entre les appareils clients et AWS IoT Core. Les appareils clients se connectent au composant du courtier MQTT qui s'exécute sur le périphérique principal. Par défaut, les appareils principaux ne relaient pas les messages ou les données MQTT entre les appareils clients et AWS IoT Core. Par défaut, les appareils clients ne peuvent communiquer qu'entre eux via MQTT.

Pour relayer des messages MQTT entre des appareils clients AWS IoT Core, configurez le [composant du pont MQTT](#) pour effectuer les opérations suivantes :

- Transmettre les messages des appareils clients vers AWS IoT Core.
- Transmettre les messages depuis AWS IoT Core les appareils clients vers les appareils clients.

Note

Le pont MQTT utilise QoS 1 pour publier et s' AWS IoT Core abonner, même lorsqu'un appareil client utilise QoS 0 pour publier et s'abonner au courtier MQTT local. Par conséquent, vous pouvez observer une latence supplémentaire lorsque vous relayez des messages MQTT depuis des appareils clients sur le broker MQTT local vers. AWS IoT Core Pour plus d'informations sur la configuration MQTT sur les appareils principaux, consultez [Configurer les délais d'expiration et les paramètres de cache du MQTT](#).

Rubriques

- [Configuration et déploiement du composant de pont MQTT](#)
- [Messages MQTT relayés](#)

Configuration et déploiement du composant de pont MQTT

Le composant du pont MQTT utilise une liste de mappages de sujets qui spécifient chacun une source et une destination de message. Pour relayer des messages entre les appareils clients AWS IoT Core, déployez le composant du pont MQTT et spécifiez chaque sujet source et destination dans la configuration du composant.

Pour déployer le composant de pont MQTT sur un périphérique principal ou un groupe de périphériques principaux, [créez un déploiement](#) incluant le `aws.greengrass.clientdevices.mqtt.Bridge` composant. Spécifiez les mappages de `sujetsmqttTopicMapping`, dans la configuration des composants du pont MQTT lors du déploiement.

L'exemple suivant définit un déploiement qui configure le composant du pont MQTT pour relayer des messages sur des sujets qui correspondent au filtre de `clients/+/hello/world` sujets des appareils clients vers. AWS IoT Core La mise à jour merge de configuration nécessite un objet JSON sérialisé. Pour de plus amples informations, veuillez consulter [Mettre à jour les configurations des composants](#).

Console

```
{
  "mqttTopicMapping": {
    "HelloWorldIotCore": {
      "topic": "clients/+/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    }
  }
}
```

AWS CLI

```
{
  "components": {
    "aws.greengrass.clientdevices.mqtt.Bridge": {
      "version": "2.0.0",
      "configurationUpdate": {
        "merge": "{\"mqttTopicMapping\":{\"HelloWorldIotCore\":{\"topic\":\"clients/+/hello/world\",\"source\":\"LocalMqtt\",\"target\":\"IotCore\"}}}"
      }
    }
  }
}
```

Messages MQTT relayés

Pour relayer les messages MQTT entre les appareils clients AWS IoT Core, [configurez et déployez le composant MQTT Bridge](#) et spécifiez les sujets à relayer.

Exemple Exemple : relayer des messages sur un sujet depuis des appareils clients vers AWS IoT Core

La configuration des composants du pont MQTT suivante spécifie le relais des messages sur des sujets qui correspondent au filtre de `clients/+/hello/world/event` sujets, des appareils clients vers. AWS IoT Core

```
{
  "mqttTopicMapping": {
    "HelloWorldEvent": {
      "topic": "clients/+/hello/world/event",
      "source": "LocalMqtt",
      "target": "IotCore"
    }
  }
}
```

Exemple Exemple : relayer des messages sur un sujet depuis AWS IoT Core des appareils clients

La configuration des composants du pont MQTT suivante spécifie le relais des messages sur les sujets qui correspondent au filtre de `clients/+/hello/world/event/response` sujets AWS IoT Core vers les appareils clients.

```
{
  "mqttTopicMapping": {
    "HelloWorldEventConfirmation": {
      "topic": "clients/+/hello/world/event/response",
      "source": "IotCore",
      "target": "LocalMqtt"
    }
  }
}
```

Interagir avec les appareils clients dans les composants

Vous pouvez développer des composants Greengrass personnalisés qui interagissent avec les appareils clients connectés à un appareil principal. Par exemple, vous pouvez développer des composants qui effectuent les opérations suivantes :

- Agissez sur les messages MQTT provenant des appareils clients et envoyez des données aux AWS Cloud destinations.
- Envoyez des messages MQTT aux appareils clients pour lancer des actions.

Les appareils clients se connectent à un périphérique principal et communiquent avec celui-ci par le biais du composant courtier MQTT qui s'exécute sur le périphérique principal. Par défaut, les appareils clients ne peuvent communiquer qu'entre eux via MQTT, et les composants Greengrass ne peuvent pas recevoir ces messages MQTT ni envoyer de messages aux appareils clients.

Les composants Greengrass utilisent l'[interface de publication/d'abonnement locale](#) pour communiquer sur un appareil principal. Pour communiquer avec les appareils clients dans les composants Greengrass, configurez le [composant pont MQTT](#) pour effectuer les opérations suivantes :

- Transférez les messages MQTT des appareils clients vers les services de publication/d'abonnement locaux.
- Transmettez les messages MQTT issus de la publication ou de l'abonnement locaux aux appareils clients.

Vous pouvez également interagir avec les ombres des appareils clients dans les composants Greengrass. Pour de plus amples informations, veuillez consulter [Interagissez avec les ombres des appareils clients et synchronisez-les](#).

Rubriques

- [Configuration et déploiement du composant de pont MQTT](#)
- [Recevoir des messages MQTT depuis les appareils clients](#)
- [Envoyer des messages MQTT aux appareils clients](#)

Configuration et déploiement du composant de pont MQTT

Le composant du pont MQTT utilise une liste de mappages de sujets qui spécifient chacun une source et une destination de message. Pour communiquer avec les appareils clients, déployez le composant du pont MQTT et spécifiez chaque sujet source et destination dans la configuration du composant.

Pour déployer le composant de pont MQTT sur un périphérique principal ou un groupe de périphériques principaux, [créez un déploiement](#) incluant le `aws.greengrass.clientdevices.mqtt.Bridge` composant. Spécifiez les mappages de sujets `mqttTopicMapping`, dans la configuration des composants du pont MQTT lors du déploiement.

L'exemple suivant définit un déploiement qui configure le composant du pont MQTT pour relayer le `clients/MyClientDevice1/hello/world` sujet des appareils clients vers le courtier de publication/d'abonnement local. La mise à jour merge de configuration nécessite un objet JSON sérialisé. Pour de plus amples informations, veuillez consulter [Mettre à jour les configurations des composants](#).

Console

```
{
  "mqttTopicMapping": {
    "HelloWorldPubsub": {
      "topic": "clients/MyClientDevice1/hello/world",
      "source": "LocalMqtt",
      "target": "Pubsub"
    }
  }
}
```

AWS CLI

```
{
  "components": {
    "aws.greengrass.clientdevices.mqtt.Bridge": {
      "version": "2.0.0",
      "configurationUpdate": {
        "merge": "\"mqttTopicMapping\":{\\\"HelloWorldPubsub\\\":{\\\"topic\\\":\\\"clients/MyClientDevice1/hello/world\\\",\\\"source\\\":\\\"LocalMqtt\\\",\\\"target\\\":\\\"Pubsub\\\"}}}"
      }
    }
  }
}
```

```
}  
...  
}  
}
```

Vous pouvez utiliser des caractères génériques de sujet MQTT pour relayer des messages sur des sujets qui correspondent à un filtre de sujet. Si vous utilisez le pont MQTT v2.2.0 ou version ultérieure, vous pouvez utiliser des caractères génériques de sujet MQTT dans les filtres de sujets lorsque le courtier source publie ou s'abonne en local. Pour plus d'informations, consultez la section [Configuration des composants du pont MQTT](#).

Recevoir des messages MQTT depuis les appareils clients

Vous pouvez vous abonner aux rubriques de publication/d'abonnement locales que vous configurez pour que le composant du pont MQTT reçoive des messages des appareils clients.

Pour recevoir des messages MQTT depuis des appareils clients dans des composants personnalisés

1. [Configurez et déployez le composant du pont MQTT](#) pour relayer les messages d'un sujet MQTT sur lequel les appareils clients publient vers un sujet de publication/d'abonnement local.
2. Utilisez l'interface IPC de publication/abonnement locale pour vous abonner au sujet où le pont MQTT relaie les messages. Pour plus d'informations, consultez [Publier/souscrire des messages locaux](#) et [SubscribeToTopic](#).

Le [didacticiel Connecter et tester les appareils clients](#) inclut une section dans laquelle vous développez un composant qui s'abonne aux messages provenant d'un appareil client. Pour de plus amples informations, veuillez consulter [Étape 4 : développer un composant qui communique avec les appareils clients](#).

Envoyer des messages MQTT aux appareils clients

Vous pouvez publier sur les rubriques de publication/d'abonnement locales que vous configurez pour que le composant du pont MQTT envoie des messages aux appareils clients.

Pour publier des messages MQTT sur les appareils clients dans des composants personnalisés

1. [Configurez et déployez le composant du pont MQTT](#) pour relayer les messages d'un sujet de publication/d'abonnement local vers un sujet MQTT auquel les appareils clients s'abonnent.

2. Utilisez l'interface IPC de publication/abonnement locale pour publier sur le sujet où le pont MQTT relaie les messages. Pour plus d'informations, consultez [Publier/souscrire des messages locaux](#) et [PublishToTopic](#).

Interagissez avec les ombres des appareils clients et synchronisez-les

Vous pouvez utiliser le [composant Shadow Manager](#) pour gérer les ombres locales, y compris les ombres du périphérique client. Vous pouvez utiliser le Shadow Manager pour effectuer les opérations suivantes :

- Interagissez avec les ombres des appareils clients dans les composants Greengrass.
- Synchronisez les ombres de l'appareil client avec AWS IoT Core.

Note

Par défaut, le composant Shadow Manager ne synchronise pas AWS IoT Core les ombres avec. Vous devez configurer le composant Shadow Manager pour spécifier les ombres du périphérique client à synchroniser.

Rubriques

- [Prérequis](#)
- [Permettre au Shadow Manager de communiquer avec les appareils clients](#)
- [Interagissez avec les ombres de l'appareil client dans les composants](#)
- [Synchroniser les ombres de l'appareil client avec AWS IoT Core](#)

Prérequis

Pour interagir avec les ombres du périphérique client et synchroniser les ombres du périphérique client avec celles-ci AWS IoT Core, un périphérique principal doit répondre aux exigences suivantes :

- Le périphérique principal doit exécuter les composants suivants, en plus des composants [Greengrass destinés à la prise en charge des appareils clients](#) :
 - [Greengrass nucleus](#) v2.6.0 ou version ultérieure

- [Shadow Manager](#) v2.2.0 ou version ultérieure
- [Pont MQTT](#) v2.2.0 ou version ultérieure
- Le composant [d'authentification du périphérique client](#) doit être configuré pour permettre aux appareils clients de communiquer sur des [sujets cachés](#).

Permettre au Shadow Manager de communiquer avec les appareils clients

Par défaut, le composant Shadow Manager ne gère pas les ombres du périphérique client. Pour activer cette fonctionnalité, vous devez relayer les messages MQTT entre les appareils clients et le composant Shadow Manager. Les appareils clients utilisent des messages MQTT pour recevoir et envoyer des mises à jour instantanées. [Le composant Shadow Manager s'abonne à l'interface de publication/d'abonnement locale de Greengrass. Vous pouvez donc configurer le composant du pont MQTT pour relayer les messages MQTT sur les sujets cachés de l'appareil.](#)

Le composant du pont MQTT utilise une liste de mappages de sujets qui spécifient chacun une source et une destination de message. Pour permettre au composant Shadow Manager de gérer les ombres du périphérique client, déployez le composant MQTT Bridge et spécifiez les sujets des ombres pour les ombres du périphérique client. Vous devez configurer le pont pour relayer les messages dans les deux sens entre le MQTT local et la publication/abonnement local.

Pour déployer le composant du pont MQTT sur un périphérique principal ou un groupe de périphériques principaux, [créez un déploiement](#) incluant le `aws.greengrass.clientdevices.mqtt.Bridge` composant. Spécifiez les mappages de `sujetsmqttTopicMapping`, dans la configuration des composants du pont MQTT lors du déploiement.

Utilisez les exemples suivants pour configurer le composant pont MQTT afin de permettre la communication entre les appareils clients et le composant Shadow Manager.

Note

Vous pouvez utiliser ces exemples de configuration dans la AWS IoT Greengrass console. Si vous utilisez l' AWS IoT Greengrass API, la mise à jour merge de configuration nécessite un objet JSON sérialisé. Vous devez donc sérialiser les objets JSON suivants sous forme de chaînes. Pour de plus amples informations, veuillez consulter [Mettre à jour les configurations des composants](#).

Exemple Exemple : gestion de tous les ombres des appareils clients

L'exemple de configuration du pont MQTT suivant permet au gestionnaire de ombres de gérer toutes les ombres pour tous les appareils clients.

```
{
  "mqttTopicMapping": {
    "ShadowsLocalMqttToPubsub": {
      "topic": "$aws/things/+ /shadow/#",
      "source": "LocalMqtt",
      "target": "Pubsub"
    },
    "ShadowsPubsubToLocalMqtt": {
      "topic": "$aws/things/+ /shadow/#",
      "source": "Pubsub",
      "target": "LocalMqtt"
    }
  }
}
```

Exemple Exemple : gestion des ombres pour un appareil client

L'exemple de configuration du pont MQTT suivant permet au gestionnaire de ombres de gérer toutes les ombres d'un périphérique client nommé `MyClientDevice`.

```
{
  "mqttTopicMapping": {
    "ShadowsLocalMqttToPubsub": {
      "topic": "$aws/things/MyClientDevice/shadow/#",
      "source": "LocalMqtt",
      "target": "Pubsub"
    },
    "ShadowsPubsubToLocalMqtt": {
      "topic": "$aws/things/MyClientDevice/shadow/#",
      "source": "Pubsub",
      "target": "LocalMqtt"
    }
  }
}
```

Exemple Exemple : gestion d'une ombre nommée pour tous les appareils clients

L'exemple de configuration du pont MQTT suivant permet à Shadow Manager de gérer un shadow nommé DeviceConfiguration pour tous les appareils clients.

```
{
  "mqttTopicMapping": {
    "ShadowsLocalMqttToPubsub": {
      "topic": "$aws/things/+/shadow/name/DeviceConfiguration/#",
      "source": "LocalMqtt",
      "target": "Pubsub"
    },
    "ShadowsPubsubToLocalMqtt": {
      "topic": "$aws/things/+/shadow/name/DeviceConfiguration/#",
      "source": "Pubsub",
      "target": "LocalMqtt"
    }
  }
}
```

Exemple Exemple : gestion des ombres anonymes de tous les appareils clients

L'exemple de configuration du pont MQTT suivant permet au gestionnaire d'ombres de gérer les ombres anonymes, mais pas les ombres nommées, pour tous les appareils clients.

```
{
  "mqttTopicMapping": {
    "DeleteShadowLocalMqttToPubsub": {
      "topic": "$aws/things/+/shadow/delete",
      "source": "LocalMqtt",
      "target": "Pubsub"
    },
    "DeleteShadowPubsubToLocalMqtt": {
      "topic": "$aws/things/+/shadow/delete/#",
      "source": "Pubsub",
      "target": "LocalMqtt"
    },
    "GetShadowLocalMqttToPubsub": {
      "topic": "$aws/things/+/shadow/get",
      "source": "LocalMqtt",
      "target": "Pubsub"
    },
    "GetShadowPubsubToLocalMqtt": {
```

```
    "topic": "$aws/things/+ /shadow/get/#",
    "source": "Pubsub",
    "target": "LocalMqtt"
  },
  "UpdateShadowLocalMqttToPubsub": {
    "topic": "$aws/things/+ /shadow/update",
    "source": "LocalMqtt",
    "target": "Pubsub"
  },
  "UpdateShadowPubsubToLocalMqtt": {
    "topic": "$aws/things/+ /shadow/update/#",
    "source": "Pubsub",
    "target": "LocalMqtt"
  }
}
```

Interagissez avec les ombres de l'appareil client dans les composants

Vous pouvez développer des composants personnalisés qui utilisent le service parallèle local pour lire et modifier les documents fictifs locaux des appareils clients. Pour de plus amples informations, veuillez consulter [Interagir avec les ombres dans les composants](#).

Synchroniser les ombres de l'appareil client avec AWS IoT Core

Vous pouvez configurer le composant Shadow Manager pour synchroniser les états cachés du périphérique client local avec AWS IoT Core. Pour de plus amples informations, veuillez consulter [Synchronisez les ombres de l'appareil local avec AWS IoT Core](#).

Utilisation IPv6 pour la messagerie locale

Vous pouvez configurer le composant de détection IP à utiliser IPv6 pour envoyer des messages locaux.

Note

Vous devez disposer du détecteur IP v2.2.0 ou version ultérieure pour pouvoir IPv6 envoyer des messages locaux.

Vous pouvez déployer le [composant de détection IP](#) pour détecter et utiliser IPv6 des adresses. Vous devez mettre à jour la configuration du composant du détecteur IP à utiliser à la IPv6 place de IPv4. Pour de plus amples informations, veuillez consulter [Utiliser le détecteur IP pour gérer automatiquement les points de terminaison](#).

Rubriques

- [Configurer le détecteur IP à utiliser IPv6](#)

Configurer le détecteur IP à utiliser IPv6

Si vous disposez d'une configuration réseau simple, telle que les appareils clients sur le même réseau que le périphérique principal, vous pouvez déployer le [composant de détection IP](#) à utiliser IPv6 pour la messagerie locale.

Le composant du détecteur IP remplace les points de terminaison que vous avez définis manuellement.

Important

La AWS IoT politique de l'appareil principal doit `greengrass:UpdateConnectivityInfo` autoriser l'utilisation du composant de détection IP. Pour plus d'informations, consultez [AWS IoT politiques relatives aux opérations du plan de données](#) et [Configuration de la AWS IoT politique des objets](#).

Vous pouvez effectuer l'une des opérations suivantes pour déployer le composant de détection IP :

- Utilisez la page Configurer la découverte de la console. Pour de plus amples informations, veuillez consulter [Configuration de la découverte du cloud \(console\)](#).
- Créez et révisiez les déploiements pour inclure le détecteur IP. Vous pouvez utiliser la console ou l' AWS CLI AWS API pour gérer les déploiements. Pour de plus amples informations, veuillez consulter [Créer des déploiements](#).

Déployer le composant de détection IP (console)

1. Dans le menu de navigation de la [AWS IoT Greengrass console](#), sélectionnez Composants.

2. Sur la page Composants, choisissez l'onglet Composants publics, puis choisissez `aws.greengrass.clientdevices.IPDetector`.
3. Dans la page `aws.greengrass.clientdevices.IPDetector`, choisissez Déployer.
4. Dans Ajouter au déploiement, choisissez un déploiement existant à réviser ou choisissez de créer un nouveau déploiement, puis choisissez Suivant.
5. Si vous avez choisi de créer un nouveau déploiement, choisissez le périphérique principal ou le groupe d'objets cible pour le déploiement. Sur la page Spécifier la cible, sous Cible de déploiement, choisissez un périphérique principal ou un groupe d'objets, puis cliquez sur Suivant.
6. Sur la page Sélectionner les composants, vérifiez que `aws.greengrass.clientdevices.IPDetector` le composant est sélectionné, choisissez Next.
7. Sur la page Configurer les composants, sélectionnez `aws.greengrass.clientdevices.IPDetector`, puis procédez comme suit :
 - a. Choisissez Configure component (Configurer un composant).
 - b. Dans la configuration `aws.greengrass.clientdevices.IPDetector`, sous Mise à jour de la configuration, dans Configuration à fusionner, vous pouvez saisir une mise à jour de configuration pour configurer le composant du détecteur IP. Vous pouvez spécifier l'une des options de configuration suivantes. Définissez `includeIPv4Addrs` sur `false` et `includeIPv6Addrs` sur `true`. Vous pouvez ensuite mettre à jour les autres options IPv6 de configuration.
 - `defaultPort`— (Facultatif) Le port du broker MQTT à signaler lorsque ce composant détecte des adresses IP. Vous devez spécifier ce paramètre si vous configurez le broker MQTT pour utiliser un port différent du port par défaut 8883.
 - `includeIPv4LoopbackAddrs`— (Facultatif) Vous pouvez activer cette option pour détecter et signaler les adresses de IPv4 boucle. Il s'agit d'adresses IP, par exemple `localhost`, où un appareil peut communiquer avec lui-même. Utilisez cette option dans les environnements de test dans lesquels le périphérique principal et le périphérique client s'exécutent sur le même système.
 - `includeIPv4LinkLocalAddrs`— (Facultatif) Vous pouvez activer cette option pour détecter et signaler les adresses IPv4 [locales liées aux liens](#). Utilisez cette option si le réseau du périphérique principal ne dispose pas du protocole DHCP (Dynamic Host Configuration Protocol) ou d'adresses IP attribuées de manière statique.

- `includeIPv6LoopbackAddr`— (Facultatif) Vous pouvez activer cette option pour détecter et signaler les adresses de IPv6 boucle. Il s'agit d'adresses IP, par exemple `localhost`, où un appareil peut communiquer avec lui-même. Utilisez cette option dans les environnements de test dans lesquels le périphérique principal et le périphérique client s'exécutent sur le même système. Vous devez définir `includeIPv4Addr` sur `false` et `includeIPv6Addr` sur `true` pour utiliser cette option. Vous devez disposer du détecteur IP v2.2.0 ou version ultérieure pour utiliser cette option.
- `includeIPv6LinkLocalAddr`— (Facultatif) Vous pouvez activer cette option pour détecter et signaler les adresses IPv6 [locales liées aux liens](#). Utilisez cette option si le réseau du périphérique principal ne dispose pas du protocole DHCP (Dynamic Host Configuration Protocol) ou d'adresses IP attribuées de manière statique. Vous devez définir `includeIPv4Addr` sur `false` et `includeIPv6Addr` sur `true` pour utiliser cette option. Vous devez disposer du détecteur IP v2.2.0 ou version ultérieure pour utiliser cette option.
- `includeIPv4Addr`— (Facultatif) La valeur par défaut est définie sur `true`. Vous pouvez activer cette option pour publier IPv4 les adresses présentes sur l'appareil principal. Vous devez disposer du détecteur IP v2.2.0 ou version ultérieure pour utiliser cette option.
- `includeIPv6Addr`— (Facultatif) Vous pouvez activer cette option pour publier IPv6 les adresses trouvées sur l'appareil principal. Réglez `false` sur `includeIPv4Addr` pour utiliser cette option. Vous devez disposer du détecteur IP v2.2.0 ou version ultérieure pour utiliser cette option.

La mise à jour de configuration peut ressembler à l'exemple suivant.

```
{
  "defaultPort": "8883",
  "includeIPv4LoopbackAddr": false,
  "includeIPv4LinkLocalAddr": false,
  "includeIPv6LoopbackAddr": true,
  "includeIPv6LinkLocalAddr": true,
  "includeIPv4Addr": false,
  "includeIPv6Addr": true
}
```

- c. Choisissez Confirmer pour fermer le modal, puis cliquez sur Suivant.
8. Sur la page Configure advanced settings (Configurer les paramètres avancés), conservez les paramètres de configuration par défaut et choisissez Next (Suivant).

9. Sur la page Review (Révision), choisissez Deploy (Déployer).

Le déploiement peut prendre jusqu'à une minute.

Déployer le composant du détecteur IP (AWS CLI)

Pour déployer le composant du détecteur IP, créez un document de déploiement qui inclut `aws.greengrass.clientdevices.IPDetector` l'componentsubject et spécifiez la mise à jour de configuration du composant. Suivez les instructions [Créer des déploiements](#) pour créer un nouveau déploiement ou modifier un déploiement existant.

Vous pouvez spécifier l'une des options suivantes pour configurer le composant du détecteur IP lorsque vous créez le document de déploiement :

- `defaultPort`— (Facultatif) Le port du broker MQTT à signaler lorsque ce composant détecte des adresses IP. Vous devez spécifier ce paramètre si vous configurez le broker MQTT pour utiliser un port différent du port par défaut 8883.
- `includeIPv4LoopbackAddr`— (Facultatif) Vous pouvez activer cette option pour détecter et signaler les adresses de IPv4 boucle. Il s'agit d'adresses IP, par exemple `localhost`, où un appareil peut communiquer avec lui-même. Utilisez cette option dans les environnements de test dans lesquels le périphérique principal et le périphérique client s'exécutent sur le même système.
- `includeIPv4LinkLocalAddr`— (Facultatif) Vous pouvez activer cette option pour détecter et signaler les adresses IPv4 [locales liées aux liens](#). Utilisez cette option si le réseau du périphérique principal ne dispose pas du protocole DHCP (Dynamic Host Configuration Protocol) ou d'adresses IP attribuées de manière statique.
- `includeIPv6LoopbackAddr`— (Facultatif) Vous pouvez activer cette option pour détecter et signaler les adresses de IPv6 boucle. Il s'agit d'adresses IP, par exemple `localhost`, où un appareil peut communiquer avec lui-même. Utilisez cette option dans les environnements de test dans lesquels le périphérique principal et le périphérique client s'exécutent sur le même système. Vous devez `includeIPv4Addr` définir sur `false` et `includeIPv6Addr` sur `true` pour utiliser cette option. Vous devez disposer du détecteur IP v2.2.0 ou version ultérieure pour utiliser cette option.
- `includeIPv6LinkLocalAddr`— (Facultatif) Vous pouvez activer cette option pour détecter et signaler les adresses IPv6 [locales liées aux liens](#). Utilisez cette option si le réseau du périphérique principal ne dispose pas du protocole DHCP (Dynamic Host Configuration Protocol) ou d'adresses IP attribuées de manière statique. Vous devez `includeIPv4Addr` définir sur `false` et

`includeIPv6Addrs` sur `true` pour utiliser cette option. Vous devez disposer du détecteur IP v2.2.0 ou version ultérieure pour utiliser cette option.

- `includeIPv4Addrs`— (Facultatif) La valeur par défaut est définie sur `true`. Vous pouvez activer cette option pour publier IPv4 les adresses présentes sur l'appareil principal. Vous devez disposer du détecteur IP v2.2.0 ou version ultérieure pour utiliser cette option.
- `includeIPv6Addrs`— (Facultatif) Vous pouvez activer cette option pour publier IPv6 les adresses trouvées sur l'appareil principal. Réglez `false` sur `includeIPv4Addrs` pour utiliser cette option. Vous devez disposer du détecteur IP v2.2.0 ou version ultérieure pour utiliser cette option.

L'exemple de document de déploiement partiel suivant indique à utiliser IPv6.

```
{
  ...,
  "components": {
    ...,
    "aws.greengrass.clientdevices.IPDetector": {
      "componentVersion": "2.1.1",
      "configurationUpdate": {
        "merge": "{\"defaultPort\":\"8883\",}"
      }
    }
  }
}
```

Résolution des problèmes liés aux appareils clients

Utilisez les informations de dépannage et les solutions de cette section pour résoudre les problèmes liés aux appareils clients Greengrass et à leurs composants.

Rubriques

- [Problèmes liés à Greengrass Discovery](#)
- [Problèmes de connexion MQTT](#)

Problèmes liés à Greengrass Discovery

Utilisez les informations suivantes pour résoudre les problèmes liés à Greengrass Discovery. Ces problèmes peuvent survenir lorsque les appareils clients utilisent l'[API de découverte Greengrass](#) pour identifier un appareil principal de Greengrass auquel ils peuvent se connecter.

Rubriques

- [Problèmes liés à la découverte de Greengrass \(API HTTP\)](#)
- [Problèmes de découverte de Greengrass \(Kit SDK des appareils AWS IoT version 2 pour Python\)](#)
- [Problèmes de découverte de Greengrass \(Kit SDK des appareils AWS IoT version 2 pour C++\)](#)
- [Problèmes de découverte de Greengrass \(Kit SDK des appareils AWS IoT version 2 pour JavaScript\)](#)
- [Problèmes de découverte de Greengrass \(Kit SDK des appareils AWS IoT v2 pour Java\)](#)

Problèmes liés à la découverte de Greengrass (API HTTP)

Utilisez les informations suivantes pour résoudre les problèmes liés à Greengrass Discovery. Ces erreurs peuvent s'afficher si vous [testez l'API de découverte avec cURL](#).

Rubriques

- [curl: \(52\) Empty reply from server](#)
- [HTTP 403: {"message":null,"traceId":"a1b2c3d4-5678-90ab-cdef-11111EXAMPLE"}](#)
- [HTTP 404: {"errorMessage":"The thing provided for discovery was not found"}](#)

curl: (52) Empty reply from server

Cette erreur peut s'afficher si vous spécifiez un AWS IoT certificat inactif dans la demande.

Vérifiez que l'appareil client possède un certificat attaché et que le certificat est actif. Pour plus d'informations, voir [Attacher un objet ou une politique à un certificat client](#) et [Activer ou désactiver un certificat client](#) dans le Guide du AWS IoT Core développeur.

HTTP 403: {"message":null,"traceId":"a1b2c3d4-5678-90ab-cdef-11111EXAMPLE"}

Cette erreur peut s'afficher si l'appareil client n'est pas autorisé à effectuer des `greengrass:Discover` appels par lui-même.

Vérifiez que le certificat de l'appareil client dispose d'une politique qui l'autorise `greengrass:Discover`. Vous ne pouvez pas utiliser les [variables de politique des objets](#) (`iot:Connection.Thing.*`) dans la Resource section relative à cette autorisation. Pour de plus amples informations, veuillez consulter [Authentification et autorisation de découverte](#).

HTTP 404: {"errorMessage":"The thing provided for discovery was not found"}

Cette erreur peut s'afficher dans les cas suivants :

- L'appareil client n'est associé à aucun appareil ou AWS IoT Greengrass V1 groupe principal de Greengrass.
- Aucun des appareils ou AWS IoT Greengrass V1 groupes principaux Greengrass associés à l'appareil client ne possède de point de terminaison de courtier MQTT.
- Aucun des périphériques principaux Greengrass associés à l'appareil client n'exécute le composant d'[authentification du périphérique client](#).

Vérifiez que l'appareil client est associé au périphérique principal auquel vous souhaitez qu'il se connecte. Vérifiez ensuite que le périphérique principal exécute le [composant d'authentification du périphérique client](#) et possède au moins un point de terminaison de courtier MQTT. Pour plus d'informations, consultez les ressources suivantes :

- [Associer des appareils clients](#)
- [Gérez les principaux points de terminaison des appareils](#)
- [Configuration de la découverte du cloud \(console\)](#)

Problèmes de découverte de Greengrass (Kit SDK des appareils AWS IoT version 2 pour Python)

Utilisez les informations suivantes pour résoudre les problèmes liés à la découverte de Greengrass dans [Kit SDK des appareils AWS IoT la version v2 pour Python](#).

Rubriques

- [awscli.exceptions.AwsCliError: AWS_ERROR_HTTP_CONNECTION_CLOSED: The connection has closed or is closing.](#)
- [awscli.greengrass_discovery.DiscoveryException: \('Error during discover call: response_code=403', 403\)](#)

- [awsiot.greengrass_discovery.DiscoveryException: \('Error during discover call: response_code=404', 404\)](#)

`awsrt.exceptions.AwsCrtError: AWS_ERROR_HTTP_CONNECTION_CLOSED: The connection has closed or is closing.`

Cette erreur peut s'afficher si vous spécifiez un AWS IoT certificat inactif dans la demande.

Vérifiez que l'appareil client possède un certificat attaché et que le certificat est actif. Pour plus d'informations, voir [Attacher un objet ou une politique à un certificat client](#) et [Activer ou désactiver un certificat client](#) dans le Guide du AWS IoT Core développeur.

`awsiot.greengrass_discovery.DiscoveryException: ('Error during discover call: response_code=403', 403)`

Cette erreur peut s'afficher si l'appareil client n'est pas autorisé à effectuer des `greengrass:Discover` appels par lui-même.

Vérifiez que le certificat de l'appareil client dispose d'une politique qui l'autorise `greengrass:Discover`. Vous ne pouvez pas utiliser les [variables de politique des](#) objets (`iot:Connection.Thing.*`) dans la Resource section relative à cette autorisation. Pour de plus amples informations, veuillez consulter [Authentification et autorisation de découverte](#).

`awsiot.greengrass_discovery.DiscoveryException: ('Error during discover call: response_code=404', 404)`

Cette erreur peut s'afficher dans les cas suivants :

- L'appareil client n'est associé à aucun appareil ou AWS IoT Greengrass V1 groupe principal de Greengrass.
- Aucun des appareils ou AWS IoT Greengrass V1 groupes principaux Greengrass associés à l'appareil client ne possède de point de terminaison de courtier MQTT.
- Aucun des périphériques principaux Greengrass associés à l'appareil client n'exécute le composant d'[authentification du périphérique client](#).

Vérifiez que l'appareil client est associé au périphérique principal auquel vous souhaitez qu'il se connecte. Vérifiez ensuite que le périphérique principal exécute le [composant d'authentification du périphérique client](#) et possède au moins un point de terminaison de courtier MQTT. Pour plus d'informations, consultez les ressources suivantes :

- [Associer des appareils clients](#)
- [Gérez les principaux points de terminaison des appareils](#)
- [Configuration de la découverte du cloud \(console\)](#)

Problèmes de découverte de Greengrass (Kit SDK des appareils AWS IoT version 2 pour C++)

Utilisez les informations suivantes pour résoudre les problèmes liés à Greengrass Discovery dans [Kit SDK des appareils AWS IoT la version v2](#) pour C++.

Rubriques

- [aws-c-http: AWS_ERROR_HTTP_CONNECTION_CLOSED, The connection has closed or is closing.](#)
- [aws-c-common: AWS_ERROR_UNKNOWN, Unknown error. \(HTTP 403\)](#)
- [aws-c-common: AWS_ERROR_UNKNOWN, Unknown error. \(HTTP 404\)](#)

aws-c-http: AWS_ERROR_HTTP_CONNECTION_CLOSED, The connection has closed or is closing.

Cette erreur peut s'afficher si vous spécifiez un AWS IoT certificat inactif dans la demande.

Vérifiez que l'appareil client possède un certificat attaché et que le certificat est actif. Pour plus d'informations, voir [Attacher un objet ou une politique à un certificat client](#) et [Activer ou désactiver un certificat client](#) dans le Guide du AWS IoT Core développeur.

aws-c-common: AWS_ERROR_UNKNOWN, Unknown error. (HTTP 403)

Cette erreur peut s'afficher si l'appareil client n'est pas autorisé à effectuer des `greengrass:Discover` appels par lui-même.

Vérifiez que le certificat de l'appareil client dispose d'une politique qui l'autorise `greengrass:Discover`. Vous ne pouvez pas utiliser les [variables de politique des](#) objets (`iot:Connection.Thing.*`) dans la Resource section relative à cette autorisation. Pour de plus amples informations, veuillez consulter [Authentification et autorisation de découverte](#).

aws-c-common: AWS_ERROR_UNKNOWN, Unknown error. (HTTP 404)

Cette erreur peut s'afficher dans les cas suivants :

- L'appareil client n'est associé à aucun appareil ou AWS IoT Greengrass V1 groupe principal de Greengrass.
- Aucun des appareils ou AWS IoT Greengrass V1 groupes principaux Greengrass associés à l'appareil client ne possède de point de terminaison de courtier MQTT.
- Aucun des périphériques principaux Greengrass associés à l'appareil client n'exécute le composant d'[authentification du périphérique client](#).

Vérifiez que l'appareil client est associé au périphérique principal auquel vous souhaitez qu'il se connecte. Vérifiez ensuite que le périphérique principal exécute le [composant d'authentification du périphérique client](#) et possède au moins un point de terminaison de courtier MQTT. Pour plus d'informations, consultez les ressources suivantes :

- [Associer des appareils clients](#)
- [Gérez les principaux points de terminaison des appareils](#)
- [Configuration de la découverte du cloud \(console\)](#)

Problèmes de découverte de Greengrass (Kit SDK des appareils AWS IoT version 2 pour) JavaScript

Utilisez les informations suivantes pour résoudre les problèmes liés à Greengrass Discovery dans [Kit SDK des appareils AWS IoT la](#) version v2 pour. JavaScript

Rubriques

- [Error: aws-c-http: AWS_ERROR_HTTP_CONNECTION_CLOSED, The connection has closed or is closing.](#)
- [Error: Discovery failed \(headers: \[object Object\]\) { response_code: 403 }](#)
- [Error: Discovery failed \(headers: \[object Object\]\) { response_code: 404 }](#)
- [Error: Discovery failed \(headers: \[object Object\]\)](#)

Error: aws-c-http: AWS_ERROR_HTTP_CONNECTION_CLOSED, The connection has closed or is closing.

Cette erreur peut s'afficher si vous spécifiez un AWS IoT certificat inactif dans la demande.

Vérifiez que l'appareil client possède un certificat attaché et que le certificat est actif. Pour plus d'informations, voir [Attacher un objet ou une politique à un certificat client](#) et [Activer ou désactiver un certificat client](#) dans le Guide du AWS IoT Core développeur.

Error: Discovery failed (headers: [object Object]) { response_code: 403 }

Cette erreur peut s'afficher si l'appareil client n'est pas autorisé à effectuer des `greengrass:Discover` appels par lui-même.

Vérifiez que le certificat de l'appareil client dispose d'une politique qui l'autorise `greengrass:Discover`. Vous ne pouvez pas utiliser les [variables de politique des](#) objets (`iot:Connection.Thing.*`) dans la Resource section relative à cette autorisation. Pour de plus amples informations, veuillez consulter [Authentification et autorisation de découverte](#).

Error: Discovery failed (headers: [object Object]) { response_code: 404 }

Cette erreur peut s'afficher dans les cas suivants :

- L'appareil client n'est associé à aucun appareil ou AWS IoT Greengrass V1 groupe principal de Greengrass.
- Aucun des appareils ou AWS IoT Greengrass V1 groupes principaux Greengrass associés à l'appareil client ne possède de point de terminaison de courtier MQTT.
- Aucun des périphériques principaux Greengrass associés à l'appareil client n'exécute le composant d'[authentification du périphérique client](#).

Vérifiez que l'appareil client est associé au périphérique principal auquel vous souhaitez qu'il se connecte. Vérifiez ensuite que le périphérique principal exécute le [composant d'authentification du périphérique client](#) et possède au moins un point de terminaison de courtier MQTT. Pour plus d'informations, consultez les ressources suivantes :

- [Associer des appareils clients](#)
- [Gérez les principaux points de terminaison des appareils](#)
- [Configuration de la découverte du cloud \(console\)](#)

Error: Discovery failed (headers: [object Object])

Cette erreur peut s'afficher (sans code de réponse HTTP) lorsque vous exécutez l'exemple de découverte Greengrass. Cette erreur peut se produire pour plusieurs raisons.

- Cette erreur peut s'afficher si l'appareil client n'est pas autorisé à effectuer des `greengrass:Discover` appels par lui-même.

Vérifiez que le certificat de l'appareil client dispose d'une politique qui l'autorise `greengrass:Discover`. Vous ne pouvez pas utiliser les [variables de politique des objets](#) (`iot:Connection.Thing.*`) dans la Resource section relative à cette autorisation. Pour de plus amples informations, veuillez consulter [Authentification et autorisation de découverte](#).

- Cette erreur peut s'afficher dans les cas suivants :
 - L'appareil client n'est associé à aucun appareil ou AWS IoT Greengrass V1 groupe principal de Greengrass.
 - Aucun des appareils ou AWS IoT Greengrass V1 groupes principaux Greengrass associés à l'appareil client ne possède de point de terminaison de courtier MQTT.
 - Aucun des périphériques principaux Greengrass associés à l'appareil client n'exécute le composant d'[authentification du périphérique client](#).

Vérifiez que l'appareil client est associé au périphérique principal auquel vous souhaitez qu'il se connecte. Vérifiez ensuite que le périphérique principal exécute le [composant d'authentification du périphérique client](#) et possède au moins un point de terminaison de courtier MQTT. Pour plus d'informations, consultez les ressources suivantes :

- [Associer des appareils clients](#)
- [Gérez les principaux points de terminaison des appareils](#)
- [Configuration de la découverte du cloud \(console\)](#)

Problèmes de découverte de Greengrass (Kit SDK des appareils AWS IoT v2 pour Java)

Utilisez les informations suivantes pour résoudre les problèmes liés à Greengrass Discovery dans [Kit SDK des appareils AWS IoT la version v2](#) pour Java.

Rubriques

- [software.amazon.awssdk.crt.CrtRuntimeException: Error Getting Response Status Code from HttpStream. \(aws_last_error: AWS_ERROR_HTTP_DATA_NOT_AVAILABLE\(2062\), This data is not yet available.\)](#)
- [java.lang.RuntimeException: Error x-amzn-ErrorType\(403\)](#)
- [java.lang.RuntimeException: Error x-amzn-ErrorType\(404\)](#)

software.amazon.awssdk.crt.CrtRuntimeException: Error Getting Response Status Code from HttpStream. (aws_last_error: AWS_ERROR_HTTP_DATA_NOT_AVAILABLE(2062), This data is not yet available.)

Cette erreur peut s'afficher si vous spécifiez un AWS IoT certificat inactif dans la demande.

Vérifiez que l'appareil client possède un certificat attaché et que le certificat est actif. Pour plus d'informations, voir [Attacher un objet ou une politique à un certificat client](#) et [Activer ou désactiver un certificat client](#) dans le Guide du AWS IoT Core développeur.

java.lang.RuntimeException: Error x-amzn-ErrorType(403)

Cette erreur peut s'afficher si l'appareil client n'est pas autorisé à effectuer des `greengrass:Discover` appels par lui-même.

Vérifiez que le certificat de l'appareil client dispose d'une politique qui l'autorise `greengrass:Discover`. Vous ne pouvez pas utiliser les [variables de politique des](#) objets (`iot:Connection.Thing.*`) dans la Resource section relative à cette autorisation. Pour de plus amples informations, veuillez consulter [Authentification et autorisation de découverte](#).

java.lang.RuntimeException: Error x-amzn-ErrorType(404)

Cette erreur peut s'afficher dans les cas suivants :

- L'appareil client n'est associé à aucun appareil ou AWS IoT Greengrass V1 groupe principal de Greengrass.
- Aucun des appareils ou AWS IoT Greengrass V1 groupes principaux Greengrass associés à l'appareil client ne possède de point de terminaison de courtier MQTT.
- Aucun des périphériques principaux Greengrass associés à l'appareil client n'exécute le composant d'[authentification du périphérique client](#).

Vérifiez que l'appareil client est associé au périphérique principal auquel vous souhaitez qu'il se connecte. Vérifiez ensuite que le périphérique principal exécute le [composant d'authentification du périphérique client](#) et possède au moins un point de terminaison de courtier MQTT. Pour plus d'informations, consultez les ressources suivantes :

- [Associer des appareils clients](#)
- [Gérez les principaux points de terminaison des appareils](#)
- [Configuration de la découverte du cloud \(console\)](#)

Problèmes de connexion MQTT

Utilisez les informations suivantes pour résoudre les problèmes liés aux connexions MQTT des appareils clients. Ces problèmes peuvent survenir lorsque des appareils clients tentent de se connecter à un périphérique principal via MQTT.

Rubriques

- [io.moquette.broker.Authorizator: Client does not have read permissions on the topic](#)
- [Problèmes de connexion MQTT \(Python\)](#)
- [Problèmes de connexion MQTT \(C++\)](#)
- [Problèmes de connexion MQTT \(Java\)](#)
- [Problèmes de connexion MQTT \(\) JavaScript](#)

io.moquette.broker.Authorizator: Client does not have read permissions on the topic

Cette erreur peut s'afficher dans les journaux de Greengrass lorsqu'un appareil client essaie de s'abonner à un sujet MQTT alors qu'il n'en a pas l'autorisation. Le message d'erreur inclut le sujet.

Vérifiez que la configuration du [composant d'authentification de l'appareil client](#) inclut les éléments suivants :

- Groupe d'appareils correspondant à l'appareil client.
- Une politique d'autorisation de l'appareil client pour ce groupe d'appareils qui accorde `l'mqtt:subscribe` autorisation pour le sujet.

Pour plus d'informations sur le déploiement et la configuration du composant d'authentification de l'appareil client, consultez les rubriques suivantes :

- [Configuration de la découverte du cloud \(console\)](#)
- [Authentification de l'appareil client](#)
- [Créer des déploiements](#)

Problèmes de connexion MQTT (Python)

Utilisez les informations suivantes pour résoudre les problèmes liés aux connexions MQTT des appareils clients lorsque vous utilisez la [Kit SDK des appareils AWS IoT version v2 pour Python](#).

Rubriques

- [AWS_ERROR_MQTT_PROTOCOL_ERROR: Protocol error occurred](#)
- [AWS_ERROR_MQTT_UNEXPECTED_HANGUP: Unexpected hangup occurred](#)

AWS_ERROR_MQTT_PROTOCOL_ERROR: Protocol error occurred

Cette erreur peut s'afficher si le [composant d'authentification du périphérique client](#) ne définit pas de politique d'autorisation du périphérique client autorisant le périphérique client à se connecter.

Vérifiez que la configuration du composant d'authentification de l'appareil client inclut les éléments suivants :

- Groupe d'appareils correspondant à l'appareil client.
- Une politique d'autorisation du dispositif client pour ce groupe d'appareils qui accorde l'mqtt:connect autorisation au dispositif client.

Pour plus d'informations sur le déploiement et la configuration du composant d'authentification de l'appareil client, consultez les rubriques suivantes :

- [Configuration de la découverte du cloud \(console\)](#)
- [Authentification de l'appareil client](#)
- [Créer des déploiements](#)

AWS_ERROR_MQTT_UNEXPECTED_HANGUP: Unexpected hangup occurred

Cette erreur peut s'afficher si le [composant d'authentification du périphérique client](#) ne définit pas de politique d'autorisation du périphérique client autorisant le périphérique client à se connecter.

Vérifiez que la configuration du composant d'authentification de l'appareil client inclut les éléments suivants :

- Groupe d'appareils correspondant à l'appareil client.
- Une politique d'autorisation du dispositif client pour ce groupe d'appareils qui accorde l'mqtt:connect autorisation au dispositif client.

Pour plus d'informations sur le déploiement et la configuration du composant d'authentification de l'appareil client, consultez les rubriques suivantes :

- [Configuration de la découverte du cloud \(console\)](#)
- [Authentification de l'appareil client](#)
- [Créer des déploiements](#)

Problèmes de connexion MQTT (C++)

Utilisez les informations suivantes pour résoudre les problèmes liés aux connexions MQTT des appareils clients lorsque vous utilisez la [Kit SDK des appareils AWS IoT version v2 pour C++](#).

Rubriques

- [AWS_ERROR_MQTT_PROTOCOL_ERROR: Protocol error occurred](#)
- [AWS_ERROR_MQTT_UNEXPECTED_HANGUP: Unexpected hangup occurred](#)

AWS_ERROR_MQTT_PROTOCOL_ERROR: Protocol error occurred

Cette erreur peut s'afficher si le [composant d'authentification du périphérique client](#) ne définit pas de politique d'autorisation du périphérique client autorisant le périphérique client à se connecter.

Vérifiez que la configuration du composant d'authentification de l'appareil client inclut les éléments suivants :

- Groupe d'appareils correspondant à l'appareil client.
- Une politique d'autorisation du dispositif client pour ce groupe d'appareils qui accorde l'mqtt:connect autorisation au dispositif client.

Pour plus d'informations sur le déploiement et la configuration du composant d'authentification de l'appareil client, consultez les rubriques suivantes :

- [Configuration de la découverte du cloud \(console\)](#)
- [Authentification de l'appareil client](#)
- [Créer des déploiements](#)

AWS_ERROR_MQTT_UNEXPECTED_HANGUP: Unexpected hangup occurred

Cette erreur peut s'afficher si le [composant d'authentification du périphérique client](#) ne définit pas de politique d'autorisation du périphérique client autorisant le périphérique client à se connecter.

Vérifiez que la configuration du composant d'authentification de l'appareil client inclut les éléments suivants :

- Groupe d'appareils correspondant à l'appareil client.
- Une politique d'autorisation du dispositif client pour ce groupe d'appareils qui accorde `mqtt:connect` autorisation au dispositif client.

Pour plus d'informations sur le déploiement et la configuration du composant d'authentification de l'appareil client, consultez les rubriques suivantes :

- [Configuration de la découverte du cloud \(console\)](#)
- [Authentification de l'appareil client](#)
- [Créer des déploiements](#)

Problèmes de connexion MQTT (Java)

Utilisez les informations suivantes pour résoudre les problèmes liés aux connexions MQTT des appareils clients lorsque vous utilisez la [Kit SDK des appareils AWS IoT version v2 pour Java](#).

Rubriques

- [software.amazon.awssdk.crt.mqtt.MqttException: Protocol error occurred](#)
- [AWS_ERROR_MQTT_UNEXPECTED_HANGUP: Unexpected hangup occurred](#)

`software.amazon.awssdk.crt.mqtt.MqttException: Protocol error occurred`

Cette erreur peut s'afficher si le [composant d'authentification du périphérique client](#) ne définit pas de politique d'autorisation du périphérique client autorisant le périphérique client à se connecter.

Vérifiez que la configuration du composant d'authentification de l'appareil client inclut les éléments suivants :

- Groupe d'appareils correspondant à l'appareil client.
- Une politique d'autorisation du dispositif client pour ce groupe d'appareils qui accorde `mqtt:connect` autorisation au dispositif client.

Pour plus d'informations sur le déploiement et la configuration du composant d'authentification de l'appareil client, consultez les rubriques suivantes :

- [Configuration de la découverte du cloud \(console\)](#)
- [Authentification de l'appareil client](#)
- [Créer des déploiements](#)

`AWS_ERROR_MQTT_UNEXPECTED_HANGUP`: Unexpected hangup occurred

Cette erreur peut s'afficher si le [composant d'authentification du périphérique client](#) ne définit pas de politique d'autorisation du périphérique client autorisant le périphérique client à se connecter.

Vérifiez que la configuration du composant d'authentification de l'appareil client inclut les éléments suivants :

- Groupe d'appareils correspondant à l'appareil client.
- Une politique d'autorisation du dispositif client pour ce groupe d'appareils qui accorde `mqtt:connect` au dispositif client.

Pour plus d'informations sur le déploiement et la configuration du composant d'authentification de l'appareil client, consultez les rubriques suivantes :

- [Configuration de la découverte du cloud \(console\)](#)
- [Authentification de l'appareil client](#)
- [Créer des déploiements](#)

Problèmes de connexion MQTT () JavaScript

Utilisez les informations suivantes pour résoudre les problèmes liés aux connexions MQTT des appareils clients lorsque vous utilisez la [Kit SDK des appareils AWS IoT version 2](#) pour JavaScript.

Rubriques

- [AWS_ERROR_MQTT_PROTOCOL_ERROR: Protocol error occurred](#)
- [AWS_ERROR_MQTT_UNEXPECTED_HANGUP: Unexpected hangup occurred](#)

AWS_ERROR_MQTT_PROTOCOL_ERROR: Protocol error occurred

Cette erreur peut s'afficher si le [composant d'authentification du périphérique client](#) ne définit pas de politique d'autorisation du périphérique client autorisant le périphérique client à se connecter.

Vérifiez que la configuration du composant d'authentification de l'appareil client inclut les éléments suivants :

- Groupe d'appareils correspondant à l'appareil client.
- Une politique d'autorisation du dispositif client pour ce groupe d'appareils qui accorde `mqtt:connect` autorisation au dispositif client.

Pour plus d'informations sur le déploiement et la configuration du composant d'authentification de l'appareil client, consultez les rubriques suivantes :

- [Configuration de la découverte du cloud \(console\)](#)
- [Authentification de l'appareil client](#)
- [Créer des déploiements](#)

AWS_ERROR_MQTT_UNEXPECTED_HANGUP: Unexpected hangup occurred

Cette erreur peut s'afficher si le [composant d'authentification du périphérique client](#) ne définit pas de politique d'autorisation du périphérique client autorisant le périphérique client à se connecter.

Vérifiez que la configuration du composant d'authentification de l'appareil client inclut les éléments suivants :

- Groupe d'appareils correspondant à l'appareil client.
- Une politique d'autorisation du dispositif client pour ce groupe d'appareils qui accorde `mqtt:connect` autorisation au dispositif client.

Pour plus d'informations sur le déploiement et la configuration du composant d'authentification de l'appareil client, consultez les rubriques suivantes :

- [Configuration de la découverte du cloud \(console\)](#)
- [Authentification de l'appareil client](#)
- [Créer des déploiements](#)

Interagissez avec les ombres de l'appareil

Les appareils Greengrass Core peuvent interagir avec leurs [ombres àAWS IoT](#) l'aide de composants. Un shadow est un document JSON qui stocke les informations d'état actuelles ou souhaitées pour un AWS IoT objet. Les ombres peuvent rendre l'état d'un appareil accessible à d'autres AWS IoT Greengrass composants, que l'appareil soit connecté AWS IoT ou non. Chaque AWS IoT appareil possède sa propre ombre classique et anonyme. Vous pouvez également créer plusieurs ombres nommées pour chaque appareil.

[Les appareils et les services peuvent créer, mettre à jour et supprimer des ombres dans les nuages à l'aide du MQTT et des sujets d'ombre MQTT réservés, du protocole HTTP à l'aide de l'API REST Device Shadow et du AWS CLI for. AWS IoT](#)

Le composant [Shadow Manager](#) permet à vos composants Greengrass de créer, de mettre à jour et de supprimer des ombres locales en utilisant le [service d'ombre local](#) et les rubriques parallèles de publication/d'abonnement locales. Le gestionnaire des ombres gère également le stockage de ces documents instantanés locaux sur votre appareil principal et gère la synchronisation des informations relatives à l'état des ombres avec les ombres des nuages.

Vous pouvez également utiliser le composant Shadow Manager pour gérer les ombres locales pour les [appareils clients](#) qui se connectent au périphérique principal. Pour permettre à Shadow Manager de gérer les ombres des périphériques clients, vous configurez le [composant du pont MQTT](#) pour relayer les messages entre le courtier MQTT local et le service de publication/d'abonnement local. Pour de plus amples informations, veuillez consulter [Interagissez avec les ombres des appareils clients et synchronisez-les](#).

Pour plus d'informations sur les concepts de AWS IoT Device Shadow, consultez le [service AWS IoT Device Shadow](#) dans le Guide du AWS IoT développeur.

Rubriques

- [Interagir avec les ombres dans les composants](#)
- [Synchronisez les ombres de l'appareil local avec AWS IoT Core](#)

Interagir avec les ombres dans les composants

Vous pouvez développer des composants personnalisés, notamment des composants de fonction Lambda, qui utilisent le service fantôme local pour lire et modifier les documents fantômes locaux et les documents fantômes du périphérique client.

Les composants personnalisés interagissent avec le service parallèle local à l'aide des bibliothèques AWS IoT Greengrass Core IPC du Kit SDK des appareils AWS IoT. Le composant [Shadow Manager](#) active le service fantôme local sur votre appareil principal.

Pour déployer le composant Shadow Manager sur un appareil principal de Greengrass, [créez un déploiement](#) incluant le `aws.greengrass.ShadowManager` composant.

Note

Par défaut, le déploiement du composant Shadow Manager active uniquement les opérations d'ombre locales. AWS IoT Greengrass Pour permettre de synchroniser les informations sur l'état des ombres des périphériques principaux ou des ombres des appareils clients avec les documents Cloud Shadow correspondants dans AWS IoT Core, vous devez créer une mise à jour de configuration pour le composant Shadow Manager qui inclut le `synchronize` paramètre. Pour de plus amples informations, veuillez consulter [Synchronisez les ombres de l'appareil local avec AWS IoT Core](#).

Rubriques

- [Récupérez et modifiez les états d'ombre](#)
- [Réagir aux changements d'état de l'ombre](#)

Récupérez et modifiez les états d'ombre

Les opérations IPC parallèles récupèrent et mettent à jour les informations d'état dans les documents fictifs locaux. Le composant Shadow Manager gère le stockage de ces documents fantômes sur votre appareil principal.

Pour modifier les états d'ombre locaux

1. Ajoutez des politiques d'autorisation à la recette de votre composant personnalisé afin de permettre au composant de recevoir des messages sur des sujets secondaires locaux.

Pour des exemples de politiques d'autorisation, voir [Exemples de politiques d'autorisation IPC fantôme locales](#).

2. Utilisez les opérations IPC de l'ombre pour récupérer et modifier les informations relatives à l'état de l'ombre. Pour plus d'informations sur l'utilisation des opérations IPC fantômes dans le code des composants, consultez [Interagissez avec les ombres locales](#).

Note

Pour permettre à un périphérique principal d'interagir avec les ombres du périphérique client, vous devez également configurer et déployer le composant de pont MQTT. Pour plus d'informations, voir [Activer le Shadow Manager pour communiquer avec les appareils clients](#).

Réagir aux changements d'état de l'ombre

Les composants Greengrass utilisent les rubriques locales `publish/subscribe` interface to communicate on a core device. To enable a custom component to react to shadow state changes, you can subscribe to the local `publish/subscribe`. Cela permet au composant de recevoir des messages sur les sujets secondaires locaux, puis d'agir sur ces messages.

Les sujets fantômes locaux utilisent le même format que les sujets MQTT fantômes de l' AWS IoT appareil. Pour plus d'informations sur les sujets cachés, consultez les [rubriques Device Shadow MQTT](#) dans le manuel du AWS IoT développeur.

Pour réagir aux changements de l'état parallèle local

1. Ajoutez des politiques de contrôle d'accès à la recette de votre composant personnalisé afin de permettre au composant de recevoir des messages sur des sujets secondaires locaux.

Pour des exemples de politiques d'autorisation, voir [Exemples de politiques d'autorisation IPC fantôme locales](#).

2. Pour lancer une action personnalisée dans un composant, utilisez les opérations `SubscribeToTopic` IPC pour vous abonner aux sujets fictifs sur lesquels vous souhaitez recevoir des messages. Pour plus d'informations sur l'utilisation des opérations IPC de publication/d'abonnement locales dans le code des composants, consultez. [Publier/souscrire des messages locaux](#)

3. Pour appeler une fonction Lambda, utilisez la configuration de la source d'événements pour fournir le nom du sujet parallèle et spécifiez qu'il s'agit d'un sujet de publication/d'abonnement local. Pour plus d'informations sur la création de composants de fonctions Lambda, consultez [Exécuter AWS Lambda des fonctions](#)

Note

Pour permettre à un périphérique principal d'interagir avec les ombres du périphérique client, vous devez également configurer et déployer le composant de pont MQTT. Pour plus d'informations, voir [Activer le Shadow Manager pour communiquer avec les appareils clients](#).

Synchronisez les ombres de l'appareil local avec AWS IoT Core

Le composant Shadow Manager permet AWS IoT Greengrass de synchroniser les états d'ombre des périphériques locaux avec AWS IoT Core. Vous devez modifier la configuration du composant Shadow Manager pour inclure le paramètre de `synchronization configuration` et spécifier les noms AWS IoT des objets pour vos appareils, ainsi que les ombres que vous souhaitez synchroniser.

Lorsque vous configurez le gestionnaire d'ombres pour synchroniser les ombres, il synchronise tous les changements d'état pour les ombres spécifiées, que les modifications se produisent dans les documents d'ombre locaux ou dans les documents d'ombre du cloud.

Vous pouvez également spécifier si le composant Shadow Manager synchronise les ombres en temps réel ou à intervalles réguliers. Par défaut, le composant Shadow Manager synchronise les ombres en temps réel, de sorte que le périphérique principal envoie et reçoit des mises à jour des ombres depuis AWS IoT Core et vers chaque mise à jour. Vous pouvez configurer des intervalles périodiques pour réduire l'utilisation de la bande passante et les frais.

Rubriques

- [Prérequis](#)
- [Configuration du composant Shadow Manager](#)
- [Synchroniser les ombres locales](#)
- [Comportement des conflits liés à la fusion](#)

Prérequis

Pour synchroniser les ombres locales avec AWS IoT Core, vous devez configurer la AWS IoT politique de l'appareil principal de Greengrass afin d'autoriser les actions de politique AWS IoT Core parallèle suivantes.

- `iot:GetThingShadow`
- `iot:UpdateThingShadow`
- `iot:DeleteThingShadow`

Pour plus d'informations, consultez les ressources suivantes :

- [AWS IoT Core actions politiques décrites](#) dans le guide du AWS IoT développeur
- [AWS IoT Politique minimale pour les appareils AWS IoT Greengrass V2 principaux](#)
- [Mettre à jour la AWS IoT politique d'un appareil principal](#)

Configuration du composant Shadow Manager

Le gestionnaire de fantômes a besoin d'une liste de mappages de noms de fantômes pour synchroniser les informations relatives à l'état des ombres contenues dans les documents d'ombre locaux avec les documents d'ombre dans AWS IoT Core le cloud.

Pour synchroniser les états des ombres, [créez un déploiement](#) incluant le `aws.greengrass.ShadowManager` composant et spécifiez les ombres que vous souhaitez synchroniser dans le paramètre de `synchronize` configuration de la configuration du gestionnaire des ombres du déploiement.

Note

Pour permettre à un périphérique principal d'interagir avec les ombres du périphérique client, vous devez également configurer et déployer le composant de pont MQTT. Pour plus d'informations, voir [Activer le Shadow Manager pour communiquer avec les appareils clients](#).

L'exemple de mise à jour de configuration suivant indique au composant Shadow Manager de synchroniser les ombres suivantes avec AWS IoT Core :

- L'ombre classique de l'appareil principal
- Le nom MyCoreShadow de l'appareil principal
- L'ombre classique d'un objet IoT nommé MyDevice2
- Les ombres nommées MyShadowA et MyShadowB pour un objet IoT nommé MyDevice1

Cette mise à jour de configuration indique de synchroniser les ombres avec AWS IoT Core en temps réel. Si vous utilisez Shadow Manager v2.1.0 ou version ultérieure, vous pouvez configurer le composant Shadow Manager pour synchroniser les ombres à intervalles réguliers. Pour configurer cette fonctionnalité, remplacez la stratégie de `periodic` synchronisation par et spécifiez un intervalle `delay` en secondes. Pour plus d'informations, consultez [le paramètre de configuration de stratégie](#) du composant Shadow Manager.

Cette mise à jour de configuration indique de synchroniser les ombres dans les deux sens entre le périphérique principal AWS IoT Core et le périphérique principal. Si vous utilisez Shadow Manager v2.2.0 ou version ultérieure, vous pouvez configurer le composant Shadow Manager pour synchroniser les ombres dans une seule direction. Pour configurer cette fonctionnalité, remplacez la synchronisation `direction` par `deviceToCloud` ou `cloudToDevice`. Pour plus d'informations, consultez [le paramètre de configuration de direction](#) du composant Shadow Manager.

```
{
  "strategy": {
    "type": "realTime"
  },
  "synchronize": {
    "coreThing": {
      "classic": true,
      "namedShadows": [
        "MyCoreShadow"
      ]
    },
  },
  "shadowDocuments": [
    {
      "thingName": "MyDevice1",
      "classic": false,
      "namedShadows": [
        "MyShadowA",
        "MyShadowB"
      ]
    }
  ],
}
```



```
{
  "thingName": "MyDevice2",
  "classic": true,
  "namedShadows": [ ]
},
"direction": "betweenDeviceAndCloud"
}
```

Synchroniser les ombres locales

Lorsque le périphérique principal de Greengrass est connecté au AWS IoT cloud, le gestionnaire d'ombres exécute les tâches suivantes pour les ombres que vous spécifiez dans la configuration des composants. Le comportement dépend de l'option de configuration de la direction de synchronisation des ombres que vous spécifiez. Par défaut, le gestionnaire d'ombres utilise l'`betweenDeviceAndCloud` option permettant de synchroniser les ombres dans les deux directions. Si vous utilisez Shadow Manager v2.2.0 ou version ultérieure, vous pouvez configurer le périphérique principal pour synchroniser les ombres dans une seule direction, qui peut être `cloudToDevice` ou `deviceToCloud`.

- Si la configuration de la direction de synchronisation des ombres est `betweenDeviceAndCloud` ou `cloudToDevice`, Shadow Manager récupère les informations d'état signalées à partir du document Cloud Shadow dans AWS IoT Core. Il met ensuite à jour les documents fictifs stockés localement pour synchroniser l'état de l'appareil.
- Si la configuration de la direction de synchronisation des ombres est `betweenDeviceAndCloud` ou `deviceToCloud`, Shadow Manager publie l'état actuel de l'appareil dans le document Cloud Shadow.

Comportement des conflits liés à la fusion

Dans certains cas, par exemple lorsque le périphérique principal est déconnecté d'Internet, une ombre peut changer dans le service parallèle local et dans le AWS IoT cloud avant que le gestionnaire parallèle ne synchronise les modifications. Par conséquent, les états souhaités et signalés diffèrent entre le service parallèle local et le AWS IoT cloud.

Lorsque le gestionnaire d'ombres synchronise l'ombre, il fusionne les modifications selon le comportement suivant :

- Si vous utilisez une version du gestionnaire d'ombres antérieure à la version 2.2.0, ou lorsque vous spécifiez la direction de synchronisation des `betweenDeviceAndCloud` ombres, le comportement suivant s'applique :
 - En cas de conflit de fusion dans l'état souhaité pour une ombre, le gestionnaire des ombres remplace la section en conflit du document parallèle local par la valeur provenant du AWS IoT cloud.
 - En cas de conflit de fusion dans l'état signalé d'une ombre, le gestionnaire des ombres remplace la section conflictuelle de l'ombre dans le AWS IoT cloud par la valeur du document parallèle local.
- Lorsque vous spécifiez la `deviceToCloud` direction de synchronisation des ombres, le gestionnaire d'ombres remplace la section conflictuelle de l'ombre dans le AWS IoT cloud par la valeur du document d'ombre local.
- Lorsque vous spécifiez la `cloudToDevice` direction de synchronisation des ombres, le gestionnaire d'ombres remplace la section en conflit du document d'ombre local par la valeur provenant du AWS IoT cloud.

Gérez les flux de données sur les appareils principaux de Greengrass

AWS IoT Greengrass le gestionnaire de flux permet de transférer de gros volumes de données IoT de manière plus efficace et plus fiable vers le AWS Cloud. Le gestionnaire de flux traite les flux de données sur le AWS IoT Greengrass Core avant de les exporter vers le AWS Cloud. Le gestionnaire de flux s'intègre aux scénarios périphériques courants, tels que l'inférence d'apprentissage automatique (ML), dans laquelle le périphérique AWS IoT Greengrass Core traite et analyse les données avant de les exporter vers des destinations de stockage locales AWS Cloud ou vers des destinations de stockage locales.

Le gestionnaire de flux fournit une interface commune pour simplifier le développement de composants personnalisés afin que vous n'ayez pas à créer de fonctionnalités de gestion de flux personnalisées. Vos composants peuvent utiliser un mécanisme standardisé pour traiter des flux importants et gérer les politiques locales de conservation des données. Vous pouvez définir des politiques relatives au type de stockage, à la taille et à la conservation des données pour chaque flux afin de contrôler la manière dont le gestionnaire de flux traite et exporte les données.

Le gestionnaire de flux fonctionne dans des environnements dotés d'une connectivité intermittente ou limitée. Vous pouvez définir l'utilisation de la bande passante, le comportement du délai d'expiration et la manière dont le AWS IoT Greengrass Core gère les données de flux lorsqu'il est connecté ou déconnecté. Vous pouvez également définir des priorités pour contrôler l'ordre dans lequel le AWS IoT Greengrass Core exporte les flux vers le AWS Cloud. Cela vous permet de traiter les données critiques plus rapidement que les autres données.

Vous pouvez configurer le gestionnaire de flux pour qu'il exporte automatiquement les données vers le AWS Cloud pour les stocker ou les traiter et les analyser ultérieurement. Le gestionnaire de flux prend en charge les exportations vers les AWS Cloud destinations suivantes :

- Canaux entrants AWS IoT Analytics. AWS IoT Analytics vous permet d'effectuer une analyse avancée de vos données afin de prendre des décisions commerciales et d'améliorer les modèles d'apprentissage automatique. Pour plus d'informations, consultez [Présentation d' AWS IoT Analytics](#) dans le Guide de l'utilisateur AWS IoT Analytics .
- Streams dans Amazon Kinesis Data Streams. Vous pouvez utiliser Kinesis Data Streams pour agréger de gros volumes de données et les charger dans un entrepôt MapReduce de données ou un cluster. Pour plus d'informations, consultez [Présentation d'Amazon Kinesis Data Streams](#) dans le Guide du développeur Amazon Kinesis Data Streams.

- Propriétés des actifs dans AWS IoT SiteWise. AWS IoT SiteWise vous permet de collecter, d'organiser et d'analyser les données des équipements industriels à grande échelle. Pour plus d'informations, voir [Qu'est-ce que c'est AWS IoT SiteWise ?](#) dans le guide de AWS IoT SiteWise l'utilisateur.
- Objets dans Amazon Simple Storage Service (Amazon S3). Vous pouvez utiliser Amazon S3 pour stocker et récupérer de grandes quantités de données. Pour plus d'informations, consultez [Qu'est-ce qu'Amazon S3 ?](#) dans le guide du développeur d'Amazon Simple Storage Service.

Flux de travail de gestion des flux

Vos applications IoT interagissent avec le gestionnaire de flux via le SDK Stream Manager.

Dans un flux de travail simple, un composant AWS IoT Greengrass central consomme des données IoT, telles que des séries chronologiques de mesures de température et de pression. Le composant peut filtrer ou compresser les données, puis appeler le SDK Stream Manager pour écrire les données dans un flux dans le gestionnaire de flux. Le gestionnaire de flux peut exporter le flux vers le AWS Cloud mode automatique en fonction des politiques que vous définissez pour le flux. Les composants peuvent également envoyer des données directement vers des bases de données locales ou des référentiels de stockage.

Vos applications IoT peuvent inclure plusieurs composants personnalisés qui lisent ou écrivent dans des flux. Ces composants peuvent lire et écrire dans des flux pour filtrer, agréger et analyser les données sur le périphérique AWS IoT Greengrass principal. Cela permet de réagir rapidement aux événements locaux et d'extraire des informations précieuses avant que les données ne soient transférées du centre vers les AWS Cloud destinations locales.

Pour commencer, déployez le composant Stream Manager sur votre appareil AWS IoT Greengrass principal. Lors du déploiement, configurez les paramètres du composant du gestionnaire de flux pour définir les paramètres qui s'appliquent à tous les flux sur le périphérique principal Greengrass. Utilisez ces paramètres pour contrôler la manière dont le gestionnaire de flux stocke, traite et exporte les flux en fonction des besoins de votre entreprise et des contraintes environnementales.

Après avoir configuré le gestionnaire de flux, vous pouvez créer et déployer vos applications IoT. Il s'agit généralement de composants personnalisés utilisés `StreamManagerClient` dans le SDK Stream Manager pour créer des flux et interagir avec ceux-ci. Lorsque vous créez un flux, vous pouvez définir des politiques par flux, telles que les destinations d'exportation, la priorité et la persistance.

Prérequis

Les exigences suivantes s'appliquent à l'utilisation du gestionnaire de flux :

- Le gestionnaire de flux nécessite un minimum de 70 Mo de RAM en plus du logiciel AWS IoT Greengrass Core. Vos besoins totaux en mémoire dépendent de votre charge de travail.
- AWS IoT Greengrass les composants doivent utiliser le SDK Stream Manager pour interagir avec le Stream Manager. Le SDK Stream Manager est disponible dans les langues suivantes :
 - [SDK Stream Manager pour Java](#) (v1.1.0 ou version ultérieure)
 - [SDK Stream Manager pour Node.js](#) (v1.1.0 ou version ultérieure)
 - [SDK Stream Manager pour Python](#) (v1.1.0 ou version ultérieure)
- AWS IoT Greengrass les composants doivent spécifier le composant du gestionnaire de flux (`aws.greengrass.StreamManager`) en tant que dépendance dans leur recette pour utiliser le gestionnaire de flux.

Note

Si vous utilisez le gestionnaire de flux pour exporter des données vers le cloud, vous ne pouvez pas mettre à niveau la version 2.0.7 du composant du gestionnaire de flux vers une version comprise entre v2.0.8 et v2.0.11. Si vous déployez le gestionnaire de flux pour la première fois, nous vous recommandons vivement de déployer la dernière version du composant du gestionnaire de flux.

- Si vous définissez des destinations AWS Cloud d'exportation pour un flux, vous devez créer vos cibles d'exportation et accorder des autorisations d'accès dans le rôle d'[appareil Greengrass](#). Selon la destination, d'autres exigences peuvent également s'appliquer. Pour plus d'informations, consultez :
 - [the section called "AWS IoT Analytics chaînes"](#)
 - [the section called "Flux de données Amazon Kinesis"](#)
 - [the section called "AWS IoT SiteWise propriétés des actifs"](#)
 - [the section called "Objets Amazon S3"](#)

Vous êtes responsable de la maintenance de ces AWS Cloud ressources.

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 entre les composants locaux du périphérique principal.

- **Données au repos.** Les données de flux sont stockées localement dans un répertoire de stockage. Pour la sécurité des données, AWS IoT Greengrass repose sur les autorisations de fichiers et le chiffrement complet du disque, s'il est activé. 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 autre répertoire de stockage, AWS IoT Greengrass cela ne supprime pas le répertoire de stockage précédent ni son contenu.
- **Données en transit localement.** AWS IoT Greengrass ne chiffre pas les données de flux en transit local entre les sources de données, AWS IoT Greengrass les composants, le SDK Stream Manager et le gestionnaire de flux.
- **Données en transit vers le AWS Cloud.** Les flux de données exportés par le gestionnaire de flux AWS Cloud utilisent le chiffrement standard du client de AWS service avec le protocole TLS (Transport Layer Security).

Authentification client

Les clients Stream Manager utilisent le SDK Stream Manager pour communiquer avec Stream Manager. Lorsque l'authentification du client est activée, seuls les composants Greengrass peuvent interagir avec les flux dans le gestionnaire de flux. Lorsque l'authentification du client est désactivée, tout processus exécuté sur le périphérique principal de Greengrass peut interagir avec les flux dans le 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 lorsque vous déployez le composant Stream Manager sur les appareils principaux.

	Activé	Désactivées
Valeur de paramètre	true (valeur par défaut et recommandée)	false
Clients autorisés	Composants Greengrass sur l'appareil principal	Composants Greengrass sur l'appareil principal Autres processus en cours d'exécution sur l'appareil principal Greengrass

Consultez aussi

- [the section called “Configuration du gestionnaire de flux”](#)
- [the section called “StreamManagerClient À utiliser pour travailler avec des flux”](#)
- [the section called “Exporter les configurations pour les destinations cloud prises en charge”](#)

Configuration du gestionnaire de AWS IoT Greengrass flux

Sur les appareils principaux de Greengrass, le gestionnaire de flux peut stocker, traiter et exporter les données des appareils IoT. Le gestionnaire de flux fournit des paramètres que vous pouvez utiliser pour configurer les paramètres d'exécution. Ces paramètres s'appliquent à tous les streams sur l'appareil principal de Greengrass. Vous pouvez utiliser la AWS IoT Greengrass console ou l'API pour configurer les paramètres du gestionnaire de flux lorsque vous déployez le composant. Les modifications prennent effet une fois le déploiement terminé.

Paramètres du gestionnaire de flux

Le gestionnaire de flux fournit les paramètres suivants que vous pouvez configurer lorsque vous déployez le composant sur vos appareils principaux. Tous les paramètres sont facultatifs.

Répertoire de stockage

Nom du paramètre: `STREAM_MANAGER_STORE_ROOT_DIR`

Le chemin absolu du dossier local utilisé pour stocker les flux. Cette valeur doit commencer par une barre oblique (par exemple, `/data`).

Vous devez spécifier un dossier existant, et l'[utilisateur du système qui exécute le composant du gestionnaire de flux](#) doit être autorisé à lire et à écrire dans ce dossier. Par exemple, vous pouvez exécuter les commandes suivantes pour créer et configurer un dossier `/var/greengrass/streams`, que vous spécifiez comme dossier racine du gestionnaire de flux. Ces commandes permettent à l'utilisateur du système par défaut de lire et d'écrire dans ce dossier. `ggc_user`

```
sudo mkdir /var/greengrass/streams
sudo chown ggc_user /var/greengrass/streams
sudo chmod 700 /var/greengrass/streams
```

Pour de plus amples informations sur la sécurité des données de flux, veuillez consulter [the section called “Sécurité des données locales”](#).

Par défaut : `/greengrass/v2/work/aws.greengrass.StreamManager`

Server port

Nom du paramètre: `STREAM_MANAGER_SERVER_PORT`

Numéro de port local utilisé pour communiquer avec le gestionnaire de flux. L'argument par défaut est 8088.

Vous pouvez spécifier 0 d'utiliser un port disponible de manière aléatoire.

Authentification du client

Nom du paramètre: `STREAM_MANAGER_AUTHENTICATE_CLIENT`

Indique si les clients doivent être authentifiés pour interagir avec le gestionnaire de flux. Toutes les interactions entre les clients et le gestionnaire de flux sont contrôlées par le SDK Stream Manager. Ce paramètre détermine quels clients peuvent appeler le SDK Stream Manager pour travailler avec des flux. Pour de plus amples informations, veuillez consulter [the section called “Authentification client”](#).

Les valeurs valides sont `true` ou `false`. La valeur par défaut est `true` (recommandé).

- `true`. Autorise uniquement les composants Greengrass en tant que clients. Les composants utilisent des protocoles AWS IoT Greengrass Core internes pour s'authentifier auprès du SDK Stream Manager.

- `false`. Permet à tout processus exécuté sur le AWS IoT Greengrass Core d'être un client. Ne définissez pas la valeur sur `false` sauf si votre analyse de rentabilisation l'exige. Par exemple, à utiliser `false` uniquement si les processus non composants du périphérique principal doivent communiquer directement avec le gestionnaire de flux.

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.

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. L'argument 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. Essayez donc de limiter le nombre de flux d'exportation sur les appareils soumis à des contraintes.

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.

Logging level (Niveau de journalisation)

Nom du paramètre: `LOG_LEVEL`

Le niveau de journalisation du composant. Choisissez parmi les niveaux de journalisation suivants, listés ici par ordre de niveau :

- TRACE
- DEBUG
- INFO
- WARN
- ERROR

Par défaut : INFO

Taille minimale pour le téléchargement en plusieurs parties

Nom du paramètre:

`STREAM_MANAGER_EXPORTER_S3_DESTINATION_MULTIPART_UPLOAD_MIN_PART_SIZE_BYTES`

Taille minimale (en octets) d'une partie dans 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 en plusieurs parties. La valeur minimale par défaut est de 5 5242880 octets (5 Mo).

 Note

Le gestionnaire de flux utilise la `sizeThresholdForMultipartUploadBytes` propriété du flux pour déterminer s'il convient d'exporter vers Amazon S3 sous forme de téléchargement en une ou plusieurs parties. Les composants Greengrass définis par l'utilisateur définissent ce seuil lorsqu'ils créent un flux exporté vers Amazon S3. Le seuil par défaut est de 5 Mo.

Consultez aussi

- [Gérez les flux de données sur les appareils principaux de Greengrass](#)
- [StreamManagerClient À utiliser pour travailler avec des flux](#)
- [Exporter les configurations pour les AWS Cloud destinations prises en charge](#)

Créez des composants personnalisés utilisant le gestionnaire de flux

Utilisez le gestionnaire de flux dans les composants Greengrass personnalisés pour stocker, traiter et exporter les données des appareils IoT. Utilisez les procédures et les exemples de cette section pour créer des recettes de composants, des artefacts et des applications compatibles avec le gestionnaire de flux. Pour plus d'informations sur le développement et le test de composants, consultez [Création de AWS IoT Greengrass composants](#).

Rubriques

- [Définissez des recettes de composants utilisant le gestionnaire de flux](#)
- [Connectez-vous au gestionnaire de flux dans le code de l'application](#)

Définissez des recettes de composants utilisant le gestionnaire de flux

Pour utiliser le gestionnaire de flux dans un composant personnalisé, vous devez définir le `aws.greengrass.StreamManager` composant en tant que dépendance. Vous devez également fournir le SDK Stream Manager. Effectuez les tâches suivantes pour télécharger et utiliser le SDK Stream Manager dans la langue de votre choix.

Utiliser le SDK Stream Manager pour Java

Le SDK Stream Manager pour Java est disponible sous forme de fichier JAR que vous pouvez utiliser pour compiler votre composant. Vous pouvez ensuite créer un fichier JAR d'application qui inclut le SDK Stream Manager, définir le fichier JAR d'application en tant qu'artefact de composant et exécuter le fichier JAR d'application dans le cycle de vie du composant.

Pour utiliser le SDK Stream Manager pour Java

1. Téléchargez le fichier [JAR du SDK Stream Manager pour Java](#).
2. Procédez de l'une des manières suivantes pour créer des artefacts de composants à partir de votre application Java et du fichier JAR du SDK Stream Manager :
 - Créez votre application sous la forme d'un fichier JAR incluant le JAR du SDK Stream Manager, et exécutez ce fichier JAR dans votre recette de composant.
 - Définissez le JAR du SDK Stream Manager en tant qu'artefact de composant. Ajoutez cet artefact au chemin de classe lorsque vous exécutez votre application dans votre recette de composant.

La recette de vos composants peut ressembler à l'exemple suivant. Ce composant exécute une version modifiée de l'exemple [StreamManagerS3.java](#), qui StreamManagerS3.jar inclut le JAR du SDK Stream Manager.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.StreamManagerS3Java",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "Uses stream manager to upload a file to an S3
bucket.",
  "ComponentPublisher": "Amazon",
  "ComponentDependencies": {
    "aws.greengrass.StreamManager": {
      "VersionRequirement": "^2.0.0"
    }
  },
  "Manifests": [
    {
      "Lifecycle": {
        "Run": "java -jar {artifacts:path}/StreamManagerS3.jar"
      },
      "Artifacts": [
        {
          "URI": "s3://amzn-s3-demo-bucket/artifacts/
com.example.StreamManagerS3Java/1.0.0/StreamManagerS3.jar"
        }
      ]
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.StreamManagerS3Java
ComponentVersion: 1.0.0
ComponentDescription: Uses stream manager to upload a file to an S3 bucket.
ComponentPublisher: Amazon
ComponentDependencies:
```

```
aws.greengrass.StreamManager:
  VersionRequirement: "^2.0.0"
Manifests:
- Lifecycle:
  Run: java -jar {artifacts:path}/StreamManagerS3.jar
  Artifacts:
  - URI: s3://amzn-s3-demo-bucket/artifacts/
    com.example.StreamManagerS3Java/1.0.0/StreamManagerS3.jar
```

Pour plus d'informations sur le développement et le test de composants, consultez [Création de AWS IoT Greengrass composants](#).

Utiliser le SDK Stream Manager pour Python

Le SDK Stream Manager pour Python est disponible sous forme de code source que vous pouvez inclure dans votre composant. Créez un fichier ZIP du SDK Stream Manager, définissez le fichier ZIP en tant qu'artefact de composant et installez les exigences du SDK dans le cycle de vie du composant.

Pour utiliser le SDK Stream Manager pour Python

1. Clonez ou téléchargez le dépôt [aws-greengrass-stream-manager-sdk-python](#).

```
git clone git@github.com:aws-greengrass/aws-greengrass-stream-manager-sdk-python.git
```

2. Créez un fichier ZIP contenant le `stream_manager` dossier contenant le code source du SDK Stream Manager pour Python. Vous pouvez fournir ce fichier ZIP sous forme d'artefact de composant que le logiciel AWS IoT Greengrass Core décompresse lors de l'installation de votre composant. Procédez comme suit :
 - a. Ouvrez le dossier contenant le référentiel que vous avez cloné ou téléchargé à l'étape précédente.

```
cd aws-greengrass-stream-manager-sdk-python
```

- b. Comprimez le `stream_manager` dossier dans un fichier ZIP nommé `stream_manager_sdk.zip`.

Linux or Unix

```
zip -rv stream_manager_sdk.zip stream_manager
```

Windows Command Prompt (CMD)

```
tar -acvf stream_manager_sdk.zip stream_manager
```

PowerShell

```
Compress-Archive stream_manager stream_manager_sdk.zip
```

- c. Vérifiez que le `stream_manager_sdk.zip` fichier contient le `stream_manager` dossier et son contenu. Exécutez la commande suivante pour répertorier le contenu du fichier ZIP.

Linux or Unix

```
unzip -l stream_manager_sdk.zip
```

Windows Command Prompt (CMD)

```
tar -tf stream_manager_sdk.zip
```

La sortie doit ressembler à ce qui suit :

```
Archive:  aws-greengrass-stream-manager-sdk-python/stream_manager.zip
 Length   Date       Time       Name
-----
      0   02-24-2021  20:45   stream_manager/
    913   02-24-2021  20:45   stream_manager/__init__.py
   9719   02-24-2021  20:45   stream_manager/utilinternal.py
   1412   02-24-2021  20:45   stream_manager/exceptions.py
   1004   02-24-2021  20:45   stream_manager/util.py
      0   02-24-2021  20:45   stream_manager/data/
  254463   02-24-2021  20:45   stream_manager/data/__init__.py
   26515   02-24-2021  20:45   stream_manager/streammanagerclient.py
-----
  294026                      8 files
```

3. Copiez les artefacts du SDK Stream Manager dans le dossier des artefacts de votre composant. Outre le fichier ZIP du SDK Stream Manager, votre composant utilise le `requirements.txt` fichier du SDK pour installer les dépendances du SDK Stream Manager. `~/greengrass-components` Remplacez-le par le chemin du dossier que vous utilisez pour le développement local.

Linux or Unix

```
cp {stream_manager_sdk.zip,requirements.txt} ~/greengrass-components/artifacts/
com.example.StreamManagerS3Python/1.0.0/
```

Windows Command Prompt (CMD)

```
robocopy . %USERPROFILE%\greengrass-components\artifacts
\com.example.StreamManagerS3Python\1.0.0 stream_manager_sdk.zip
robocopy . %USERPROFILE%\greengrass-components\artifacts
\com.example.StreamManagerS3Python\1.0.0 requirements.txt
```

PowerShell

```
cp .\stream_manager_sdk.zip,.\requirements.txt ~\greengrass-components\artifacts
\com.example.StreamManagerS3Python\1.0.0\
```

4. Créez votre recette de composants. Dans la recette, procédez comme suit :
 - a. Définissez `stream_manager_sdk.zip` et `requirements.txt` en tant qu'artefacts.
 - b. Définissez votre application Python comme un artefact.
 - c. Au cours du cycle de vie d'installation, installez les exigences du SDK Stream Manager à partir de `requirements.txt`.
 - d. Au cours du cycle de vie d'exécution, ajoutez le SDK Stream Manager à votre `PYTHONPATH` application Python et exécutez-la.

La recette de vos composants peut ressembler à l'exemple suivant. Ce composant exécute l'exemple [stream_manager_s3.py](#).

JSON

```
{
```

```

"RecipeFormatVersion": "2020-01-25",
"ComponentName": "com.example.StreamManagerS3Python",
"ComponentVersion": "1.0.0",
"ComponentDescription": "Uses stream manager to upload a file to an S3
bucket.",
"ComponentPublisher": "Amazon",
"ComponentDependencies": {
  "aws.greengrass.StreamManager": {
    "VersionRequirement": "^2.0.0"
  }
},
"Manifests": [
  {
    "Platform": {
      "os": "linux"
    },
    "Lifecycle": {
      "install": "pip3 install --user -r {artifacts:path}/requirements.txt",
      "Run": "export PYTHONPATH=$PYTHONPATH:{artifacts:decompressedPath}/
stream_manager_sdk; python3 {artifacts:path}/stream_manager_s3.py"
    },
    "Artifacts": [
      {
        "URI": "s3://amzn-s3-demo-bucket/artifacts/
com.example.StreamManagerS3Python/1.0.0/stream_manager_sdk.zip",
        "Unarchive": "ZIP"
      },
      {
        "URI": "s3://amzn-s3-demo-bucket/artifacts/
com.example.StreamManagerS3Python/1.0.0/stream_manager_s3.py"
      },
      {
        "URI": "s3://amzn-s3-demo-bucket/artifacts/
com.example.StreamManagerS3Python/1.0.0/requirements.txt"
      }
    ]
  },
  {
    "Platform": {
      "os": "windows"
    },
    "Lifecycle": {
      "install": "pip3 install --user -r {artifacts:path}/requirements.txt",

```



```

    "Run": "set \"PYTHONPATH=%PYTHONPATH%;{artifacts:decompressedPath}/
stream_manager_sdk\" & py -3 {artifacts:path}/stream_manager_s3.py"
  },
  "Artifacts": [
    {
      "URI": "s3://amzn-s3-demo-bucket/artifacts/
com.example.StreamManagerS3Python/1.0.0/stream_manager_sdk.zip",
      "Unarchive": "ZIP"
    },
    {
      "URI": "s3://amzn-s3-demo-bucket/artifacts/
com.example.StreamManagerS3Python/1.0.0/stream_manager_s3.py"
    },
    {
      "URI": "s3://amzn-s3-demo-bucket/artifacts/
com.example.StreamManagerS3Python/1.0.0/requirements.txt"
    }
  ]
}
]
}
}

```

YAML

```

---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.StreamManagerS3Python
ComponentVersion: 1.0.0
ComponentDescription: Uses stream manager to upload a file to an S3 bucket.
ComponentPublisher: Amazon
ComponentDependencies:
  aws.greengrass.StreamManager:
    VersionRequirement: "^2.0.0"
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      install: pip3 install --user -r {artifacts:path}/requirements.txt
      Run: |
        export PYTHONPATH=$PYTHONPATH:{artifacts:decompressedPath}/
stream_manager_sdk
        python3 {artifacts:path}/stream_manager_s3.py
    Artifacts:

```

```

- URI: s3://amzn-s3-demo-bucket/artifacts/
com.example.StreamManagerS3Python/1.0.0/stream_manager_sdk.zip
  Unarchive: ZIP
- URI: s3://amzn-s3-demo-bucket/artifacts/
com.example.StreamManagerS3Python/1.0.0/stream_manager_s3.py
- URI: s3://amzn-s3-demo-bucket/artifacts/
com.example.StreamManagerS3Python/1.0.0/requirements.txt
- Platform:
  os: windows
Lifecycle:
  install: pip3 install --user -r {artifacts:path}/requirements.txt
  Run: |
    set "PYTHONPATH=%PYTHONPATH%;{artifacts:decompressedPath}/
stream_manager_sdk"
    py -3 {artifacts:path}/stream_manager_s3.py
Artifacts:
- URI: s3://amzn-s3-demo-bucket/artifacts/
com.example.StreamManagerS3Python/1.0.0/stream_manager_sdk.zip
  Unarchive: ZIP
- URI: s3://amzn-s3-demo-bucket/artifacts/
com.example.StreamManagerS3Python/1.0.0/stream_manager_s3.py
- URI: s3://amzn-s3-demo-bucket/artifacts/
com.example.StreamManagerS3Python/1.0.0/requirements.txt

```

Pour plus d'informations sur le développement et le test de composants, consultez [Création de AWS IoT Greengrass composants](#).

Utilisez le SDK Stream Manager pour JavaScript

Le SDK Stream Manager pour JavaScript est disponible sous forme de code source que vous pouvez inclure dans votre composant. Créez un fichier ZIP du SDK Stream Manager, définissez le fichier ZIP en tant qu'artefact de composant et installez le SDK dans le cycle de vie du composant.

Pour utiliser le SDK Stream Manager pour JavaScript

1. Clonez ou téléchargez le dépôt [aws-greengrass-stream-manager-sdk-js](#).

```
git clone git@github.com:aws-greengrass/aws-greengrass-stream-manager-sdk-js.git
```

2. Créez un fichier ZIP contenant le `aws-greengrass-stream-manager-sdk` dossier contenant le code source du SDK Stream Manager pour JavaScript. Vous pouvez fournir ce fichier ZIP

sous forme d'artefact de composant que le logiciel AWS IoT Greengrass Core décompresse lors de l'installation de votre composant. Procédez comme suit :

- a. Ouvrez le dossier contenant le référentiel que vous avez cloné ou téléchargé à l'étape précédente.

```
cd aws-greengrass-stream-manager-sdk-js
```

- b. Comprimez le `aws-greengrass-stream-manager-sdk` dossier dans un fichier ZIP nommé `stream-manager-sdk.zip`.

Linux or Unix

```
zip -rv stream-manager-sdk.zip aws-greengrass-stream-manager-sdk
```

Windows Command Prompt (CMD)

```
tar -acvf stream-manager-sdk.zip aws-greengrass-stream-manager-sdk
```

PowerShell

```
Compress-Archive aws-greengrass-stream-manager-sdk stream-manager-sdk.zip
```

- c. Vérifiez que le `stream-manager-sdk.zip` fichier contient le `aws-greengrass-stream-manager-sdk` dossier et son contenu. Exécutez la commande suivante pour répertorier le contenu du fichier ZIP.

Linux or Unix

```
unzip -l stream-manager-sdk.zip
```

Windows Command Prompt (CMD)

```
tar -tf stream-manager-sdk.zip
```

La sortie doit ressembler à ce qui suit :

```
Archive:  stream-manager-sdk.zip
 Length   Date   Time    Name
```

```

-----
      0  02-24-2021 22:36  aws-greengrass-stream-manager-sdk/
    369  02-24-2021 22:36  aws-greengrass-stream-manager-sdk/package.json
   1017  02-24-2021 22:36  aws-greengrass-stream-manager-sdk/util.js
   8374  02-24-2021 22:36  aws-greengrass-stream-manager-sdk/utilInternal.js
   1937  02-24-2021 22:36  aws-greengrass-stream-manager-sdk/exceptions.js
      0  02-24-2021 22:36  aws-greengrass-stream-manager-sdk/data/
  353343 02-24-2021 22:36  aws-greengrass-stream-manager-sdk/data/index.js
   22599 02-24-2021 22:36  aws-greengrass-stream-manager-sdk/client.js
     216 02-24-2021 22:36  aws-greengrass-stream-manager-sdk/index.js
-----
  387855                                9 files

```

3. Copiez l'artefact du SDK Stream Manager dans le dossier des artefacts de votre composant.
`~/greengrass-components` Remplacez-le par le chemin du dossier que vous utilisez pour le développement local.

Linux or Unix

```
cp stream-manager-sdk.zip ~/greengrass-components/artifacts/
com.example.StreamManagerS3JS/1.0.0/
```

Windows Command Prompt (CMD)

```
robocopy . %USERPROFILE%\greengrass-components\artifacts
\com.example.StreamManagerS3JS\1.0.0 stream-manager-sdk.zip
```

PowerShell

```
cp .\stream-manager-sdk.zip ~\greengrass-components\artifacts
\com.example.StreamManagerS3JS\1.0.0\
```

4. Créez votre recette de composants. Dans la recette, procédez comme suit :
 - a. Définissez `stream-manager-sdk.zip` comme un artefact.
 - b. Définissez votre JavaScript application comme un artefact.
 - c. Au cours du cycle de vie de l'installation, installez le SDK Stream Manager à partir de l'`stream-manager-sdk.zip` artefact. Cette `npm install` commande crée un `node_modules` dossier contenant le SDK Stream Manager et ses dépendances.

- d. Dans le cycle de vie d'exécution, ajoutez le `node_modules` dossier et exécutez votre JavaScript application. `NODE_PATH`

La recette de vos composants peut ressembler à l'exemple suivant. Ce composant exécute l'exemple [StreamManagerS3](#).

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.StreamManagerS3JS",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "Uses stream manager to upload a file to an S3 bucket.",
  "ComponentPublisher": "Amazon",
  "ComponentDependencies": {
    "aws.greengrass.StreamManager": {
      "VersionRequirement": "^2.0.0"
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "install": "npm install {artifacts:decompressedPath}/stream-manager-sdk/aws-greengrass-stream-manager-sdk",
        "Run": "export NODE_PATH=$NODE_PATH:{work:path}/node_modules; node {artifacts:path}/index.js"
      },
      "Artifacts": [
        {
          "URI": "s3://amzn-s3-demo-bucket/artifacts/com.example.StreamManagerS3JS/1.0.0/stream-manager-sdk.zip",
          "Unarchive": "ZIP"
        },
        {
          "URI": "s3://amzn-s3-demo-bucket/artifacts/com.example.StreamManagerS3JS/1.0.0/index.js"
        }
      ]
    }
  ]
}
```

```

    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "install": "npm install {artifacts:decompressedPath}/stream-manager-sdk/
aws-greengrass-stream-manager-sdk",
        "Run": "set \"NODE_PATH=%NODE_PATH%;{work:path}/node_modules\" & node
{artifacts:path}/index.js"
      },
      "Artifacts": [
        {
          "URI": "s3://amzn-s3-demo-bucket/artifacts/
com.example.StreamManagerS3JS/1.0.0/stream-manager-sdk.zip",
          "Unarchive": "ZIP"
        },
        {
          "URI": "s3://amzn-s3-demo-bucket/artifacts/
com.example.StreamManagerS3JS/1.0.0/index.js"
        }
      ]
    }
  ]
}

```

YAML

```

---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.StreamManagerS3JS
ComponentVersion: 1.0.0
ComponentDescription: Uses stream manager to upload a file to an S3 bucket.
ComponentPublisher: Amazon
ComponentDependencies:
  aws.greengrass.StreamManager:
    VersionRequirement: "^2.0.0"
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      install: npm install {artifacts:decompressedPath}/stream-manager-sdk/aws-
greengrass-stream-manager-sdk

```

```

Run: |
    export NODE_PATH=$NODE_PATH:{work:path}/node_modules
    node {artifacts:path}/index.js
Artifacts:
  - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3JS/1.0.0/stream-manager-sdk.zip
    Unarchive: ZIP
  - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3JS/1.0.0/index.js
  - Platform:
    os: windows
  Lifecycle:
    install: npm install {artifacts:decompressedPath}/stream-manager-sdk/aws-
greengrass-stream-manager-sdk
Run: |
    set "NODE_PATH=%NODE_PATH%;{work:path}/node_modules"
    node {artifacts:path}/index.js
Artifacts:
  - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3JS/1.0.0/stream-manager-sdk.zip
    Unarchive: ZIP
  - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3JS/1.0.0/index.js

```

Pour plus d'informations sur le développement et le test de composants, consultez [Création de AWS IoT Greengrass composants](#).

Connectez-vous au gestionnaire de flux dans le code de l'application

Pour vous connecter au gestionnaire de flux dans votre application, créez une instance de `StreamManagerClient` depuis le SDK de Stream Manager. Ce client se connecte au composant du gestionnaire de flux sur son port par défaut 8088, ou sur le port que vous spécifiez. Pour plus d'informations sur la façon de l'utiliser `StreamManagerClient` après avoir créé une instance, consultez [StreamManagerClient À utiliser pour travailler avec des flux](#).

Exemple Exemple : Connexion au gestionnaire de flux avec le port par défaut

Java

```
import com.amazonaws.greengrass.streammanager.client.StreamManagerClient;
```

```
public class MyStreamManagerComponent {  
  
    void connectToStreamManagerWithDefaultPort() {  
        StreamManagerClient client = StreamManagerClientFactory.standard().build();  
  
        // Use the client.  
    }  
}
```

Python

```
from stream_manager import (  
    StreamManagerClient  
)  
  
def connect_to_stream_manager_with_default_port():  
    client = StreamManagerClient()  
  
    # Use the client.
```

JavaScript

```
const {  
    StreamManagerClient  
} = require('aws-greengrass-stream-manager-sdk');  
  
function connectToStreamManagerWithDefaultPort() {  
    const client = new StreamManagerClient();  
  
    // Use the client.  
}
```

Exemple Exemple : Connexion au gestionnaire de flux avec un port autre que celui par défaut

Si vous configurez le gestionnaire de flux avec un port autre que le port par défaut, vous devez utiliser [la communication interprocessus](#) pour récupérer le port depuis la configuration du composant.

Note

Le paramètre `port` de configuration contient la valeur que vous spécifiez `STREAM_MANAGER_SERVER_PORT` lorsque vous déployez le gestionnaire de flux.

Java

```
void connectToStreamManagerWithCustomPort() {
    EventStreamRPCConnection eventStreamRpcConnection =
    IPCUtils.getEventStreamRpcConnection();
    GreengrassCoreIPCClient greengrassCoreIPCClient = new
    GreengrassCoreIPCClient(eventStreamRpcConnection);
    List<String> keyPath = new ArrayList<>();
    keyPath.add("port");

    GetConfigurationRequest request = new GetConfigurationRequest();
    request.setComponentName("aws.greengrass.StreamManager");
    request.setKeyPath(keyPath);
    GetConfigurationResponse response =
        greengrassCoreIPCClient.getConfiguration(request,
Optional.empty()).getResponse().get();
    String port = response.getValue().get("port").toString();
    System.out.print("Stream Manager is running on port: " + port);

    final StreamManagerClientConfig config = StreamManagerClientConfig.builder()

    .serverInfo(StreamManagerServerInfo.builder().port(Integer.parseInt(port)).build()).build()

    StreamManagerClient client =
    StreamManagerClientFactory.standard().withClientConfig(config).build();

    // Use the client.
}
```

Python

```
import awsiot.greengrasscoreipc
from awsiot.greengrasscoreipc.model import (
    GetConfigurationRequest
)
from stream_manager import (
```

```
StreamManagerClient
)

TIMEOUT = 10

def connect_to_stream_manager_with_custom_port():
    # Use IPC to get the port from the stream manager component configuration.
    ipc_client = awsiot.greengrasscoreipc.connect()
    request = GetConfigurationRequest()
    request.component_name = "aws.greengrass.StreamManager"
    request.key_path = ["port"]
    operation = ipc_client.new_get_configuration()
    operation.activate(request)
    future_response = operation.get_response()
    response = future_response.result(TIMEOUT)
    stream_manager_port = str(response.value["port"])

    # Use port to create a stream manager client.
    stream_client = StreamManagerClient(port=stream_manager_port)

    # Use the client.
```

StreamManagerClient À utiliser pour travailler avec des flux

Les composants Greengrass définis par l'utilisateur qui s'exécutent sur le périphérique principal de Greengrass peuvent utiliser `StreamManagerClient` l'objet du SDK Stream Manager pour créer des flux dans le gestionnaire de flux, [puis interagir avec les flux](#). Lorsqu'un composant crée un flux, il définit les AWS Cloud destinations, la hiérarchisation et les autres politiques d'exportation et de conservation des données pour le flux. Pour envoyer des données au gestionnaire de flux, les composants ajoutent les données au flux. Si une destination d'exportation est définie pour le flux, le gestionnaire de flux exporte le flux automatiquement.

Note

Généralement, les clients du gestionnaire de flux sont des composants Greengrass définis par l'utilisateur. Si votre analyse de rentabilisation l'exige, vous pouvez également autoriser les processus non liés aux composants exécutés sur le cœur de Greengrass (par exemple, un conteneur Docker) à interagir avec le gestionnaire de flux. Pour de plus amples informations, veuillez consulter [the section called "Authentication client"](#).

Les extraits de cette rubrique vous montrent comment les clients appellent des `StreamManagerClient` méthodes pour travailler avec des flux. Pour plus de détails sur l'implémentation des méthodes et de leurs arguments, utilisez les liens vers la référence du SDK répertoriée après chaque extrait.

Si vous utilisez le gestionnaire de flux dans une fonction Lambda, votre fonction Lambda doit être instanciée `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 instantiation `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 “Mettre à jour le flux de messages”](#)
- [the section called “Supprimer le flux de messages”](#)

Créer un flux de messages

Pour créer un flux, un composant Greengrass défini par l'utilisateur appelle la méthode `create` et transmet un objet `MessageStreamDefinition`. Cet objet spécifie le nom unique du flux et définit comment 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 :

- La cible AWS IoT Analytics, Kinesis Data Streams et AWS IoT SiteWise les destinations Amazon S3 pour les exportations automatiques. Pour de plus amples informations, veuillez consulter [the section called “Exporter les configurations pour les destinations cloud prises en charge”](#).
- Exportez la priorité. Le gestionnaire de flux exporte les flux de priorité supérieure avant les flux de priorité inférieure.
- Taille de lot et intervalle de lot maximaux pour AWS IoT Analytics Kinesis Data Streams AWS IoT SiteWise et les destinations. Le gestionnaire de flux exporte les messages lorsque l'une ou l'autre des conditions est remplie.
- Time-to-live (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 départ. Spécifiez le numéro de séquence du message à utiliser comme message de départ lors de l'exportation.

Pour plus d'informations `MessageStreamDefinition`, consultez la référence du SDK pour votre langue cible :

- [MessageStreamDefinition](#) dans le SDK Java
- [MessageStreamDefinition](#) dans le SDK Node.js
- [MessageStreamDefinition](#) dans le 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 composants Greengrass peuvent [ajouter des messages](#) au flux pour envoyer des données à exporter et [lire les messages](#) du flux pour un traitement local. Le nombre de flux que vous créez dépend de vos capacités matérielles et de votre analyse de rentabilisation. L'une

des stratégies consiste à créer un flux pour chaque canal cible dans AWS IoT Analytics le flux de données Kinesis, bien que vous puissiez définir plusieurs cibles pour un flux. Un flux a un cycle de vie durable.

Prérequis

Cette opération répond aux exigences suivantes :

- Version minimale du SDK Stream Manager : Python : 1.1.0 | Java : 1.1.0 | Node.js : 1.1.0

Exemples

L'extrait de code suivant crée un flux nommé StreamName. Il définit les propriétés du flux dans les types de données 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 SDK Python : [create_message_stream](#) | [MessageStreamDefinition](#)

Java

```
try (final StreamManagerClient client =
    StreamManagerClientFactory.standard().build()) {
    client.createMessageStream(
        new MessageStreamDefinition()
            .WithName("StreamName") // Required.
            .withMaxSize(268435456L) // Default is 256 MB.
            .withStreamSegmentSize(16777216L) // Default is 16 MB.
            .withTimeToLiveMillis(null) // By default, no TTL is enabled.
            .withStrategyOnFull(StrategyOnFull.OverwriteOldestData) //
Required.

            .withPersistence(Persistence.File) // Default is File.
            .withFlushOnWrite(false) // Default is false.
            .withExportDefinition( // Optional. Choose where/how the
stream is exported to the AWS Cloud.
                new ExportDefinition()
                    .withKinesis(null)
                    .withIotAnalytics(null)
                    .withIotSitewise(null)
                    .withS3(null)
                )
            );
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

Référence du SDK Java : | [createMessageStreamMessageStreamDefinition](#)

Node.js

```
const client = new StreamManagerClient();
client.onConnected(async () => {
    try {
        await client.createMessageStream(
            new MessageStreamDefinition()
                .WithName("StreamName") // Required.
                .withMaxSize(268435456) // Default is 256 MB.
                .withStreamSegmentSize(16777216) // Default is 16 MB.
                .withTimeToLiveMillis(null) // By default, no TTL is enabled.
                .withStrategyOnFull(StrategyOnFull.OverwriteOldestData) // Required.
            );
    }
});
```

```
.withPersistence(Persistence.File) // Default is File.
.withFlushOnWrite(false) // Default is false.
.withExportDefinition( // Optional. Choose where/how the stream is exported
to the AWS Cloud.
    new ExportDefinition()
        .withKinesis(null)
        .withIotAnalytics(null)
        .withIotSiteWise(null)
        .withS3(null)
    )
);
} catch (e) {
    // Properly handle errors.
}
});
client.onError((err) => {
    // Properly handle connection errors.
    // This is called only when the connection to the StreamManager server fails.
});
```

Référence du SDK Node.js : | [createMessageStreamMessageStreamDefinition](#)

Pour plus d'informations sur la configuration des destinations d'exportation, consultez [the section called “Exporter les configurations pour les destinations cloud prises en charge”](#).

Ajouter un message

Pour envoyer des données au gestionnaire de flux à des fins d'exportation, vos composants Greengrass ajoutent les données au flux cible. La destination d'exportation détermine le type de données à transmettre à cette méthode.

Prérequis

Cette opération répond aux exigences suivantes :

- Version minimale du SDK Stream Manager : Python : 1.1.0 | Java : 1.1.0 | Node.js : 1.1.0

Exemples

AWS IoT Analytics ou destinations d'exportation Kinesis Data Streams

L'extrait de code suivant ajoute un message au flux nommé `StreamName`. Pour AWS IoT Analytics nos destinations Kinesis Data Streams, vos composants Greengrass ajoutent un blob de données.

Cet extrait de code répond aux exigences suivantes :

- Version minimale du SDK Stream Manager : Python : 1.1.0 | Java : 1.1.0 | Node.js : 1.1.0

Python

```
client = StreamManagerClient()

try:
    sequence_number = client.append_message(stream_name="StreamName",
    data=b'Arbitrary bytes data')
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
    # Properly handle errors.
```

[Référence du SDK Python : `append\_message`](#)

Java

```
try (final StreamManagerClient client =
    StreamManagerClientFactory.standard().build()) {
    long sequenceNumber = client.appendMessage("StreamName", "Arbitrary byte
    array".getBytes());
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

[Référence du 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 SDK Node.js : AppendMessage](#)

AWS IoT SiteWise destinations d'exportation

L'extrait de code suivant ajoute un message au flux nommé StreamName. Pour les AWS IoT SiteWise destinations, vos composants Greengrass ajoutent un objet sérialisé. PutAssetPropertyValueEntry Pour de plus amples informations, veuillez consulter [the section called "Exporter vers AWS IoT SiteWise"](#).

Note

Lorsque vous envoyez des données à AWS IoT SiteWise, celles-ci doivent répondre aux exigences de l'BatchPutAssetPropertyValueaction. Pour plus d'informations, consultez [BatchPutAssetPropertyValue](#) dans la Référence d'API AWS IoT SiteWise .

Cet extrait de code répond aux exigences suivantes :

- Version minimale du SDK Stream Manager : Python : 1.1.0 | Java : 1.1.0 | Node.js : 1.1.0

Python

```
client = StreamManagerClient()

try:
    # SiteWise requires unique timestamps in all messages and also needs timestamps
    not earlier
```

```

# than 10 minutes in the past. Add some randomness to time and offset.

# Note: To create a new asset property data, you should use the classes defined
in the
# greengrasssdk.stream_manager module.

time_in_nanos = TimeInNanos(
    time_in_seconds=calendar.timegm(time.gmtime()) - random.randint(0, 60),
    offset_in_nanos=random.randint(0, 10000)
)
variant = Variant(double_value=random.random())
asset = [AssetPropertyValue(value=variant, quality=Quality.GOOD,
timestamp=time_in_nanos)]
putAssetPropertyValueEntry =
PutAssetPropertyValueEntry(entry_id=str(uuid.uuid4()),
property_alias="PropertyAlias", property_values=asset)
sequence_number = client.append_message(stream_name="StreamName",
Util.validate_and_serialize_to_json_bytes(putAssetPropertyValueEntry))
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
    # Properly handle errors.

```

Référence du 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 and also needs
timestamps not earlier
    // than 10 minutes in the past. Add some randomness to time and offset.
    final int maxTimeRandomness = 60;
    final int maxOffsetRandomness = 10000;
    double randomValue = rand.nextDouble();
    TimeInNanos timestamp = new TimeInNanos()

```

```

        .withTimeInSeconds(Instant.now().getEpochSecond() -
rand.nextInt(maxTimeRandomness))
        .withOffsetInNanos((long) (rand.nextInt(maxOffsetRandomness)));
    AssetPropertyValue entry = new AssetPropertyValue()
        .withValue(new Variant().withDoubleValue(randomValue))
        .withQuality(Quality.GOOD)
        .withTimestamp(timestamp);
    entries.add(entry);

    PutAssetPropertyValueEntry putAssetPropertyValueEntry = new
PutAssetPropertyValueEntry()
        .withEntryId(UUID.randomUUID().toString())
        .withPropertyAlias("PropertyAlias")
        .withPropertyValues(entries);
    long sequenceNumber = client.appendMessage("StreamName",
ValidateAndSerialize.validateAndSerializeToJsonBytes(putAssetPropertyValueEntry));
} catch (StreamManagerException e) {
    // Properly handle exception.
}

```

Référence du 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 SDK Node.js : [AppendMessage | PutAssetPropertyValueEntry](#)

Destinations d'exportation Amazon S3

L'extrait suivant ajoute une tâche d'exportation au flux nommé. `StreamName` Pour les destinations Amazon S3, vos composants Greengrass ajoutent un `S3ExportTaskDefinition` objet sérialisé contenant des 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 de plus amples informations, veuillez consulter [the section called “Exportation vers Amazon S3”](#).

Cet extrait de code répond aux exigences suivantes :

- Version minimale du SDK Stream Manager : Python : 1.1.0 | Java : 1.1.0 | Node.js : 1.1.0

Python

```

client = StreamManagerClient()

try:
    # Append an Amazon S3 Task definition and print the sequence number.
    s3_export_task_definition = S3ExportTaskDefinition(input_url="URLToFile",
    bucket="BucketName", key="KeyName")
    sequence_number = client.append_message(stream_name="StreamName",
    Util.validate_and_serialize_to_json_bytes(s3_export_task_definition))
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:

```

```
pass
# Properly handle errors.
```

[Référence du SDK Python : `append\_message` | `S3 ExportTaskDefinition`](#)

Java

```
try (final StreamManagerClient client =
    GreengrassClientBuilder.streamManagerClient().build()) {
    // Append an Amazon S3 export task definition and print the sequence number.
    S3ExportTaskDefinition s3ExportTaskDefinition = new S3ExportTaskDefinition()
        .withBucket("BucketName")
        .withKey("KeyName")
        .withInputUrl("URLToFile");
    long sequenceNumber = client.appendMessage("StreamName",
        ValidateAndSerialize.validateAndSerializeToJsonBytes(s3ExportTaskDefinition));
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

[Référence du SDK Java : `AppendMessage` | `S3 ExportTaskDefinition`](#)

Node.js

```
const client = new StreamManagerClient();
client.onConnected(async () => {
    try {
        // Append an Amazon S3 export task definition and print the sequence number.
        const taskDefinition = new S3ExportTaskDefinition()
            .withBucket("BucketName")
            .withKey("KeyName")
            .withInputUrl("URLToFile");
        const sequenceNumber = await client.appendMessage("StreamName",
            util.validateAndSerializeToJsonBytes(taskDefinition));
    } catch (e) {
        // Properly handle errors.
    }
});
client.onError((err) => {
    // Properly handle connection errors.
    // This is called only when the connection to the StreamManager server fails.
});
```

[Référence du SDK Node.js : `AppendMessage` | `S3 ExportTaskDefinition`](#)

Lire des messages

Lisez les messages d'un stream.

Prérequis

Cette opération répond aux exigences suivantes :

- Version minimale du SDK Stream Manager : Python : 1.1.0 | Java : 1.1.0 | Node.js : 1.1.0

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_message_count to be
            fulfilled. By default, this is 0, which immediately returns the messages or an
            exception.
        )
    )
except StreamManagerException:
```

```
pass
# Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
# Properly handle errors.
```

Référence du SDK Python : [read_messages](#) | [ReadMessagesOptions](#)

Java

```
try (final StreamManagerClient client =
    StreamManagerClientFactory.standard().build()) {
    List<Message> messages = client.readMessages("StreamName",
        // By default, if no options are specified, it tries to read one message
        from the beginning of the stream.
        new ReadMessagesOptions()
            // Try to read from sequence number 100 or greater. By default
            this is 0.
            .withDesiredStartSequenceNumber(100L)
            // Try to read 10 messages. If 10 messages are not available,
            then NotEnoughMessagesException is raised. By default, this is 1.
            .withMinMessageCount(10L)
            // Accept up to 100 messages. By default this is 1.
            .withMaxMessageCount(100L)
            // Try to wait at most 5 seconds for the min_message_count to
            be fulfilled. By default, this is 0, which immediately returns the messages or an
            exception.
            .withReadTimeoutMillis(Duration.ofSeconds(5L).toMillis())
    );
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

Référence du 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 SDK Node.js : [ReadMessages](#) | [ReadMessagesOptions](#)

Afficher la liste des flux

Obtenez la liste des flux dans le gestionnaire de flux.

Prérequis

Cette opération répond aux exigences suivantes :

- Version minimale du SDK Stream Manager : Python : 1.1.0 | Java : 1.1.0 | Node.js : 1.1.0

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 SDK Python : list_streams](#)

Java

```
try (final StreamManagerClient client =
    StreamManagerClientFactory.standard().build()) {
    List<String> streamNames = client.listStreams();
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

[Référence du 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 SDK Node.js : ListStreams](#)

Décrire le flux de messages

Obtenez les métadonnées relatives à un flux, notamment sa définition, sa taille et son statut d'exportation.

Prérequis

Cette opération répond aux exigences suivantes :

- Version minimale du SDK Stream Manager : Python : 1.1.0 | Java : 1.1.0 | Node.js : 1.1.0

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 SDK Python : `describe\_message\_stream`](#)

Java

```
try (final StreamManagerClient client =
    StreamManagerClientFactory.standard().build()) {
    MessageStreamInfo description = client.describeMessageStream("StreamName");
    String lastErrorMessage =
description.getExportStatuses().get(0).getErrorMessage();
    if (lastErrorMessage != null && !lastErrorMessage.equals("")) {
        // The last export of export destination 0 failed with some error.
        // Here is the last sequence number that was successfully exported.
        description.getExportStatuses().get(0).getLastExportedSequenceNumber();
    }

    if (description.getStorageStatus().getNewestSequenceNumber() >
        description.getExportStatuses().get(0).getLastExportedSequenceNumber())
    {
        // The end of the stream is ahead of the last exported sequence number.
    }
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

Référence du 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 SDK Node.js : [describeMessageStream](#)

Mettre à jour le flux de messages

Mettez à jour les propriétés d'un flux existant. Vous souhaitez peut-être mettre à jour un flux si vos exigences changent après sa création. Par exemple :

- Ajoutez une nouvelle [configuration d'exportation](#) pour une AWS Cloud destination.
- 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 en matière de paramètres complets peut entraîner la suppression ou le rejet de données avant que le gestionnaire de flux ne puisse les traiter.
- Interrompez et reprenez les exportations, par exemple, si les tâches d'exportation sont longues et que vous souhaitez rationner vos données de téléchargement.

Vos composants Greengrass suivent ce processus de haut niveau pour mettre à jour un flux :

1. [Obtenez la description du stream.](#)
2. Mettez à jour les propriétés cibles sur les objets correspondants `MessageStreamDefinition` et subordonnés.
3. Transmettez 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 leurs valeurs par défaut.

Vous pouvez spécifier le numéro de séquence du message à utiliser comme message de départ lors de l'exportation.

Prérequis

Cette opération répond aux exigences suivantes :

- Version minimale du SDK Stream Manager : Python : 1.1.0 | Java : 1.1.0 | Node.js : 1.1.0

Exemples

L'extrait suivant met à jour le flux nommé. StreamName Elle 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 SDK Python : | [updateMessageStreamMessageStreamDefinition](#)

Java

```
try (final StreamManagerClient client =
    GreengrassClientBuilder.streamManagerClient().build()) {
    MessageStreamInfo messageStreamInfo = client.describeMessageStream(STREAM_NAME);
    // Update the message stream with new values.
    client.updateMessageStream(
        messageStreamInfo.getDefinition()
            .withStrategyOnFull(StrategyOnFull.RejectNewData) // Required. Updating
            Strategy on full to reject new data.
```

```

        // Max Size update should be greater than initial Max Size defined in
        Create Message Stream request
        .withMaxSize(536870912L) // Update Max Size to 512 MB.
        .withStreamSegmentSize(33554432L) // Update Segment Size to 32 MB.
        .withFlushOnWrite(true) // Update flush on write to true.
        .withPersistence(Persistence.Memory) // Update the persistence to
        Memory.

        .withTimeToLiveMillis(3600000L) // Update TTL to 1 hour.
        .withExportDefinition(
            // Optional. Choose where/how the stream is exported to the 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 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 SDK Node.js : | [updateMessageStreamMessageStreamDefinition](#)

Contraintes de mise à jour des flux

Les contraintes suivantes s'appliquent lors de la mise à jour des flux. Sauf indication contraire 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, [supprimez le flux](#) et [créez un flux](#) qui 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, [décrivez le flux](#), puis vérifiez l'état de stockage de l'`MessageStreamInfo` objet renvoyé.
 - 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 Time to Live (TTL) s'appliquent aux nouvelles opérations d'ajout. Si vous diminuez cette valeur, le gestionnaire de flux peut également supprimer les segments existants qui dépassent le TTL.
- Les mises à jour de la stratégie relative à la propriété complète s'appliquent aux nouvelles opérations d'ajout. Si vous définissez une stratégie pour remplacer les données les plus anciennes, le gestionnaire de flux peut également remplacer 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. Dans le cas contraire, 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.
- Pour [mettre à jour](#) le numéro 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, [décrivez le flux](#), puis vérifiez l'état de stockage de l'MessageStreamInfoobjet renvoyé.

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 répond aux exigences suivantes :

- Version minimale du SDK Stream Manager : Python : 1.1.0 | Java : 1.1.0 | Node.js : 1.1.0

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 SDK Python : [deleteMessageStream](#)

Java

```
try (final StreamManagerClient client =
    StreamManagerClientFactory.standard().build()) {
    client.deleteMessageStream("StreamName");
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

Référence du 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.
});
```

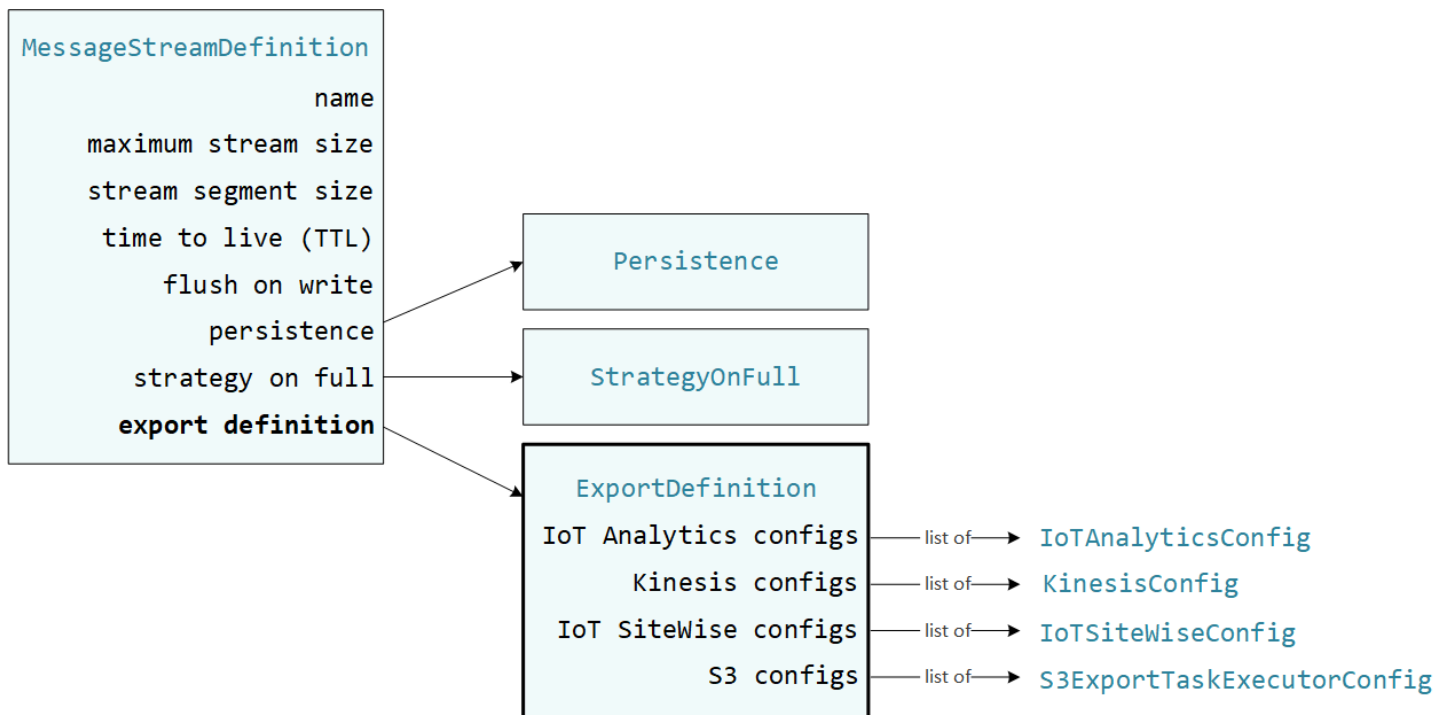
Référence du SDK Node.js : [deleteMessageStream](#)

Consultez aussi

- [Gérez les flux de données sur les appareils principaux de Greengrass](#)
- [Configuration du gestionnaire de AWS IoT Greengrass flux](#)
- [Exporter les configurations pour les AWS Cloud destinations prises en charge](#)
- StreamManagerClient dans la référence du SDK Stream Manager :
 - [Python](#)
 - [Java](#)
 - [Node.js](#)

Exporter les configurations pour les AWS Cloud destinations prises en charge

Les composants Greengrass définis par l'utilisateur sont StreamManagerClient utilisés dans le SDK Stream Manager pour interagir avec le Stream Manager. Lorsqu'un composant [crée un flux](#) ou [met à jour un flux](#), il transmet un MessageStreamDefinition objet qui représente les propriétés du flux, y compris la définition de l'exportation. L'ExportDefinition objet contient les configurations d'exportation définies pour le flux. Le gestionnaire de flux utilise ces configurations d'exportation pour déterminer où et comment exporter le flux.



Vous pouvez définir zéro ou plusieurs configurations d'exportation sur un flux, y compris plusieurs configurations d'exportation pour un seul type de destination. Par exemple, vous pouvez exporter un flux vers deux AWS IoT Analytics canaux et un flux de données Kinesis.

En cas d'échec des tentatives d'exportation, le gestionnaire de flux essaie continuellement d'exporter les données à des AWS Cloud intervalles allant jusqu'à cinq minutes. Le nombre de nouvelles tentatives n'est pas limité.

 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.

AWS Cloud Destinations prises en charge

- [AWS IoT Analytics chaînes](#)
- [Flux de données Amazon Kinesis](#)
- [AWS IoT SiteWise propriétés des actifs](#)
- [Objets Amazon S3](#)

Vous êtes responsable de la maintenance de ces AWS Cloud ressources.

AWS IoT Analytics chaînes

Le gestionnaire de flux prend en charge les exportations automatiques vers AWS IoT Analytics. AWS IoT Analytics vous permet d'effectuer une analyse avancée de vos données afin de prendre des décisions commerciales et d'améliorer les modèles d'apprentissage automatique. Pour plus d'informations, voir [Qu'est-ce que c'est AWS IoT Analytics ?](#) dans le guide de AWS IoT Analytics l'utilisateur.

Dans le SDK Stream Manager, vos composants Greengrass utilisent `IoTAnalyticsConfig` le pour définir la configuration d'exportation pour ce type de destination. Pour plus d'informations, consultez la référence du SDK pour votre langue cible :

- [IoTAnalyticsConfig](#) dans le SDK Python

- [IoT Analytics Config](#) dans le SDK Java
- [IoT Analytics Config](#) dans le SDK Node.js

Exigences

Cette destination d'exportation répond aux exigences suivantes :

- Les canaux cibles AWS IoT Analytics entrants doivent être identiques à ceux Compte AWS de Région AWS l'appareil principal Greengrass.
- Ils [Autoriser les appareils principaux à interagir avec les AWS services](#) doivent `iotanalytics:BatchPutMessage` autoriser les chaînes cibles. Par exemple :

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iotanalytics:BatchPutMessage"
      ],
      "Resource": [
        "arn:aws:iotanalytics:us-east-1:123456789012:channel/channel_1_name",
        "arn:aws:iotanalytics:us-east-1:123456789012:channel/channel_2_name"
      ]
    }
  ]
}
```

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.

Exporter vers AWS IoT Analytics

Pour créer un flux exporté vers AWS IoT Analytics, vos composants Greengrass [créent un flux avec une](#) définition d'exportation qui inclut un ou plusieurs `IoTAnalyticsConfig` objets. Cet objet définit les paramètres d'exportation, tels que le canal cible, la taille du lot, l'intervalle entre les lots et la priorité.

Lorsque vos composants Greengrass reçoivent des données provenant d'appareils, ils [ajoutent des messages](#) contenant une quantité importante de données au 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.

Flux de données Amazon Kinesis

Le gestionnaire de flux prend en charge les exportations automatiques vers Amazon Kinesis Data Streams. Kinesis Data Streams est couramment utilisé pour agréger de gros volumes de données et les charger dans un entrepôt MapReduce de données ou un cluster. Pour plus d'informations, consultez [Qu'est-ce qu'Amazon Kinesis Data Streams ?](#) dans le manuel Amazon Kinesis Developer Guide.

Dans le SDK Stream Manager, vos composants Greengrass utilisent `KinesisConfig` pour définir la configuration d'exportation pour ce type de destination. Pour plus d'informations, consultez la référence du SDK pour votre langue cible :

- [KinesisConfig](#) dans le SDK Python
- [KinesisConfig](#) dans le SDK Java
- [KinesisConfig](#) dans le SDK Node.js

Exigences

Cette destination d'exportation répond aux exigences suivantes :

- Dans Kinesis Data Streams, les flux cibles doivent se trouver dans le Compte AWS même appareil que celui Région AWS de base de Greengrass.
- (Recommandé) Le gestionnaire de flux v2.2.1 améliore les performances d'exportation de flux vers des destinations Kinesis Data Streams. Pour utiliser les améliorations de cette dernière version, mettez à niveau votre [composant de gestionnaire de flux](#) vers la version 2.2.1 et utilisez la `kinesis:ListShards` politique du rôle d'échange de jetons [Greengrass](#).
- Ils [Autoriser les appareils principaux à interagir avec les AWS services](#) doivent `kinesis:PutRecords` autoriser le ciblage des flux de données. Par exemple :

JSON

```
{
```

```
"Version": "2012-10-17",
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "kinesis:PutRecords"
    ],
    "Resource": [
      "arn:aws:kinesis:us-east-1:123456789012:stream/stream_1_name",
      "arn:aws:kinesis:us-east-1:123456789012:stream/stream_2_name"
    ]
  }
]
```

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.

Exportation vers Kinesis Data Streams

Pour créer un flux exporté vers Kinesis Data Streams, vos [composants Greengrass créent un flux avec une](#) définition d'exportation qui inclut un ou plusieurs objets. KinesisConfig Cet objet définit les paramètres d'exportation, tels que le flux de données cible, la taille du lot, l'intervalle entre les lots et la priorité.

Lorsque vos composants Greengrass reçoivent des données provenant d'appareils, ils [ajoutent des messages](#) contenant une quantité importante de données au 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.

Le gestionnaire de flux génère un UUID unique et aléatoire comme clé de partition pour chaque enregistrement chargé sur Amazon Kinesis.

AWS IoT SiteWise propriétés des actifs

Le gestionnaire de flux prend en charge les exportations automatiques vers AWS IoT SiteWise. AWS IoT SiteWise vous permet de collecter, d'organiser et d'analyser les données des équipements industriels à grande échelle. Pour plus d'informations, voir [Qu'est-ce que c'est AWS IoT SiteWise ?](#) dans le guide de AWS IoT SiteWise l'utilisateur.

Dans le SDK Stream Manager, vos composants Greengrass utilisent `IoTSiteWiseConfig` pour définir la configuration d'exportation pour ce type de destination. Pour plus d'informations, consultez la référence du SDK pour votre langue cible :

- [IoTSiteWiseConfig](#) dans le SDK Python
- [IoTSiteWiseConfig](#) dans le SDK Java
- [IoTSiteWiseConfig](#) dans le SDK Node.js

 Note

AWS fournit également des AWS IoT SiteWise composants, qui offrent une solution prédéfinie que vous pouvez utiliser pour diffuser des données à partir de sources OPC-UA. Pour de plus amples informations, veuillez consulter [Collecteur IoT SiteWise OPC UA](#).

Prérequis

Cette destination d'exportation répond aux exigences suivantes :

- Les propriétés de l'actif cible AWS IoT SiteWise doivent être identiques Compte AWS à Région AWS celles du périphérique principal Greengrass.

 Note

Pour la liste de Région AWS compatibles, voir AWS IoT SiteWise les [AWS IoT SiteWise points de terminaison et les quotas](#) dans la référence AWS générale.

- Ils [Autoriser les appareils principaux à interagir avec les AWS services](#) doivent `iotsitewise:BatchPutAssetPropertyValue` autoriser le ciblage des propriétés des actifs. L'exemple de politique suivant utilise la clé de `iotsitewise:assetHierarchyPath` condition pour accorder l'accès à une ressource racine cible et à ses enfants. Vous pouvez le supprimer Condition de la politique pour autoriser l'accès à tous vos AWS IoT SiteWise actifs ou spécifier ARNs des actifs individuels.

JSON

```
{
  "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 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.

Pour obtenir des informations de sécurité importantes, consultez la section [BatchPutAssetPropertyValue autorisation](#) dans le guide de AWS IoT SiteWise l'utilisateur.

Exporter vers AWS IoT SiteWise

Pour créer un flux exporté vers AWS IoT SiteWise, vos composants Greengrass [créent un flux avec une](#) définition d'exportation qui inclut un ou plusieurs `IoTSiteWiseConfig` objets. Cet objet définit les paramètres d'exportation, tels que la taille du lot, l'intervalle entre les lots et la priorité.

Lorsque vos composants Greengrass reçoivent des données sur les propriétés des actifs depuis des appareils, ils ajoutent des messages contenant ces données au flux cible. Les messages sont des `PutAssetPropertyValueEntry` objets sérialisés en JSON qui contiennent des valeurs de propriété pour une ou plusieurs propriétés d'actifs. Pour plus d'informations, voir [Ajouter un message pour](#) les destinations AWS IoT SiteWise d'exportation.

Note

Lorsque vous envoyez des données à AWS IoT SiteWise, celles-ci doivent répondre aux exigences de l'`BatchPutAssetPropertyValue` action. Pour plus d'informations, consultez [BatchPutAssetPropertyValue](#) dans la Référence d'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 composants Greengrass pour concevoir votre stratégie d'exportation. Par exemple :

- Pour les exportations en temps quasi réel, définissez des paramètres de taille de lot et d'intervalle faibles et ajoutez les données au flux dès leur réception.
- Pour optimiser le traitement par lots, atténuer les contraintes de bande passante ou minimiser les coûts, vos composants Greengrass peuvent regrouper timestamp-quality-value les points de données (TQV) reçus pour une propriété d'actif unique avant d'ajouter les données au flux. L'une des stratégies consiste à regrouper les entrées pour un maximum de 10 combinaisons propriétés-actifs différentes, ou alias de propriété, dans un seul message au lieu d'envoyer plusieurs entrées pour la même propriété. Cela permet au gestionnaire de flux de rester dans les limites [AWS IoT SiteWise des quotas](#).

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 plus d'informations, consultez [Qu'est-ce qu'Amazon S3 ?](#) dans le guide du développeur d'Amazon Simple Storage Service.

Dans le SDK Stream Manager, vos composants Greengrass utilisent `S3ExportTaskExecutorConfig` pour définir la configuration d'exportation pour ce type de destination. Pour plus d'informations, consultez la référence du SDK pour votre langue cible :

- [S3 ExportTaskExecutorConfig](#) dans le SDK Python
- [S3 ExportTaskExecutorConfig](#) dans le SDK Java
- [S3 ExportTaskExecutorConfig](#) dans le SDK Node.js

Exigences

Cette destination d'exportation répond aux exigences suivantes :

- Les compartiments Amazon S3 cibles doivent se trouver dans le même emplacement Compte AWS que le périphérique principal de Greengrass.

- Si une fonction Lambda exécutée en mode conteneur Greengrass écrit des fichiers d'entrée dans un répertoire de fichiers d'entrée, vous devez monter le répertoire en tant que volume dans le conteneur avec des autorisations d'écriture. Cela garantit que les fichiers sont écrits dans le système de fichiers racine et visibles par le composant du gestionnaire de flux, qui s'exécute en dehors du conteneur.
- Si un composant de conteneur Docker écrit des fichiers d'entrée dans un répertoire de fichiers d'entrée, vous devez monter le répertoire en tant que volume dans le conteneur avec des autorisations d'écriture. Cela garantit que les fichiers sont écrits dans le système de fichiers racine et visibles par le composant du gestionnaire de flux, qui s'exécute en dehors du conteneur.
- Ils [Autoriser les appareils principaux à interagir avec les AWS services](#) doivent accorder les autorisations suivantes aux compartiments cibles. Par exemple :

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:PutObject",
        "s3:AbortMultipartUpload",
        "s3:ListMultipartUploadParts"
      ],
      "Resource": [
        "arn:aws:s3:::bucket-1-name/*",
        "arn:aws:s3:::bucket-2-name/*"
      ]
    }
  ]
}
```

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.

Exportation vers Amazon S3

Pour créer un flux exporté vers Amazon S3, vos composants Greengrass utilisent l'`S3ExportTaskExecutorConfig` objet pour configurer la politique d'exportation. La politique définit les paramètres d'exportation, tels que le seuil et la priorité du téléchargement en plusieurs parties. Pour les exportations Amazon S3, le gestionnaire de flux télécharge les données qu'il lit à partir de fichiers locaux sur l'appareil principal. Pour lancer un téléchargement, vos composants Greengrass 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 l'ordre dans lequel elles sont ajoutées au flux.

Note

Le compartiment cible doit déjà exister dans votre Compte AWS. Si aucun objet correspondant à la clé spécifiée n'existe, le gestionnaire de flux le crée pour vous.

Le gestionnaire de flux utilise la propriété de seuil de téléchargement en plusieurs parties, le paramètre de [taille de pièce minimale](#) et la taille du fichier d'entrée pour déterminer le mode de téléchargement des données. Le seuil de téléchargement partitionné doit être supérieur ou égal à la taille de pièce minimale. Si vous souhaitez télécharger des données en parallèle, vous pouvez créer plusieurs flux.

Les clés qui spécifient vos objets Amazon S3 cibles peuvent inclure des `DateTimeFormatter` chaînes [Java](#) valides dans les `!{timestamp: value}` espaces réservés. Vous pouvez utiliser ces espaces réservés d'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 correspond à une valeur telle que `my-key/!{timestamp:YYYY}/!{timestamp:MM}/!{timestamp:dd}/data.txt`.

```
my-key/!{timestamp:YYYY}/!{timestamp:MM}/!{timestamp:dd}/data.txt
```

Note

Si vous souhaitez surveiller l'état d'exportation d'un flux, créez d'abord un flux d'état, puis configurez le flux d'exportation pour l'utiliser. Pour de plus amples informations, veuillez consulter [the section called “Surveiller les tâches d'exportation”](#).

Gérer les données d'entrée

Vous pouvez créer du code que les applications IoT utilisent pour gérer le cycle de vie des données d'entrée. L'exemple de flux de travail suivant montre comment vous pouvez utiliser les composants Greengrass pour gérer ces données.

1. Un processus local reçoit des données provenant d'appareils ou de périphériques, puis écrit les données dans des fichiers situés dans un répertoire du périphérique principal. Il s'agit des fichiers d'entrée pour le gestionnaire de flux.
2. Un composant Greengrass analyse le répertoire et [ajoute une tâche d'exportation](#) au flux cible lorsqu'un nouveau fichier est créé. La tâche est un `S3ExportTaskDefinition` objet sérialisé en JSON qui spécifie l'URL du fichier d'entrée, le compartiment et la clé Amazon S3 cibles, 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 votre Compte AWS. Si aucun objet correspondant à la clé spécifiée n'existe, le gestionnaire de flux le crée pour vous.
4. Le composant Greengrass [lit les messages](#) d'un flux d'état pour surveiller le statut de l'exportation. Une fois les tâches d'exportation terminées, le composant Greengrass peut supprimer les fichiers d'entrée correspondants. Pour de plus amples informations, veuillez consulter [the section called “Surveiller les tâches d'exportation”](#).

Surveiller les tâches d'exportation

Vous pouvez créer du code que les applications IoT utilisent pour surveiller le statut de vos exportations Amazon S3. Vos composants Greengrass doivent créer un flux d'état, puis configurer le flux d'exportation pour écrire des mises à jour de statut dans le flux d'état. Un seul flux de statut peut recevoir des mises à jour de statut provenant de plusieurs flux exportés vers Amazon S3.

[Créez d'abord un flux](#) à utiliser comme flux d'état. Vous pouvez configurer la taille et les politiques de rétention du flux afin de contrôler la durée de vie des messages d'état. Par exemple :

- Définissez `Persistence` cette `Memory` option si vous ne souhaitez pas enregistrer les messages d'état.
- Réglez `StrategyOnFull` sur `OverwriteOldestData` que les nouveaux messages d'état ne soient pas perdus.

Créez ou mettez à jour le flux d'exportation pour utiliser le flux d'état. Spécifiquement, définissez la propriété de configuration d'état de la configuration `S3ExportTaskExecutorConfig` d'exportation du flux. Ce paramètre indique au gestionnaire de flux d'écrire des messages d'état concernant les tâches d'exportation dans le flux d'état. Dans l'`StatusConfig`objet, spécifiez le nom du flux d'état et le niveau de verbosité. Les valeurs prises en charge suivantes vont de la moins détaillée (ERROR) à la plus détaillée (). TRACE La valeur par défaut est INFO.

- ERROR
- WARN
- INFO
- DEBUG
- TRACE

L'exemple de flux de travail suivant montre comment les composants Greengrass peuvent utiliser un flux d'état pour surveiller le statut des exportations.

1. Comme décrit dans le flux de travail précédent, un composant [Greengrass ajoute une tâche d'exportation](#) à un flux configuré pour écrire des messages d'état concernant les tâches d'exportation dans un flux de statut. L'opération d'ajout renvoie un numéro de séquence qui représente l'ID de tâche.
2. Un composant Greengrass [lit les messages](#) de 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 du contexte du message. Par exemple, le composant Greengrass peut filtrer en fonction de l'URL du fichier d'entrée de la tâche d'exportation, qui est représentée par l'`S3ExportTaskDefinition`objet dans le contexte du message.

Les codes d'état suivants indiquent qu'une tâche d'exportation est terminée :

- Success. Le téléchargement a été effectué avec succès.
- Failure. Le gestionnaire de flux a rencontré une erreur. Par exemple, le bucket spécifié n'existe pas. Une fois le problème résolu, vous pouvez à nouveau ajouter la tâche d'exportation au flux.
- Canceled. La tâche a été arrêtée car le flux ou la définition d'exportation a été supprimé ou parce que la période time-to-live (TTL) de la tâche a expiré.

 Note

La tâche peut également avoir le statut `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 téléchargement partiel renvoie un avertissement.

3. Une fois les tâches d'exportation terminées, le composant Greengrass peut supprimer les fichiers d'entrée correspondants.

L'exemple suivant montre comment un composant Greengrass peut lire et traiter les messages d'état.

Python

```
import time
from stream_manager import (
    ReadMessagesOptions,
    Status,
    StatusConfig,
    StatusLevel,
    StatusMessage,
    StreamManagerClient,
)
from stream_manager.util import Util

client = StreamManagerClient()

try:
    # Read the statuses from the export status stream
    is_file_uploaded_to_s3 = False
    while not is_file_uploaded_to_s3:
        try:
            messages_list = client.read_messages(
                "StatusStreamName", ReadMessagesOptions(min_message_count=1,
read_timeout_millis=1000)
            )
            for message in messages_list:
                # Deserialize the status message first.
                status_message = Util.deserialize_json_bytes_to_obj(message.payload,
StatusMessage)
```

```

        # Check the status of the status message. If the status is
        "Success",
        # the file was successfully uploaded to S3.
        # If the status was either "Failure" or "Cancelled", the server was
        unable to upload the file to S3.
        # We will print the message for why the upload to S3 failed from the
        status message.
        # If the status was "InProgress", the status indicates that the
        server has started uploading
        # the S3 task.
        if status_message.status == Status.Success:
            logger.info("Successfully uploaded file at path " + file_url + "
to S3.")
            is_file_uploaded_to_s3 = True
        elif status_message.status == Status.Failure or
status_message.status == Status.Canceled:
            logger.info(
                "Unable to upload file at path " + file_url + " to S3.
Message: " + status_message.message
            )
            is_file_uploaded_to_s3 = True
        time.sleep(5)
    except StreamManagerException:
        logger.exception("Exception while running")
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
    # Properly handle errors.

```

Référence du SDK Python : [read_messages](#) | [StatusMessage](#)

Java

```

import com.amazonaws.greengrass.streammanager.client.StreamManagerClient;
import com.amazonaws.greengrass.streammanager.client.StreamManagerClientFactory;
import com.amazonaws.greengrass.streammanager.client.utils.ValidateAndSerialize;
import com.amazonaws.greengrass.streammanager.model.ReadMessagesOptions;
import com.amazonaws.greengrass.streammanager.model.Status;
import com.amazonaws.greengrass.streammanager.model.StatusConfig;
import com.amazonaws.greengrass.streammanager.model.StatusLevel;
import com.amazonaws.greengrass.streammanager.model.StatusMessage;

```

```
try (final StreamManagerClient client =
StreamManagerClientFactory.standard().build()) {
    try {
        boolean isS3UploadComplete = false;
        while (!isS3UploadComplete) {
            try {
                // Read the statuses from the export status stream
                List<Message> messages = client.readMessages("StatusStreamName",
                    new
ReadMessagesOptions().withMinMessageCount(1L).withReadTimeoutMillis(1000L));
                for (Message message : messages) {
                    // Deserialize the status message first.
                    StatusMessage statusMessage =
ValidateAndSerialize.deserializeJsonBytesToObj(message.getPayload(),
StatusMessage.class);
                    // Check the status of the status message. If the status is
"Success", the file was successfully uploaded to S3.
                    // If the status was either "Failure" or "Canceled", the server
was unable to upload the file to S3.
                    // We will print the message for why the upload to S3 failed
from the status message.
                    // If the status was "InProgress", the status indicates that the
server has started uploading the S3 task.
                    if (Status.Success.equals(statusMessage.getStatus())) {
                        System.out.println("Successfully uploaded file at path " +
FILE_URL + " to S3.");
                        isS3UploadComplete = true;
                    } else if (Status.Failure.equals(statusMessage.getStatus()) ||
Status.Canceled.equals(statusMessage.getStatus())) {
                        System.out.println(String.format("Unable to upload file at
path %s to S3. Message %s",
statusMessage.getStatusContext().getS3ExportTaskDefinition().getInputUrl(),
statusMessage.getMessage()));
                        isS3UploadComplete = 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 SDK Java : [ReadMessages](#) | [StatusMessage](#)

Node.js

```
const {
    StreamManagerClient, ReadMessagesOptions,
    Status, StatusConfig, StatusLevel, StatusMessage,
    util,
} = require('*aws-greengrass-stream-manager-sdk*');

const client = new StreamManagerClient();
client.onConnected(async () => {
    try {
        let isS3UploadComplete = false;
        while (!isS3UploadComplete) {
            try {
                // Read the statuses from the export status stream
                const messages = await c.readMessages("StatusStreamName",
                    new ReadMessagesOptions()
                        .withMinMessageCount(1)
                        .withReadTimeoutMillis(1000));

                messages.forEach((message) => {
                    // Deserialize the status message first.
                    const statusMessage =
                        util.deserializeJsonBytesToObj(message.payload, StatusMessage);
                    // Check the status of the status message. If the status is
                    // 'Success', the file was successfully uploaded to S3.
                    // If the status was either 'Failure' or 'Cancelled', the server
                    // was unable to upload the file to S3.
                    // We will print the message for why the upload to S3 failed
                    // from the status message.
                    // If the status was "InProgress", the status indicates that the
                    // server has started uploading the S3 task.
                    if (statusMessage.status === Status.Success) {
                        console.log(`Successfully uploaded file at path ${FILE_URL}
to S3.`);
                        isS3UploadComplete = true;
                    }
                });
            } catch (e) {
                // Handle exception
            }
        }
    } catch (e) {
        // Handle exception
    }
});
```

```
        } 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 SDK Node.js : [ReadMessages](#) | [StatusMessage](#)

Exécuter l'inférence de Machine Learning

Avec AWS IoT Greengrass, vous pouvez effectuer des inférences d'apprentissage automatique (ML) sur vos appareils périphériques à partir de données générées localement à l'aide de modèles conçus 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.

AWS IoT Greengrass rend les étapes nécessaires à l'inférence plus efficaces. Vous pouvez entraîner vos modèles d'inférence n'importe où et les déployer localement en tant que composants d'apprentissage automatique. Par exemple, vous pouvez créer et entraîner des modèles d'apprentissage profond dans [Amazon SageMaker AI](#). Vous pouvez ensuite stocker ces modèles dans un compartiment [Amazon S3](#), afin de pouvoir les utiliser comme artefacts dans vos composants pour effectuer des inférences sur vos appareils principaux.

Rubriques

- [Comment fonctionne l'inférence AWS IoT Greengrass ML](#)
- [Qu'est-ce qui est différent dans AWS IoT Greengrass la version 2 ?](#)
- [Prérequis](#)
- [Sources de modèles prises en charge](#)
- [Runtimes d'apprentissage automatique pris en charge](#)
- [AWS-composants d'apprentissage automatique fournis](#)
- [Utiliser Amazon SageMaker AI Edge Manager sur les appareils principaux de Greengrass](#)
- [Personnalisez vos composants d'apprentissage automatique](#)
- [Résolution des problèmes liés à l'inférence par apprentissage automatique](#)

Comment fonctionne l'inférence AWS IoT Greengrass ML

AWS fournit des [composants d'apprentissage automatique](#) que vous pouvez utiliser pour créer des déploiements en une étape afin d'effectuer des inférences d'apprentissage automatique sur votre appareil. Vous pouvez également utiliser ces composants comme modèles pour créer des composants personnalisés répondant à vos besoins spécifiques.

AWS fournit les catégories suivantes de composants d'apprentissage automatique :

- Composant du modèle : contient des modèles d'apprentissage automatique sous forme d'artefacts Greengrass.
- Composant d'exécution : contient le script qui installe le framework d'apprentissage automatique et ses dépendances sur le périphérique principal de Greengrass.
- Composant d'inférence : contient le code d'inférence et inclut les dépendances des composants pour installer le framework d'apprentissage automatique et télécharger des modèles d'apprentissage automatique préentraînés.

Chaque déploiement que vous créez pour effectuer une inférence d'apprentissage automatique comprend au moins un composant qui exécute votre application d'inférence, installe l'infrastructure d'apprentissage automatique et télécharge vos modèles d'apprentissage automatique. Pour effectuer un exemple d'inférence avec les composants AWS fournis, vous déployez un composant d'inférence sur votre périphérique principal, qui inclut automatiquement le modèle et les composants d'exécution correspondants en tant que dépendances. Pour personnaliser vos déploiements, vous pouvez intégrer ou remplacer les composants du modèle d'exemple par des composants de modèle personnalisés, ou vous pouvez utiliser les recettes de composants pour les composants AWS fournis comme modèles pour créer vos propres composants d'inférence, de modèle et d'exécution personnalisés.

Pour effectuer une inférence d'apprentissage automatique à l'aide de composants personnalisés :

1. Créez un composant de modèle. Ce composant contient les modèles d'apprentissage automatique que vous souhaitez utiliser pour effectuer des inférences. AWS fournit des exemples de modèles DLR et TensorFlow Lite préentraînés. Pour utiliser un modèle personnalisé, créez votre propre composant de modèle.
2. Créez un composant d'exécution. Ce composant contient les scripts nécessaires pour installer le moteur d'apprentissage automatique pour vos modèles. AWS fournit des exemples de composants d'exécution pour [Deep Learning Runtime](#) (DLR) et [TensorFlow Lite](#). Pour utiliser d'autres environnements d'exécution avec vos modèles personnalisés et votre code d'inférence, créez vos propres composants d'exécution.
3. Créez un composant d'inférence. Ce composant contient votre code d'inférence et inclut votre modèle et vos composants d'exécution en tant que dépendances. AWS fournit des exemples de composants d'inférence pour la classification d'images et la détection d'objets à l'aide de DLR et TensorFlow Lite. Pour effectuer d'autres types d'inférence ou pour utiliser des modèles et des environnements d'exécution personnalisés, créez votre propre composant d'inférence.

4. Déployez le composant d'inférence. Lorsque vous déployez ce composant, il déploie AWS IoT Greengrass également automatiquement les dépendances du modèle et du composant d'exécution.

Pour commencer à utiliser les composants AWS fournis, consultez [the section called “Effectuer une inférence de classification d'images d'échantillons”](#).

Pour plus d'informations sur la création de composants d'apprentissage automatique personnalisés, consultez [Personnalisez vos composants d'apprentissage automatique](#).

Qu'est-ce qui est différent dans AWS IoT Greengrass la version 2 ?

AWS IoT Greengrass consolide les unités fonctionnelles pour l'apprentissage automatique, telles que les modèles, les environnements d'exécution et le code d'inférence, en composants qui vous permettent d'utiliser un processus en une étape pour installer le moteur d'apprentissage automatique, télécharger vos modèles entraînés et effectuer des inférences sur votre appareil.

En utilisant les composants d'apprentissage automatique AWS fournis, vous avez la possibilité de commencer à effectuer des inférences par apprentissage automatique à l'aide d'exemples de code d'inférence et de modèles préentraînés. Vous pouvez intégrer des composants de modèles personnalisés pour utiliser vos propres modèles entraînés sur mesure avec les composants d'inférence et d'exécution fournis. AWS Pour une solution d'apprentissage automatique entièrement personnalisée, vous pouvez utiliser les composants publics comme modèles pour créer des composants personnalisés et utiliser le type d'exécution, de modèle ou d'inférence de votre choix.

Prérequis

Pour créer et utiliser des composants d'apprentissage automatique, vous devez disposer des éléments suivants :

- Un appareil Greengrass Core. Si vous n'en avez pas, veuillez consulter [Tutoriel : Débuter avec AWS IoT Greengrass V2](#).
- 500 Mo d'espace de stockage local minimum pour utiliser les exemples de composants d'apprentissage automatique AWS fournis.

Sources de modèles prises en charge

AWS IoT Greengrass prend en charge l'utilisation de modèles d'apprentissage automatique formés sur mesure qui sont stockés dans Amazon S3. Vous pouvez également utiliser les tâches d'empaquetage Amazon SageMaker AI Edge pour créer directement des composants de modèle pour vos modèles compilés par SageMaker AI Neo. Pour plus d'informations sur l'utilisation d'SageMaker AI Edge Manager avec AWS IoT Greengrass, consultez [Utiliser Amazon SageMaker AI Edge Manager sur les appareils principaux de Greengrass](#).

Les compartiments S3 qui contiennent vos modèles doivent répondre aux exigences suivantes :

- Ils ne doivent pas être chiffrés à l'aide du protocole SSE-C. Pour les buckets qui utilisent le chiffrement côté serveur, l'inférence basée sur l'apprentissage AWS IoT Greengrass automatique prend actuellement en charge uniquement les options de chiffrement SSE-S3 ou SSE-KMS. Pour plus d'informations sur les options de chiffrement côté serveur, consultez la section [Protection des données à l'aide du chiffrement côté serveur dans](#) le guide de l'utilisateur d'Amazon Simple Storage Service.
- Leurs noms ne doivent pas inclure de points (.). Pour plus d'informations, consultez la règle concernant l'utilisation de compartiments de type hébergé virtuel avec SSL dans [Règles de dénomination des compartiments du guide de l'utilisateur](#) d'Amazon Simple Storage Service.
- Les compartiments S3 qui stockent les sources de vos modèles doivent se trouver dans Compte AWS les mêmes compartiments Région AWS que vos composants d'apprentissage automatique.
- AWS IoT Greengrass doit avoir read l'autorisation d'accéder à la source du modèle. Pour permettre l'accès AWS IoT Greengrass aux compartiments S3, le rôle d'[appareil Greengrass](#) doit autoriser `s3:GetObject` l'action. Pour plus d'informations sur le rôle de l'appareil, consultez [Autoriser les appareils principaux à interagir avec les AWS services](#).

Runtimes d'apprentissage automatique pris en charge

AWS IoT Greengrass vous permet de créer des composants personnalisés pour utiliser n'importe quel environnement d'apprentissage automatique de votre choix pour effectuer des inférences d'apprentissage automatique avec vos modèles entraînés sur mesure. Pour plus d'informations sur la création de composants d'apprentissage automatique personnalisés, consultez [Personnalisez vos composants d'apprentissage automatique](#).

Pour rendre le processus de démarrage du machine learning plus efficace, AWS IoT Greengrass fournit des exemples de composants d'inférence, de modèle et d'exécution qui utilisent les environnements d'exécution d'apprentissage automatique suivants :

- [Deep Learning Runtime](#) (DLR) v1.6.0 et v1.3.0
- [TensorFlow Lite v2.5.0](#)

AWS-composants d'apprentissage automatique fournis

Le tableau suivant répertorie les composants AWS fournis utilisés pour l'apprentissage automatique.

Note

Plusieurs composants AWS fournis dépendent de versions mineures spécifiques du noyau Greengrass. En raison de cette dépendance, vous devez mettre à jour ces composants lorsque vous mettez à jour le noyau Greengrass vers une nouvelle version mineure. Pour plus d'informations sur les versions spécifiques du noyau dont dépend chaque composant, consultez la rubrique correspondante sur les composants. Pour plus d'informations sur la mise à jour du noyau, consultez [Mettre à jour le logiciel AWS IoT Greengrass principal \(OTA\)](#).

Composant	Description	Type de composant	Systèmes d'exploitation pris en charge	Open source
SageMaker Gestionnaire AI Edge	Déploie l'agent Amazon SageMaker AI Edge Manager sur l'appareil principal de Greengrass.	Générique	Linux, Windows	Non
Classification des images DLR	Composant d'inférence qui utilise	Générique	Linux, Windows	Non

Composant	Description	Type de composant	Systèmes d'exploitation pris en charge	Open source
	le magasin de modèles de classification d'images DLR et le composant d'exécution DLR comme dépendances pour installer le DLR, télécharger des exemples de modèles de classification d'images et effectuer une inférence de classification d'images sur les appareils pris en charge.			

Composant	Description	Type de composant	Systèmes d'exploitation pris en charge	Open source
Détection d'objets DLR	Composant d'inférence qui utilise le model store de détection d'objets DLR et le composant d'exécution DLR comme dépendances pour installer le DLR, télécharger des exemples de modèles de détection d'objets et effectuer une inférence de détection d'objets sur les appareils pris en charge.	Générique	Linux, Windows	Non

Composant	Description	Type de composant	Systèmes d'exploitation pris en charge	Open source
magasin de modèles de classification d'images DLR	Composant de modèle contenant des exemples de ResNet 50 modèles de classification d'images sous forme d'artefacts Greengrass.	Générique	Linux, Windows	Non
Model Store dédié à la détection d'objets DLR	Composant de modèle contenant des exemples de modèles de détection YOLOv3 d'objets sous forme d'artefacts Greengrass.	Générique	Linux, Windows	Non

Composant	Description	Type de composant	Systèmes d'exploitation pris en charge	Open source
Temps d'exécution du DLR	Composant d'exécution contenant un script d'installation utilisé pour installer le DLR et ses dépendances sur le périphérique principal de Greengrass.	Générique	Linux, Windows	Non

Composant	Description	Type de composant	Systèmes d'exploitation pris en charge	Open source
TensorFlow Classification d'images Lite	Composant d'inférence qui utilise le magasin de modèles de classification d'images TensorFlow Lite et le composant d'exécution TensorFlow Lite comme dépendances pour installer TensorFlow Lite, télécharger des exemples de modèles de classification d'images et effectuer une inférence de classification d'images sur les appareils pris en charge.	Générique	Linux, Windows	Non

Composant	Description	Type de composant	Systèmes d'exploitation pris en charge	Open source
TensorFlow Détection d'objets allégée	Composant d'inférence qui utilise le magasin de modèles de détection d'objets TensorFlow Lite et le composant d'exécution TensorFlow Lite comme dépendances pour installer TensorFlow Lite, télécharger des exemples de modèles de détection d'objets et effectuer une inférence de détection d'objets sur les appareils pris en charge.	Générique	Linux, Windows	Non

Composant	Description	Type de composant	Systèmes d'exploitation pris en charge	Open source
TensorFlow Boutique de modèles de classification d'images Lite	Composant de modèle contenant un exemple de modèle MobileNet v1 en tant qu'artefact Greengrass.	Générique	Linux, Windows	Non
TensorFlow Boutique de modèles de détection d'objets Lite	Composant de modèle contenant un exemple de MobileNet modèle de détection par injection unique (SSD) sous forme d'artefact Greengrass.	Générique	Linux, Windows	Non

Composant	Description	Type de composant	Systèmes d'exploitation pris en charge	Open source
TensorFlow Temps d'exécution allégé	Composant d'exécution contenant un script d'installation utilisé pour installer TensorFlow Lite et ses dépendances sur le périphérique principal de Greengrass.	Générique	Linux, Windows	Non

Utiliser Amazon SageMaker AI Edge Manager sur les appareils principaux de Greengrass

Important

SageMaker AI Edge Manager a été abandonné le 26 avril 2024. Pour plus d'informations sur la poursuite du déploiement de vos modèles sur des appareils Edge, consultez [SageMaker AI Edge Manager end of life](#).

Amazon SageMaker AI Edge Manager est un agent logiciel qui s'exécute sur des appareils périphériques. SageMaker AI Edge Manager assure la gestion des modèles pour les appareils de pointe afin que vous puissiez empaqueter et utiliser les modèles compilés par Amazon SageMaker AI Neo directement sur les appareils principaux de Greengrass. En utilisant SageMaker AI Edge Manager, vous pouvez également échantillonner les données d'entrée et de sortie du modèle à partir de vos principaux appareils, et envoyer ces données à des AWS Cloud fins de surveillance

et d'analyse. Dans la SageMaker mesure où AI Edge Manager utilise SageMaker AI Neo pour optimiser vos modèles en fonction de votre matériel cible, vous n'avez pas besoin d'installer le moteur d'exécution DLR directement sur votre appareil. Sur les appareils Greengrass, SageMaker AI Edge Manager ne charge pas les AWS IoT certificats locaux et n'appelle pas directement le point de terminaison du fournisseur AWS IoT d'informations d'identification. SageMaker AI Edge Manager utilise plutôt le [service d'échange de jetons](#) pour récupérer des informations d'identification temporaires depuis un point de terminaison TES.

Cette section décrit le fonctionnement d' SageMaker AI Edge Manager sur les appareils principaux de Greengrass.

Comment fonctionne SageMaker AI Edge Manager sur les appareils Greengrass

Pour déployer l'agent SageMaker AI Edge Manager sur vos appareils principaux, créez un déploiement incluant le `aws.greengrass.SageMakerEdgeManager` composant. AWS IoT Greengrass gère l'installation et le cycle de vie de l'agent Edge Manager sur vos appareils. Lorsqu'une nouvelle version du binaire de l'agent est disponible, déployez la version mise à jour du `aws.greengrass.SageMakerEdgeManager` composant pour mettre à niveau la version de l'agent installée sur votre appareil.

Lorsque vous utilisez SageMaker AI Edge Manager avec AWS IoT Greengrass, votre flux de travail inclut les étapes de haut niveau suivantes :

1. Compilez des modèles avec SageMaker AI Neo.
2. Packagez vos modèles compilés SageMaker AI Neo à l'aide de tâches d'emballage SageMaker AI Edge. Lorsque vous exécutez une tâche d'emballage périphérique pour votre modèle, vous pouvez choisir de créer un composant de modèle avec le modèle emballé en tant qu'artefact qui peut être déployé sur votre appareil principal Greengrass.
3. Créez un composant d'inférence personnalisé. Vous utilisez ce composant d'inférence pour interagir avec l'agent Edge Manager afin d'effectuer une inférence sur le périphérique principal. Ces opérations incluent le chargement de modèles, l'appel de demandes de prédiction pour exécuter des inférences et le déchargement de modèles lorsque le composant s'arrête.
4. Déployez le composant SageMaker AI Edge Manager, le composant de modèle intégré et le composant d'inférence pour exécuter votre modèle sur le moteur d'inférence SageMaker AI (agent Edge Manager) de votre appareil.

Pour plus d'informations sur la création de tâches d'empaquetage Edge et de composants d'inférence compatibles avec SageMaker AI Edge Manager, consultez la section [Deploy Model Package et Edge Manager Agent with AWS IoT Greengrass](#) dans le manuel Amazon SageMaker AI Developer Guide.

Le [Tutoriel : Démarrez avec SageMaker AI Edge Manager](#) didacticiel explique comment configurer et utiliser l'agent SageMaker AI Edge Manager sur un appareil principal Greengrass existant, à l'aide d'un exemple de code AWS fourni que vous pouvez utiliser pour créer des exemples d'inférence et des composants de modèle.

Lorsque vous utilisez SageMaker AI Edge Manager sur les appareils principaux de Greengrass, vous pouvez également utiliser la fonction de capture de données pour télécharger des exemples de données sur le. AWS Cloud La capture de données est une fonctionnalité d' SageMaker intelligence artificielle que vous utilisez pour télécharger des entrées d'inférence, des résultats d'inférence et des données d'inférence supplémentaires dans un compartiment S3 ou un répertoire local en vue d'une analyse future. Pour plus d'informations sur l'utilisation des données de capture avec SageMaker AI Edge Manager, consultez [Manage Model](#) dans le manuel Amazon SageMaker AI Developer Guide.

Prérequis

Vous devez satisfaire aux exigences suivantes pour utiliser l'agent SageMaker AI Edge Manager sur les appareils principaux de Greengrass.

- Un appareil Greengrass principal fonctionnant sous Amazon Linux 2, une plate-forme Linux basée sur Debian (x86_64 ou Armv8) ou Windows (x86_64). Si vous n'en avez pas, veuillez consulter [Tutoriel : Débuter avec AWS IoT Greengrass V2](#).
- [Python](#) 3.6 ou version ultérieure, y compris pip pour votre version de Python, installé sur votre appareil principal.
- Le [rôle d'appareil Greengrass](#) est configuré comme suit :
 - Une relation de confiance qui permet `credentials.iot.amazonaws.com` et permet `sagemaker.amazonaws.com` d'assumer le rôle, comme le montre l'exemple de politique IAM suivant.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "credentials.iot.amazonaws.com"
```

```

    },
    "Action": "sts:AssumeRole"
  },
  {
    "Effect": "Allow",
    "Principal": {
      "Service": "sagemaker.amazonaws.com"
    },
    "Action": "sts:AssumeRole"
  }
]
}

```

- La politique gérée par [AmazonSageMakerEdgeDeviceFleetPolicy](#) IAM.
- L'action `s3:PutObject`, comme illustré dans l'exemple de politique IAM suivant.

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "s3:PutObject"
      ],
      "Resource": [
        "*"
      ],
      "Effect": "Allow"
    }
  ]
}

```

- Un bucket Amazon S3 créé en même temps que le Compte AWS et en même temps que la Région AWS que votre appareil principal Greengrass. SageMaker AI Edge Manager nécessite un compartiment S3 pour créer un parc d'appareils Edge et pour stocker des exemples de données provenant de l'exécution d'inférences sur votre appareil. Pour plus d'informations sur la création de compartiments S3, consultez [Getting started with Amazon S3](#).
- Un parc d'appareils SageMaker AI Edge qui utilise le même alias de rôle AWS IoT que votre appareil principal Greengrass. Pour de plus amples informations, veuillez consulter [Créez un parc d'appareils de pointe](#).
- Votre appareil principal Greengrass est enregistré en tant qu'appareil Edge dans votre parc d'appareils SageMaker AI Edge. Le nom de l'appareil Edge doit correspondre au nom de l'

AWS IoT objet de votre appareil principal. Pour de plus amples informations, veuillez consulter [Enregistrez votre appareil Greengrass Core](#).

Commencez avec SageMaker AI Edge Manager

Vous pouvez suivre un didacticiel pour commencer à utiliser SageMaker AI Edge Manager. Le didacticiel vous montre comment commencer à utiliser SageMaker AI Edge Manager avec des exemples de composants AWS fournis sur un périphérique principal existant. Ces exemples de composants utilisent le composant SageMaker AI Edge Manager comme dépendance pour déployer l'agent Edge Manager et effectuer des inférences à l'aide de modèles préentraînés compilés à l'aide d' SageMaker AI Neo. Pour de plus amples informations, veuillez consulter [Tutoriel : Démarrez avec SageMaker AI Edge Manager](#).

Personnalisez vos composants d'apprentissage automatique

Dans AWS IoT Greengrass, vous pouvez configurer des exemples de [composants d'apprentissage automatique](#) pour personnaliser la manière dont vous effectuez l'inférence d'apprentissage automatique sur vos appareils en utilisant les composants d'inférence, de modèle et d'exécution comme éléments de base. AWS IoT Greengrass vous offre également la possibilité d'utiliser les exemples de composants comme modèles et de créer vos propres composants personnalisés selon vos besoins. Vous pouvez combiner cette approche modulaire pour personnaliser vos composants d'inférence d'apprentissage automatique de la manière suivante :

Utilisation d'échantillons de composants d'inférence

- Modifiez la configuration des composants d'inférence lorsque vous les déployez.
- Utilisez un modèle personnalisé avec le composant d'inférence d'exemple en remplaçant l'exemple de composant de magasin de modèles par un composant de modèle personnalisé. Votre modèle personnalisé doit être entraîné en utilisant le même environnement d'exécution que le modèle d'exemple.

Utilisation de composants d'inférence personnalisés

- Utilisez un code d'inférence personnalisé avec les exemples de modèles et d'environnements d'exécution en ajoutant des composants de modèles publics et des composants d'exécution en tant que dépendances des composants d'inférence personnalisés.
- Créez et ajoutez des composants de modèle personnalisés ou des composants d'exécution en tant que dépendances de composants d'inférence personnalisés. Vous devez utiliser des

composants personnalisés si vous souhaitez utiliser un code d'inférence personnalisé ou un environnement d'exécution pour lequel aucun exemple de composant AWS IoT Greengrass n'est fourni.

Rubriques

- [Modifier la configuration d'un composant d'inférence public](#)
- [Utiliser un modèle personnalisé avec le composant d'inférence d'échantillon](#)
- [Création de composants d'apprentissage automatique personnalisés](#)
- [Création d'un composant d'inférence personnalisé](#)

Modifier la configuration d'un composant d'inférence public

Dans la [AWS IoT Greengrass console](#), la page du composant affiche la configuration par défaut de ce composant. Par exemple, la configuration par défaut du composant de classification d'images TensorFlow Lite est la suivante :

```
{
  "accessControl": {
    "aws.greengrass.ipc.mqttproxy": {
      "aws.greengrass.TensorFlowLiteImageClassification:mqttproxy:1": {
        "policyDescription": "Allows access to publish via topic ml/tflite/image-classification.",
        "operations": [
          "aws.greengrass#PublishToIoTCore"
        ],
        "resources": [
          "ml/tflite/image-classification"
        ]
      }
    }
  },
  "PublishResultsOnTopic": "ml/tflite/image-classification",
  "ImageName": "cat.jpeg",
  "InferenceInterval": 3600,
  "ModelResourceKey": {
    "model": "TensorFlowLite-Mobilenet"
  }
}
```

Lorsque vous déployez un composant d'inférence public, vous pouvez modifier la configuration par défaut pour personnaliser votre déploiement. Pour plus d'informations sur les paramètres de configuration disponibles pour chaque composant d'inférence public, consultez la rubrique relative au composant dans [AWS-composants d'apprentissage automatique fournis](#).

Cette section décrit comment déployer un composant modifié depuis la AWS IoT Greengrass console. Pour plus d'informations sur le déploiement de composants à l'aide du AWS CLI, consultez [Créer des déploiements](#).

Pour déployer un composant d'inférence publique modifié (console)

1. Connectez-vous à la [console AWS IoT Greengrass](#).
2. Dans le menu de navigation, sélectionnez Composants.
3. Sur la page Composants, sous l'onglet Composants publics, choisissez le composant que vous souhaitez déployer.
4. Sur la page du composant, choisissez Deploy.
5. Dans Ajouter au déploiement, sélectionnez l'une des options suivantes :
 - a. Pour fusionner ce composant avec un déploiement existant sur votre dispositif cible, choisissez Add to existing deployment (Ajouter à un déploiement existant), puis sélectionnez le déploiement à réviser.
 - b. Pour créer un nouveau déploiement sur votre dispositif cible, choisissez Create new deployment (Créer un déploiement). S'il existe un déploiement sur votre dispositif et que vous choisissez cette étape, le déploiement existant sera remplacé.
6. Sur la page Specify target (Spécifier une cible), procédez comme suit :
 - a. Sous Deployment information (Informations sur le déploiement), saisissez ou modifiez le nom convivial de votre déploiement.
 - b. Sous Deployment targets (Cibles de déploiement), sélectionnez une cible pour votre déploiement, puis choisissez Next (Suivant). Vous ne pouvez pas modifier la cible de déploiement si vous révisez un déploiement existant.
7. Sur la page Sélectionner les composants, sous Composants publics, vérifiez que le composant d'inférence avec votre configuration modifiée est sélectionné, puis choisissez Next.
8. Sur la page Configurer les composants, procédez comme suit :
 - a. Sélectionnez le composant d'inférence, puis sélectionnez Configurer le composant.

- b. Sous Mise à jour de la configuration, entrez les valeurs de configuration que vous souhaitez mettre à jour. Par exemple, entrez la mise à jour de configuration suivante dans la zone Configuration à fusionner pour modifier l'intervalle d'inférence à 15 secondes, et demandez au composant de rechercher l'image nommée `custom.jpg` dans le `/custom-ml-inference/images/` dossier.

```
{
  "InferenceInterval": "15",
  "ImageName": "custom.jpg",
  "ImageDirectory": "/custom-ml-inference/images/"
}
```

Pour rétablir les valeurs par défaut de l'ensemble de la configuration d'un composant, spécifiez une seule chaîne vide `""` dans le champ Réinitialiser les chemins.

- c. Choisissez Confirm (Confirmer), puis Next (Suivant).
9. Sur la page Configurer les paramètres avancés, conservez les paramètres de configuration par défaut et choisissez Next.
10. Sur la page de révision, choisissez Déployer

Utiliser un modèle personnalisé avec le composant d'inférence d'échantillon

Si vous souhaitez utiliser le composant d'inférence d'exemple avec vos propres modèles d'apprentissage automatique pour un environnement d'exécution AWS IoT Greengrass fournissant un exemple de composant d'exécution, vous devez remplacer les composants du modèle public par des composants qui utilisent ces modèles comme artefacts. À un niveau élevé, vous devez suivre les étapes suivantes pour utiliser un modèle personnalisé avec le composant d'inférence d'échantillon :

1. Créez un composant de modèle qui utilise un modèle personnalisé dans un compartiment S3 comme artefact. Votre modèle personnalisé doit être entraîné en utilisant le même environnement d'exécution que le modèle que vous souhaitez remplacer.
2. Modifiez le paramètre `ModelResourceKey` de configuration dans le composant d'inférence pour utiliser le modèle personnalisé. Pour plus d'informations sur la mise à jour de la configuration du composant d'inférence, voir [Modifier la configuration d'un composant d'inférence public](#)

Lorsque vous déployez le composant d'inférence AWS IoT Greengrass, recherchez la dernière version de ses dépendances. Il remplace le composant du modèle public dépendant si une version personnalisée ultérieure du composant existe dans le même Compte AWS et Région AWS.

Création d'un composant de modèle personnalisé (console)

1. Téléchargez votre modèle dans un compartiment S3. Pour plus d'informations sur le téléchargement de vos modèles dans un compartiment S3, consultez la section [Utilisation des compartiments Amazon S3](#) dans le guide de l'utilisateur d'Amazon Simple Storage Service.

Note

Vous devez stocker vos artefacts dans des compartiments S3 situés en même temps Région AWS que Compte AWS les composants. Pour permettre AWS IoT Greengrass l'accès à ces artefacts, le [rôle de l'appareil Greengrass](#) doit autoriser l'`s3:GetObject`. Pour plus d'informations sur le rôle de l'appareil, consultez [Autoriser les appareils principaux à interagir avec les AWS services](#).

2. Dans le menu de navigation de la [AWS IoT Greengrass console](#), sélectionnez Composants.
3. Récupérez la recette du composant public du model store.
 - a. Sur la page Composants, sous l'onglet Composants publics, recherchez et choisissez le composant du modèle public pour lequel vous souhaitez créer une nouvelle version. Par exemple, `variant.DLR.ImageClassification.ModelStore`.
 - b. Sur la page du composant, choisissez Afficher la recette et copiez la recette JSON affichée.
4. Sur la page Composants, dans l'onglet Mes composants, choisissez Créer un composant.
5. Sur la page Créer un composant, sous Informations sur le composant, sélectionnez Entrer la recette au format JSON comme source de composant.
6. Dans le champ Recette, collez la recette du composant que vous avez précédemment copiée.
7. Dans la recette, mettez à jour les valeurs suivantes :
 - `ComponentVersion`: incrémente la version secondaire du composant.

Lorsque vous créez un composant personnalisé pour remplacer un composant de modèle public, vous devez uniquement mettre à jour la version secondaire de la version du composant existant. Par exemple, si la version du composant public est `2.1.0`, vous pouvez créer un composant personnalisé avec la version `2.1.1`.

- `Manifests.Artifacts.Uri`: mettez à jour chaque valeur d'URI en fonction de l'URI Amazon S3 du modèle que vous souhaitez utiliser.

 Note

Ne modifiez pas le nom du composant.

8. Choisissez Créer un composant.

Création d'un composant de modèle personnalisé (AWS CLI)

1. Téléchargez votre modèle dans un compartiment S3. Pour plus d'informations sur le téléchargement de vos modèles dans un compartiment S3, consultez la section [Utilisation des compartiments Amazon S3](#) dans le guide de l'utilisateur d'Amazon Simple Storage Service.

 Note

Vous devez stocker vos artefacts dans des compartiments S3 situés en même temps Région AWS que Compte AWS les composants. Pour permettre AWS IoT Greengrass l'accès à ces artefacts, le [rôle de l'appareil Greengrass](#) doit autoriser `s3:GetObject`. Pour plus d'informations sur le rôle de l'appareil, consultez [Autoriser les appareils principaux à interagir avec les AWS services](#).

2. Exécutez la commande suivante pour récupérer la recette du composant public. Cette commande écrit la recette du composant dans le fichier de sortie que vous fournissez dans votre commande. Convertissez la chaîne codée en base64 récupérée en JSON ou YAML, selon vos besoins.

Linux, macOS, or Unix

```
aws greengrassv2 get-component \  
  --arn <arn> \  
  --recipe-output-format <recipe-format> \  
  --query recipe \  
  --output text | base64 --decode > <recipe-file>
```


Windows Command Prompt (CMD)

```
aws greengrassv2 get-component ^  
  --arn <arn> ^  
  --recipe-output-format <recipe-format> ^  
  --query recipe ^  
  --output text > <recipe-file>.base64  
  
certutil -decode <recipe-file>.base64 <recipe-file>
```

PowerShell

```
aws greengrassv2 get-component `  
  --arn <arn> `  
  --recipe-output-format <recipe-format> `  
  --query recipe `  
  --output text > <recipe-file>.base64  
  
certutil -decode <recipe-file>.base64 <recipe-file>
```

3. Mettez à jour le nom du fichier de recette en **<component-name>-<component-version>**, où la version du composant est la version cible du nouveau composant. Par exemple, `variant.DLR.ImageClassification.ModelStore-2.1.1.yaml`.
4. Dans la recette, mettez à jour les valeurs suivantes :
 - `ComponentVersion`: incrémente la version secondaire du composant.

Lorsque vous créez un composant personnalisé pour remplacer un composant de modèle public, vous devez uniquement mettre à jour la version secondaire de la version du composant existant. Par exemple, si la version du composant public est `2.1.0`, vous pouvez créer un composant personnalisé avec la version `2.1.1`.

- `Manifests.Artifacts.Uri`: mettez à jour chaque valeur d'URI en fonction de l'URI Amazon S3 du modèle que vous souhaitez utiliser.

Note

Ne modifiez pas le nom du composant.

5. Exécutez la commande suivante pour créer un nouveau composant à l'aide de la recette que vous avez récupérée et modifiée.

```
aws greengrassv2 create-component-version \  
  --inline-recipe fileb://path/to/component/recipe
```

Note

Cette étape crée le composant dans le AWS IoT Greengrass service dans le AWS Cloud. Vous pouvez utiliser la CLI Greengrass pour développer, tester et déployer votre composant localement avant de le télécharger dans le cloud. Pour de plus amples informations, veuillez consulter [Développer des AWS IoT Greengrass composants](#).

Pour plus d'informations sur la création de composants, consultez [Développer des AWS IoT Greengrass composants](#).

Création de composants d'apprentissage automatique personnalisés

Vous devez créer des composants personnalisés si vous souhaitez utiliser un code d'inférence personnalisé ou un environnement d'exécution pour lequel aucun exemple de composant AWS IoT Greengrass n'est fourni. Vous pouvez utiliser votre code d'inférence personnalisé avec les exemples de modèles et d'environnements d'exécution d'apprentissage automatique AWS fournis, ou vous pouvez développer une solution d'inférence d'apprentissage automatique entièrement personnalisée avec vos propres modèles et environnements d'exécution. Si vos modèles utilisent un environnement d'exécution qui AWS IoT Greengrass fournit un exemple de composant d'exécution, vous pouvez utiliser ce composant d'exécution et vous devez créer des composants personnalisés uniquement pour votre code d'inférence et les modèles que vous souhaitez utiliser.

Rubriques

- [Récupérez la recette d'un composant public](#)
- [Récupérez des exemples d'artefacts de composants](#)
- [Télécharger des artefacts de composants dans un compartiment S3](#)
- [Création de composants personnalisés](#)

Récupérez la recette d'un composant public

Vous pouvez utiliser la recette d'un composant d'apprentissage automatique public existant comme modèle pour créer un composant personnalisé. Pour afficher la recette de composant de la dernière version d'un composant public, utilisez la console ou procédez AWS CLI comme suit :

- Utilisation de la console
 1. Sur la page Composants, sous l'onglet Composants publics, recherchez et sélectionnez le composant public.
 2. Sur la page du composant, choisissez Afficher la recette.
- En utilisant AWS CLI

Exécutez la commande suivante pour récupérer la recette du composant variant public. Cette commande écrit la recette du composant dans le fichier de recette JSON ou YAML que vous fournissez dans votre commande.

Linux, macOS, or Unix

```
aws greengrassv2 get-component \  
  --arn <arn> \  
  --recipe-output-format <recipe-format> \  
  --query recipe \  
  --output text | base64 --decode > <recipe-file>
```

Windows Command Prompt (CMD)

```
aws greengrassv2 get-component ^  
  --arn <arn> ^  
  --recipe-output-format <recipe-format> ^  
  --query recipe ^  
  --output text > <recipe-file>.base64  
  
certutil -decode <recipe-file>.base64 <recipe-file>
```

PowerShell

```
aws greengrassv2 get-component `  
  --arn <arn> `  
  --recipe-output-format <recipe-format> `  
  --query recipe `
```

```
--output text > <recipe-file>.base64  
  
certutil -decode <recipe-file>.base64 <recipe-file>
```

Remplacez les valeurs de votre commande comme suit :

- *<arn>*. Le nom de ressource Amazon (ARN) du composant public.
- *<recipe-format>*. Format dans lequel vous souhaitez créer le fichier de recette. Les valeurs prises en charge sont JSON et YAML.
- *<recipe-file>*. Le nom de la recette dans le format *<component-name>-<component-version>*.

Récupérez des exemples d'artefacts de composants

Vous pouvez utiliser les artefacts utilisés par les composants d'apprentissage automatique publics comme modèles pour créer vos artefacts de composants personnalisés, tels que du code d'inférence ou des scripts d'installation d'exécution.

Pour afficher les exemples d'artefacts inclus dans les composants publics d'apprentissage automatique, déployez le composant d'inférence public, puis affichez les artefacts sur votre appareil dans le */greengrass/v2/packages/artifacts-unarchived/component-name/component-version/* dossier.

Télécharger des artefacts de composants dans un compartiment S3

Avant de créer un composant personnalisé, vous devez télécharger les artefacts du composant dans un compartiment S3 et utiliser le S3 URIs dans la recette de votre composant. Par exemple, pour utiliser un code d'inférence personnalisé dans votre composant d'inférence, téléchargez le code dans un compartiment S3. Vous pouvez ensuite utiliser l'URI Amazon S3 de votre code d'inférence comme artefact dans votre composant.

Pour plus d'informations sur le téléchargement de contenu dans un compartiment S3, consultez la section [Utilisation des compartiments Amazon S3](#) dans le guide de l'utilisateur d'Amazon Simple Storage Service.

Note

Vous devez stocker vos artefacts dans des compartiments S3 situés en même temps Région AWS que Compte AWS les composants. Pour permettre AWS IoT Greengrass l'accès à ces

artefacts, le [rôle de l'appareil Greengrass](#) doit autoriser l'`s3:GetObject`. Pour plus d'informations sur le rôle de l'appareil, consultez [Autoriser les appareils principaux à interagir avec les AWS services](#).

Création de composants personnalisés

Vous pouvez utiliser les artefacts et les recettes que vous avez récupérés pour créer vos composants d'apprentissage automatique personnalisés. Pour obtenir un exemple, consultez [Création d'un composant d'inférence personnalisé](#).

Pour des informations détaillées sur la création et le déploiement de composants sur les appareils Greengrass, reportez-vous [Développer des AWS IoT Greengrass composants](#) aux sections et [Déployer AWS IoT Greengrass des composants sur des appareils](#)

Création d'un composant d'inférence personnalisé

Cette section explique comment créer un composant d'inférence personnalisé en utilisant le composant de classification d'images DLR comme modèle.

Rubriques

- [Téléchargez votre code d'inférence dans un compartiment Amazon S3](#)
- [Créez une recette pour votre composant d'inférence](#)
- [Création du composant d'inférence](#)

Téléchargez votre code d'inférence dans un compartiment Amazon S3

Créez votre code d'inférence, puis chargez-le dans un compartiment S3. Pour plus d'informations sur le téléchargement de contenu dans un compartiment S3, consultez la section [Utilisation des compartiments Amazon S3](#) dans le guide de l'utilisateur d'Amazon Simple Storage Service.

Note

Vous devez stocker vos artefacts dans des compartiments S3 situés en même temps Région AWS que Compte AWS les composants. Pour permettre AWS IoT Greengrass l'accès à ces artefacts, le [rôle de l'appareil Greengrass](#) doit autoriser l'`s3:GetObject`. Pour plus

d'informations sur le rôle de l'appareil, consultez [Autoriser les appareils principaux à interagir avec les AWS services](#).

Créez une recette pour votre composant d'inférence

1. Exécutez la commande suivante pour récupérer la recette du composant de classification d'images DLR. Cette commande écrit la recette du composant dans le fichier de recette JSON ou YAML que vous fournissez dans votre commande.

Linux, macOS, or Unix

```
aws greengrassv2 get-component \
  --arn
arn:aws:greengrass:region:aws:components:aws.greengrass.DLRImageClassification:versions
\
  --recipe-output-format JSON | YAML \
  --query recipe \
  --output text | base64 --decode > <recipe-file>
```

Windows Command Prompt (CMD)

```
aws greengrassv2 get-component ^
  --arn
arn:aws:greengrass:region:aws:components:aws.greengrass.DLRImageClassification:versions
^
  --recipe-output-format JSON | YAML ^
  --query recipe ^
  --output text > <recipe-file>.base64

certutil -decode <recipe-file>.base64 <recipe-file>
```

PowerShell

```
aws greengrassv2 get-component `
  --arn
arn:aws:greengrass:region:aws:components:aws.greengrass.DLRImageClassification:versions
`
  --recipe-output-format JSON | YAML `
  --query recipe `
  --output text > <recipe-file>.base64
```

```
certutil -decode <recipe-file>.base64 <recipe-file>
```

Remplacez *<recipe-file>* par le nom de la recette dans le format *<component-name>-<component-version>*.

2. Dans l'`ComponentDependencies` objet de votre recette, effectuez l'une ou plusieurs des opérations suivantes en fonction du modèle et des composants d'exécution que vous souhaitez utiliser :
 - Conservez la dépendance aux composants DLR si vous souhaitez utiliser des modèles compilés par DLR. Vous pouvez également le remplacer par une dépendance à un composant d'exécution personnalisé, comme illustré dans l'exemple suivant.

Composant d'exécution

JSON

```
{
  "<runtime-component>": {
    "VersionRequirement": "<version>",
    "DependencyType": "HARD"
  }
}
```

YAML

```
<runtime-component>:
  VersionRequirement: "<version>"
  DependencyType: HARD
```

- Conservez la dépendance du magasin de modèles de classification des images DLR pour utiliser les modèles ResNet -50 préentraînés qui le AWS fournissent, ou modifiez-la pour utiliser un composant de modèle personnalisé. Lorsque vous incluez une dépendance pour un composant de modèle public, si une version personnalisée ultérieure du composant existe dans le même Compte AWS et Région AWS, le composant d'inférence utilise ce composant personnalisé. Spécifiez la dépendance entre les composants du modèle, comme indiqué dans les exemples suivants.

Composant du modèle public

JSON

```
{
  "variant.DLR.ImageClassification.ModelStore": {
    "VersionRequirement": "<version>",
    "DependencyType": "HARD"
  }
}
```

YAML

```
variant.DLR.ImageClassification.ModelStore:
  VersionRequirement: "<version>"
  DependencyType: HARD
```

Composant de modèle personnalisé

JSON

```
{
  "<custom-model-component>": {
    "VersionRequirement": "<version>",
    "DependencyType": "HARD"
  }
}
```

YAML

```
<custom-model-component>:
  VersionRequirement: "<version>"
  DependencyType: HARD
```

3. Dans l'`ComponentConfiguration` objet, ajoutez la configuration par défaut pour ce composant. Vous pouvez modifier cette configuration ultérieurement lorsque vous déployez le composant. L'extrait suivant montre la configuration du composant de classification d'images DLR.

Par exemple, si vous utilisez un composant de modèle personnalisé comme dépendance pour votre composant d'inférence personnalisé, `ModelResourceKey` modifiez-le pour fournir les noms des modèles que vous utilisez.

JSON

```
{
  "accessControl": {
    "aws.greengrass.ipc.mqttproxy": {
      "aws.greengrass.ImageClassification:mqttproxy:1": {
        "policyDescription": "Allows access to publish via topic ml/dlr/image-
classification.",
        "operations": [
          "aws.greengrass#PublishToIoTCore"
        ],
        "resources": [
          "ml/dlr/image-classification"
        ]
      }
    }
  },
  "PublishResultsOnTopic": "ml/dlr/image-classification",
  "ImageName": "cat.jpeg",
  "InferenceInterval": 3600,
  "ModelResourceKey": {
    "armv7l": "DLR-resnet50-armv7l-cpu-ImageClassification",
    "x86_64": "DLR-resnet50-x86_64-cpu-ImageClassification",
    "aarch64": "DLR-resnet50-aarch64-cpu-ImageClassification"
  }
}
```

YAML

```
accessControl:
  aws.greengrass.ipc.mqttproxy:
    'aws.greengrass.ImageClassification:mqttproxy:1':
      policyDescription: 'Allows access to publish via topic ml/dlr/image-
classification.'
      operations:
        - 'aws.greengrass#PublishToIoTCore'
      resources:
        - ml/dlr/image-classification
PublishResultsOnTopic: ml/dlr/image-classification
ImageName: cat.jpeg
InferenceInterval: 3600
ModelResourceKey:
```

```
armv7l: "DLR-resnet50-armv7l-cpu-ImageClassification"
x86_64: "DLR-resnet50-x86_64-cpu-ImageClassification"
aarch64: "DLR-resnet50-aarch64-cpu-ImageClassification"
```

4. Dans l'`Manifests`objet, fournissez des informations sur les artefacts et la configuration de ce composant utilisés lorsque le composant est déployé sur différentes plateformes, ainsi que toute autre information requise pour exécuter correctement le composant. L'extrait suivant montre la configuration de l'`Manifests`objet pour la plate-forme Linux dans le composant de classification d'images DLR.

JSON

```
{
  "Manifests": [
    {
      "Platform": {
        "os": "linux",
        "architecture": "arm"
      },
      "Name": "32-bit armv7l - Linux (raspberry pi)",
      "Artifacts": [
        {
          "URI": "s3://SAMPLE-BUCKET/sample-artifacts-directory/
image_classification.zip",
          "Unarchive": "ZIP"
        }
      ],
      "Lifecycle": {
        "Setenv": {
          "DLR_IC_MODEL_DIR":
            "{variant.DLR.ImageClassification.ModelStore:artifacts:decompressedPath}/
{configuration:/ModelResourceKey/armv7l}",
          "DEFAULT_DLR_IC_IMAGE_DIR": "{artifacts:decompressedPath}/
image_classification/sample_images/"
        },
        "Run": {
          "RequiresPrivilege": true,
          "script": ". {variant.DLR:configuration:/MLRootPath}/
greengrass_ml_dlr_venv/bin/activate\npython3 {artifacts:decompressedPath}/
image_classification/inference.py"
        }
      }
    }
  ]
}
```

```
]
}
```

YAML

```
Manifests:
- Platform:
  os: linux
  architecture: arm
  Name: 32-bit armv7l - Linux (raspberry pi)
  Artifacts:
    - URI: s3://SAMPLE-BUCKET/sample-artifacts-directory/
      image_classification.zip
      Unarchive: ZIP
  Lifecycle:
    SetEnv:
      DLR_IC_MODEL_DIR:
        "{variant.DLR.ImageClassification.ModelStore:artifacts:decompressedPath}/{configuration:/ModelResourceKey/armv7l}"
      DEFAULT_DLR_IC_IMAGE_DIR: "{artifacts:decompressedPath}/image_classification/sample_images/"
  Run:
    RequiresPrivilege: true
    script: |-
      . {variant.DLR:configuration:/MLRootPath}/greengrass_ml_dlr_venv/bin/activate
      python3 {artifacts:decompressedPath}/image_classification/inference.py
```

Pour des informations détaillées sur la création de recettes de composants, consultez [AWS IoT Greengrass référence de recette de composant](#).

Création du composant d'inférence

Utilisez la AWS IoT Greengrass console ou le AWS CLI pour créer un composant à l'aide de la recette que vous venez de définir. Après avoir créé le composant, vous pouvez le déployer pour effectuer des inférences sur votre appareil. Pour un exemple de déploiement d'un composant d'inférence, consultez [Tutoriel : Effectuer une inférence de classification d'images d'échantillons à l'aide TensorFlow de Lite](#).

Création d'un composant d'inférence personnalisé (console)

1. Connectez-vous à la [console AWS IoT Greengrass](#).
2. Dans le menu de navigation, sélectionnez Composants.
3. Sur la page Composants, dans l'onglet Mes composants, choisissez Créer un composant.
4. Sur la page Créer un composant, sous Informations sur le composant, sélectionnez Entrer la recette en tant que JSON ou Entrer la recette en tant que YAML comme source de composant.
5. Dans le champ Recette, entrez la recette personnalisée que vous avez créée.
6. Cliquez sur Créer un composant.

Créer un composant d'inférence personnalisé (AWS CLI)

Exécutez la commande suivante pour créer un nouveau composant personnalisé à l'aide de la recette que vous avez créée.

```
aws greengrassv2 create-component-version \  
  --inline-recipe fileb://path/to/recipe/file
```

Note

Cette étape crée le composant dans le AWS IoT Greengrass service dans le AWS Cloud. Vous pouvez utiliser la CLI Greengrass pour développer, tester et déployer votre composant localement avant de le télécharger dans le cloud. Pour de plus amples informations, veuillez consulter [Développer des AWS IoT Greengrass composants](#).

Résolution des problèmes liés à l'inférence par apprentissage automatique

Utilisez les informations de dépannage et les solutions de cette section pour résoudre les problèmes liés à vos composants d'apprentissage automatique. Pour les composants publics d'inférence d'apprentissage automatique, consultez les messages d'erreur dans les journaux des composants suivants :

Linux or Unix

- */greengrass/v2/logs/aws.greengrass.DLRImageClassification.log*

- `/greengrass/v2/logs/aws.greengrass.DLRObjectDetection.log`
- `/greengrass/v2/logs/
aws.greengrass.TensorFlowLiteImageClassification.log`
- `/greengrass/v2/logs/aws.greengrass.TensorFlowLiteObjectDetection.log`

Windows

- `C:\greengrass\v2\logs\aws.greengrass.DLRImageClassification.log`
- `C:\greengrass\v2\logs\aws.greengrass.DLRObjectDetection.log`
- `C:\greengrass\v2\logs
\aws.greengrass.TensorFlowLiteImageClassification.log`
- `C:\greengrass\v2\logs\aws.greengrass.TensorFlowLiteObjectDetection.log`

Si un composant est correctement installé, le journal du composant contient l'emplacement de la bibliothèque qu'il utilise pour l'inférence.

Problèmes

- [Impossible de récupérer la bibliothèque](#)
- [Cannot open shared object file](#)
- [Error: ModuleNotFoundError: No module named '<library>'](#)
- [Aucun appareil compatible CUDA n'est détecté](#)
- [Aucun fichier ou répertoire de ce type](#)
- [RuntimeError: module compiled against API version 0xf but this version of NumPy is <version>](#)
- [picamera.exc.PiCameraError: Camera is not enabled](#)
- [Erreurs mémoires](#)
- [Erreurs liées à l'espace disque](#)
- [Erreurs de délai d'attente](#)

Impossible de récupérer la bibliothèque

L'erreur suivante se produit lorsque le script d'installation ne parvient pas à télécharger une bibliothèque requise lors du déploiement sur un périphérique Raspberry Pi.

```
Err:2 http://raspbian.raspberrypi.org/raspbian buster/main armhf python3.7-dev armhf
3.7.3-2+deb10u1
404 Not Found [IP: 93.93.128.193 80]
E: Failed to fetch http://raspbian.raspberrypi.org/raspbian/pool/main/p/python3.7/
libpython3.7-dev_3.7.3-2+deb10u1_armhf.deb 404 Not Found [IP: 93.93.128.193 80]
```

Exécutez `sudo apt-get update` et déployez à nouveau votre composant.

Cannot open shared object file

Des erreurs similaires aux suivantes peuvent s'afficher lorsque le script d'installation ne parvient pas à télécharger une dépendance requise `opencv-python` lors du déploiement sur un périphérique Raspberry Pi.

```
ImportError: libopenjp2.so.7: cannot open shared object file: No such file or directory
```

Exécutez la commande suivante pour installer manuellement les dépendances pour `opencv-python` :

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

Error: ModuleNotFoundError: No module named '<library>'

Cette erreur peut s'afficher dans les journaux des composants d'exécution ML (`variant.DLR.log` ou `variant.TensorFlowLite.log`) lorsque la bibliothèque d'exécution ML ou ses dépendances ne sont pas installées correctement. Cette erreur peut se produire dans les cas suivants :

- Si vous utilisez l'`UseInstallerOption`, qui est activée par défaut, cette erreur indique que le composant d'exécution ML n'a pas réussi à installer le moteur d'exécution ou ses dépendances. Procédez comme suit :
 1. Configurez le composant d'exécution ML pour désactiver l'`UseInstallerOption`.
 2. Installez le moteur d'exécution ML et ses dépendances, puis mettez-les à la disposition de l'utilisateur du système qui exécute les composants du ML. Pour plus d'informations, consultez les ressources suivantes :
 - [Option d'exécution UseInstaller du DLR](#)

- [TensorFlow UseInstaller Option d'exécution allégée](#)
- Si vous n'utilisez pas cette UseInstaller option, cette erreur indique que le moteur d'exécution ML ou ses dépendances ne sont pas installés pour l'utilisateur du système qui exécute les composants ML. Procédez comme suit :
1. Vérifiez que la bibliothèque est installée pour l'utilisateur du système qui exécute les composants ML. Remplacez *ggc_user* par le nom de l'utilisateur du système et remplacez *tflite_runtime* par le nom de la bibliothèque à vérifier.

Linux or Unix

```
sudo -H -u ggc_user bash -c "python3 -c 'import tflite_runtime'"
```

Windows

```
runas /user:ggc_user "py -3 -c \"import tflite_runtime\""
```

2. Si la bibliothèque n'est pas installée, installez-la pour cet utilisateur. Remplacez *ggc_user* par le nom de l'utilisateur du système et remplacez *tflite_runtime* par le nom de la bibliothèque.

Linux or Unix

```
sudo -H -u ggc_user bash -c "python3 -m pip install --user tflite_runtime"
```

Windows

```
runas /user:ggc_user "py -3 -m pip install --user tflite_runtime"
```

Pour plus d'informations sur les dépendances de chaque environnement d'exécution ML, consultez les rubriques suivantes :

- [Option d'exécution UseInstaller du DLR](#)
 - [TensorFlow UseInstaller Option d'exécution allégée](#)
3. Si le problème persiste, installez la bibliothèque pour un autre utilisateur afin de vérifier si ce périphérique peut installer la bibliothèque. L'utilisateur peut être, par exemple, votre utilisateur, l'utilisateur root ou un utilisateur administrateur. Si vous ne parvenez pas à installer correctement la bibliothèque pour un utilisateur, il est possible que votre appareil ne la prenne pas en charge. Consultez la documentation de la bibliothèque pour connaître les exigences et résoudre les problèmes d'installation.

Aucun appareil compatible CUDA n'est détecté

L'erreur suivante peut s'afficher lorsque vous utilisez l'accélération GPU. Exécutez la commande suivante pour permettre à l'utilisateur de Greengrass d'accéder au GPU.

```
sudo usermod -a -G video ggc_user
```

Aucun fichier ou répertoire de ce type

Les erreurs suivantes indiquent que le composant d'exécution n'a pas pu configurer correctement l'environnement virtuel :

- *MLRootPath*/greengrass_ml_dlr_conda/bin/conda: No such file or directory
- *MLRootPath*/greengrass_ml_dlr_venv/bin/activate: No such file or directory
- *MLRootPath*/greengrass_ml_tflite_conda/bin/conda: No such file or directory
- *MLRootPath*/greengrass_ml_tflite_venv/bin/activate: No such file or directory

Vérifiez les journaux pour vous assurer que toutes les dépendances d'exécution ont été correctement installées. Pour plus d'informations sur les bibliothèques installées par le script d'installation, consultez les rubriques suivantes :

- [Temps d'exécution du DLR](#)
- [TensorFlow Temps d'exécution allégé](#)

Par défaut, *MLRootPath* il est défini sur */greengrass/v2/work/component-name/greengrass_ml*. Pour modifier cet emplacement, incluez le composant [TensorFlow Temps d'exécution allégé](#) d'exécution [Temps d'exécution du DLR](#) ou directement dans votre déploiement et spécifiez une valeur modifiée pour le *MLRootPath* paramètre dans une mise à jour de fusion de configuration. Pour plus d'informations sur la configuration du composant, consultez [Mettre à jour les configurations des composants](#).

 Note

Pour le composant DLR v1.3.x, vous définissez le MLRootPath paramètre dans la configuration du composant d'inférence, et la valeur par défaut est. \$HOME/greengrass_ml

RuntimeError: module compiled against API version 0xf but this version of NumPy is <version>

Les erreurs suivantes peuvent s'afficher lorsque vous exécutez l'inférence d'apprentissage automatique sur un Raspberry Pi exécutant Raspberry Pi OS Bullseye.

```
RuntimeError: module compiled against API version 0xf but this version of numpy is 0xd
ImportError: numpy.core.multiarray failed to import
```

Cette erreur se produit parce que Raspberry Pi OS Bullseye inclut une version antérieure NumPy à celle requise par OpenCV. Pour résoudre ce problème, exécutez la commande suivante pour effectuer la mise NumPy à niveau vers la dernière version.

```
pip3 install --upgrade numpy
```

picamera.exc.PiCameraError: Camera is not enabled

L'erreur suivante peut s'afficher lorsque vous exécutez l'inférence d'apprentissage automatique sur un Raspberry Pi exécutant Raspberry Pi OS Bullseye.

```
picamera.exc.PiCameraError: Camera is not enabled. Try running 'sudo raspi-config' and
ensure that the camera has been enabled.
```

Cette erreur se produit car Raspberry Pi OS Bullseye inclut une nouvelle pile de caméras qui n'est pas compatible avec les composants ML. Pour résoudre ce problème, activez l'ancienne pile de caméras.

Pour activer l'ancienne pile de caméras

1. Exécutez la commande suivante pour ouvrir l'outil de configuration du Raspberry Pi.

```
sudo raspi-config
```

2. Sélectionnez Options d'interface.
3. Sélectionnez Legacy camera pour activer l'ancienne pile de caméras.
4. Redémarrez l'appareil Raspberry Pi.

Erreurs mémoires

Les erreurs suivantes se produisent généralement lorsque le périphérique ne dispose pas de suffisamment de mémoire et que le processus du composant est interrompu.

- `stderr. Killed.`
- `exitCode=137`

Nous recommandons un minimum de 500 Mo de mémoire pour déployer un composant public d'inférence d'apprentissage automatique.

Erreurs liées à l'espace disque

L'erreur `no space left on device` se produit généralement lorsqu'un appareil ne dispose pas de suffisamment d'espace de stockage. Assurez-vous qu'il y a suffisamment d'espace disque disponible sur votre appareil avant de déployer à nouveau le composant. Nous recommandons un minimum de 500 Mo d'espace disque libre pour déployer un composant public d'inférence d'apprentissage automatique.

Erreurs de délai d'attente

Les composants publics d'apprentissage automatique téléchargent de gros fichiers de modèles d'apprentissage automatique d'une taille supérieure à 200 Mo. Si le délai de téléchargement expire pendant le déploiement, vérifiez la vitesse de votre connexion Internet et réessayez le déploiement.

Gérez les appareils principaux de Greengrass avec AWS Systems Manager

Note

AWS IoT Greengrass ne prend actuellement pas en charge cette fonctionnalité sur les appareils Windows principaux.

Systems Manager est un AWS service que vous pouvez utiliser pour visualiser et contrôler votre infrastructure AWS, y compris les EC2 instances Amazon, les serveurs sur site et les machines virtuelles (VMs), ainsi que les appareils périphériques. Systems Manager vous permet de visualiser les données opérationnelles, d'automatiser les tâches opérationnelles et de garantir la sécurité et la conformité. Lorsque vous enregistrez une machine auprès de Systems Manager, elle est appelée nœud géré. Pour plus d'informations, voir [Qu'est-ce que c'est AWS Systems Manager ?](#) dans le guide de AWS Systems Manager l'utilisateur.

L' AWS Systems Manager Agent (Systems Manager Agent) est un logiciel que vous pouvez installer sur les appareils pour permettre à Systems Manager de les mettre à jour, de les gérer et de les configurer. Pour installer l'agent Systems Manager sur les appareils principaux de Greengrass, déployez le composant [Systems Manager Agent](#). Lorsque vous déployez l'agent Systems Manager pour la première fois, celui-ci enregistre le périphérique principal en tant que nœud géré par Systems Manager. L'agent Systems Manager s'exécute sur l'appareil pour permettre la communication avec le service Systems Manager dans le AWS Cloud. Pour plus d'informations sur l'installation et la configuration du composant Agent Systems Manager, reportez-vous à la section [Installation de l' AWS Systems Manager agent](#).

Les outils et fonctionnalités de Systems Manager sont appelés fonctionnalités. Les appareils principaux de Greengrass prennent en charge toutes les fonctionnalités de Systems Manager. Pour plus d'informations sur ces fonctionnalités et sur la façon d'utiliser Systems Manager pour gérer les appareils principaux, consultez la section [Fonctionnalités de Systems Manager](#) dans le Guide de AWS Systems Manager l'utilisateur.

AWS Systems Manager propose un niveau d'instances standard et un niveau d'instances avancées pour les nœuds gérés par Systems Manager. Si vous utilisez Systems Manager pour la première fois, vous devez commencer par le niveau des instances standard. Région AWS Au niveau des

instances standard, vous pouvez enregistrer jusqu'à 1 000 nœuds gérés par utilisateur. Compte AWS Si vous devez enregistrer plus de 1 000 nœuds gérés dans un seul compte et une seule région, ou si vous devez utiliser la [fonctionnalité Gestionnaire de session](#), utilisez le niveau d'instances avancées. Pour plus d'informations, consultez [la section Configuration des niveaux d'instance](#) dans le Guide de AWS Systems Manager l'utilisateur.

Rubriques

- [Installation de l' AWS Systems Manager agent](#)
- [Désinstallez l' AWS Systems Manager agent](#)

Installation de l' AWS Systems Manager agent

L' AWS Systems Manager agent (Systems Manager Agent) est un logiciel Amazon que vous installez pour permettre à Systems Manager de mettre à jour, de gérer et de configurer les appareils principaux de Greengrass, les EC2 instances Amazon et d'autres ressources. L'agent traite et exécute les demandes du service Systems Manager dans le AWS Cloud. L'agent renvoie ensuite les informations d'état et d'exécution au service Systems Manager. Pour plus d'informations, reportez-vous [à la section À propos de Systems Manager Agent](#) dans le guide de AWS Systems Manager l'utilisateur.

AWS fournit l'agent Systems Manager sous la forme d'un composant Greengrass que vous pouvez déployer sur vos appareils principaux Greengrass afin de les gérer avec Systems Manager. Le [composant Systems Manager Agent](#) installe le logiciel Systems Manager Agent et enregistre le périphérique principal en tant que nœud géré dans Systems Manager. Suivez les étapes indiquées sur cette page pour effectuer les prérequis et déployer le composant Systems Manager Agent sur un périphérique principal ou un groupe de périphériques principaux.

Rubriques

- [Étape 1 : Exécution des étapes générales de configuration de Systems Manager](#)
- [Étape 2 : Création d'un rôle de service IAM pour Systems Manager](#)
- [Étape 3 : ajouter des autorisations au rôle d'échange de jetons](#)
- [Étape 4 : Déploiement du composant Systems Manager Agent](#)
- [Étape 5 : vérifier l'enregistrement de l'appareil principal avec Systems Manager](#)

Étape 1 : Exécution des étapes générales de configuration de Systems Manager

Si ce n'est pas déjà fait, suivez les étapes de configuration générales pour AWS Systems Manager. Pour plus d'informations, reportez-vous à la section [Étapes générales complètes de configuration de Systems Manager](#) dans le Guide de AWS Systems Manager l'utilisateur.

Étape 2 : Création d'un rôle de service IAM pour Systems Manager

L'agent Systems Manager utilise un rôle de service Gestion des identités et des accès AWS (IAM) pour communiquer avec AWS Systems Manager. Systems Manager assume ce rôle pour activer les fonctionnalités de Systems Manager sur chaque périphérique principal. Le composant Systems Manager Agent utilise également ce rôle pour enregistrer le périphérique principal en tant que nœud géré par Systems Manager lorsque vous déployez le composant. Si ce n'est pas déjà fait, créez un rôle de service Systems Manager à utiliser par le composant Systems Manager Agent. Pour plus d'informations, consultez la section [Créer un rôle de service IAM pour les appareils Edge](#) dans le guide de l'AWS Systems Manager utilisateur.

Étape 3 : ajouter des autorisations au rôle d'échange de jetons

Les appareils principaux de Greengrass utilisent un rôle de service IAM, appelé rôle d'échange de jetons, pour interagir avec les services. AWS Chaque périphérique principal possède un rôle d'échange de jetons que vous créez lorsque vous [installez le logiciel AWS IoT Greengrass Core](#). De nombreux composants de Greengrass, tels que l'agent Systems Manager, nécessitent des autorisations supplémentaires pour ce rôle. Le composant de l'agent Systems Manager nécessite les autorisations suivantes, notamment l'autorisation d'utiliser le rôle que vous avez créé dans [Étape 2 : Création d'un rôle de service IAM pour Systems Manager](#).

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "iam:PassRole"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:iam::account-id:role/SSMServiceRole"
      ]
    }
  ]
}
```

```
    },
    {
      "Action": [
        "ssm:AddTagsToResource",
        "ssm:RegisterManagedInstance"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

Si ce n'est pas déjà fait, ajoutez ces autorisations au rôle d'échange de jetons du périphérique principal pour permettre à l'agent Systems Manager de fonctionner. Vous pouvez ajouter une nouvelle politique au rôle d'échange de jetons pour accorder cette autorisation.

Pour ajouter des autorisations au rôle d'échange de jetons (console)

1. Dans le menu de navigation de [la console IAM](#), sélectionnez Rôles.
2. Choisissez le rôle IAM que vous avez configuré comme rôle d'échange de jetons lors de l'installation du logiciel AWS IoT Greengrass Core. Si vous n'avez pas spécifié de nom pour le rôle d'échange de jetons lorsque vous avez installé le logiciel AWS IoT Greengrass Core, celui-ci a créé un rôle nommé `GreengrassV2TokenExchangeRole`.
3. Sous Autorisations, choisissez Ajouter des autorisations, puis choisissez Joindre des politiques.
4. Choisissez Create Policy (Créer une politique). La page Créer une politique s'ouvre dans un nouvel onglet du navigateur.
5. Sur la page Create policy (Créer une stratégie), procédez comme suit :
 - a. Choisissez JSON pour ouvrir l'éditeur JSON.
 - b. Collez le politique suivante dans l'éditeur JSON. *SSMServiceRole* Remplacez-le par le nom du rôle de service que vous avez créé dans [Étape 2 : Création d'un rôle de service IAM pour Systems Manager](#).

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "iam:PassRole"
      ],
```

```

    "Effect": "Allow",
    "Resource": [
      "arn:aws:iam::account-id:role/SSMSERVICERole"
    ],
  },
  {
    "Action": [
      "ssm:AddTagsToResource",
      "ssm:RegisterManagedInstance"
    ],
    "Effect": "Allow",
    "Resource": "*"
  }
]
}

```

- c. Choisissez Suivant : Balises.
 - d. Choisissez Suivant : Vérification.
 - e. Dans le champ Nom, entrez le nom de la stratégie, par exemple **GreengrassSSMAgentComponentPolicy**.
 - f. Choisissez Create Policy (Créer une politique).
 - g. Passez à l'onglet de navigateur précédent dans lequel le rôle d'échange de jetons est ouvert.
6. Sur la page Ajouter des autorisations, cliquez sur le bouton d'actualisation, puis sélectionnez la politique d'agent Greengrass Systems Manager que vous avez créée à l'étape précédente.
 7. Choisissez Attach Policies (Attacher des politiques).

Les principaux appareils qui utilisent ce rôle d'échange de jetons sont désormais autorisés à interagir avec le service Systems Manager.

Pour ajouter des autorisations au rôle d'échange de jetons (AWS CLI)

Pour ajouter une politique autorisant l'utilisation de Systems Manager

1. Créez un fichier appelé `ssm-agent-component-policy.json` et copiez-y le code JSON suivant. *SSMSERVICERole* Remplacez-le par le nom du rôle de service que vous avez créé dans [Étape 2 : Création d'un rôle de service IAM pour Systems Manager](#).

```
{
```

```
"Version": "2012-10-17",
"Statement": [
  {
    "Action": [
      "iam:PassRole"
    ],
    "Effect": "Allow",
    "Resource": [
      "arn:aws:iam::account-id:role/SSMServiceRole"
    ]
  },
  {
    "Action": [
      "ssm:AddTagsToResource",
      "ssm:RegisterManagedInstance"
    ],
    "Effect": "Allow",
    "Resource": "*"
  }
]
```

2. Exécutez la commande suivante pour créer la politique à partir du document de stratégie dans `ssm-agent-component-policy.json`.

Linux or Unix

```
aws iam create-policy \
  --policy-name GreengrassSSMAgentComponentPolicy \
  --policy-document file://ssm-agent-component-policy.json
```

Windows Command Prompt (CMD)

```
aws iam create-policy ^
  --policy-name GreengrassSSMAgentComponentPolicy ^
  --policy-document file://ssm-agent-component-policy.json
```

PowerShell

```
aws iam create-policy `
  --policy-name GreengrassSSMAgentComponentPolicy `
  --policy-document file://ssm-agent-component-policy.json
```


Copiez le nom Amazon Resource Name (ARN) de la politique à partir des métadonnées de la politique dans la sortie. Vous utilisez cet ARN pour associer cette politique au rôle principal de l'appareil à l'étape suivante.

3. Exécutez la commande suivante pour associer la politique au rôle d'échange de jetons.

- *GreengrassV2TokenExchangeRole* Remplacez-le par le nom du rôle d'échange de jetons que vous avez spécifié lors de l'installation du logiciel AWS IoT Greengrass Core. Si vous n'avez pas spécifié de nom pour le rôle d'échange de jetons lorsque vous avez installé le logiciel AWS IoT Greengrass Core, celui-ci a créé un rôle nommé *GreengrassV2TokenExchangeRole*.
- Remplacez l'ARN de la politique par l'ARN de l'étape précédente.

Linux or Unix

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

Windows Command Prompt (CMD)

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

PowerShell

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

Si la commande n'a aucune sortie, elle a réussi. Les principaux appareils qui utilisent ce rôle d'échange de jetons sont désormais autorisés à interagir avec le service Systems Manager.

Étape 4 : Déploiement du composant Systems Manager Agent

Procédez comme suit pour déployer et configurer le composant Systems Manager Agent. Vous pouvez déployer le composant sur un seul périphérique principal ou sur un groupe de périphériques principaux.

Pour déployer le composant Systems Manager Agent (console)

1. Dans le menu de navigation de la [AWS IoT Greengrass console](#), sélectionnez Composants.
2. Sur la page Composants, choisissez l'onglet Composants publics, puis sélectionnez `aws.greengrass.SystemsManagerAgent`.
3. Sur la page `aws.greengrass.SystemsManagerAgent`, choisissez Deploy (Déployer).
4. Dans Ajouter au déploiement, choisissez un déploiement existant à réviser ou choisissez de créer un nouveau déploiement, puis choisissez Suivant.
5. Si vous avez choisi de créer un nouveau déploiement, choisissez le périphérique principal ou le groupe d'objets cible pour le déploiement. Sur la page Spécifier la cible, sous Cible de déploiement, choisissez un périphérique principal ou un groupe d'objets, puis cliquez sur Suivant.
6. Sur la page Sélectionner les composants, vérifiez que le `aws.greengrass.SystemsManagerAgent` composant est sélectionné, puis choisissez Next.
7. Sur la page Configurer les composants, sélectionnez `aws.greengrass.SystemsManagerAgent`, puis effectuez les opérations suivantes :
 - a. Choisissez Configure component (Configurer un composant).
 - b. Dans le `aws.greengrass.SystemsManagerAgent` mode Configurer, sous Mise à jour de la configuration, dans Configuration à fusionner, entrez la mise à jour de configuration suivante. `SSMServiceRole` Remplacez-le par le nom du rôle de service que vous avez créé dans [Étape 2 : Création d'un rôle de service IAM pour Systems Manager](#).

```
{
  "SSMRegistrationRole": "SSMServiceRole",
  "SSMOverrideExistingRegistration": false
}
```

 Note

Si le périphérique principal exécute déjà l'agent Systems Manager enregistré avec une activation hybride, passez `SSMOverrideExistingRegistration` à `true`. Ce paramètre indique si le composant Systems Manager Agent enregistre le périphérique principal alors que l'agent Systems Manager est déjà en cours d'exécution sur le périphérique avec une activation hybride.

Vous pouvez également spécifier des balises (`SSMResourceTags`) à ajouter au nœud géré par Systems Manager créé par le composant Systems Manager Agent pour le périphérique principal. Pour plus d'informations, consultez la section [Configuration des composants de l'agent Systems Manager](#).

- c. Choisissez Confirmer pour fermer le modal, puis cliquez sur Suivant.
8. Sur la page Configure advanced settings (Configurer les paramètres avancés), conservez les paramètres de configuration par défaut et choisissez Next (Suivant).
9. Sur la page Review (Révision), choisissez Deploy (Déployer).

Le déploiement peut prendre jusqu'à une minute.

Pour déployer le composant Systems Manager Agent (AWS CLI)

Pour déployer le composant Systems Manager Agent, créez un document de déploiement qui inclut `aws.greengrass.SystemsManagerAgent` l'componentsobjet et spécifiez la mise à jour de configuration pour le composant. Suivez les instructions [Créer des déploiements](#) pour créer un nouveau déploiement ou modifier un déploiement existant.

L'exemple de document de déploiement partiel suivant indique d'utiliser un rôle de service nommé `SSMServiceRole`. *SSMServiceRole* Remplacez-le par le nom du rôle de service que vous avez créé dans [Étape 2 : Création d'un rôle de service IAM pour Systems Manager](#).

```
{
  ...,
  "components": {
    ...,
    "aws.greengrass.SystemsManagerAgent": {
      "componentVersion": "1.0.0",
      "configurationUpdate": {
```

```
    "merge": "{\\"SSMRegistrationRole\\":\\"SSMSERVICERole\\",
\\"SSMOverrideExistingRegistration\\":false}"
  }
}
```

Note

Si le périphérique principal exécute déjà l'agent Systems Manager enregistré avec une activation hybride, passez `SSMOverrideExistingRegistration` à `true`. Ce paramètre indique si le composant Systems Manager Agent enregistre le périphérique principal alors que l'agent Systems Manager est déjà en cours d'exécution sur le périphérique avec une activation hybride.

Vous pouvez également spécifier des balises (`SSMResourceTags`) à ajouter au nœud géré par Systems Manager créé par le composant Systems Manager Agent pour le périphérique principal. Pour plus d'informations, consultez la section [Configuration des composants de l'agent Systems Manager](#).

L'exécution du déploiement peut prendre plusieurs minutes. Vous pouvez utiliser le AWS IoT Greengrass service pour vérifier l'état du déploiement, et vous pouvez consulter les journaux du logiciel AWS IoT Greengrass principal et les journaux des composants de l'agent Systems Manager pour vérifier que l'agent Systems Manager s'exécute correctement. Pour plus d'informations, consultez les ressources suivantes :

- [Vérifier l'état du déploiement](#)
- [AWS IoT Greengrass Journaux de surveillance](#)
- [Affichage des journaux de l'agent Systems Manager](#) dans le guide de AWS Systems Manager l'utilisateur

Si le déploiement échoue ou si l'agent Systems Manager ne s'exécute pas, vous pouvez résoudre les problèmes liés au déploiement sur chaque appareil principal. Pour plus d'informations, consultez les ressources suivantes :

- [Résolution des problèmes AWS IoT Greengrass V2](#)

- [Résolution des problèmes liés à l'agent Systems Manager](#) dans le guide de AWS Systems Manager l'utilisateur

Étape 5 : vérifier l'enregistrement de l'appareil principal avec Systems Manager

Lorsque le composant Systems Manager Agent s'exécute, il enregistre le périphérique principal en tant que nœud géré dans Systems Manager. Vous pouvez utiliser la AWS IoT Greengrass console, la console Systems Manager et l'API Systems Manager pour vérifier qu'un périphérique principal est enregistré en tant que nœud géré. Les nœuds gérés sont également appelés instances dans certaines parties de la AWS console et de l'API.

Pour vérifier l'enregistrement de l'appareil principal (AWS IoT Greengrass console)

1. Dans le menu de navigation de la [AWS IoT Greengrass console](#), choisissez Core devices.
2. Choisissez le périphérique principal à vérifier.
3. Sur la page de détails du périphérique principal, recherchez la propriété de l'AWS Systems Manager instance. Si cette propriété est présente et affiche un lien vers la console Systems Manager, le périphérique principal est enregistré en tant que nœud géré.

Vous pouvez également trouver la propriété AWS Systems Manager ping status pour vérifier l'état de l'agent Systems Manager sur le périphérique principal. Lorsque le statut est En ligne, vous pouvez gérer le périphérique principal avec Systems Manager.

Pour vérifier l'enregistrement du périphérique principal (console Systems Manager)

1. Dans le menu de navigation [de la console Systems Manager](#), choisissez Fleet Manager.
2. Sous Nœuds gérés, procédez comme suit :
 - a. Ajoutez un filtre où se trouve le type de source `AWS::IoT::Thing`.
 - b. Ajoutez un filtre dans lequel l'ID source est le nom du périphérique principal à vérifier.
3. Trouvez le périphérique principal dans le tableau des nœuds gérés. Si le périphérique principal figure dans le tableau, il est enregistré en tant que nœud géré.

Vous pouvez également trouver la propriété ping status de l'agent Systems Manager pour vérifier l'état de l'agent Systems Manager sur le périphérique principal. Lorsque le statut est En ligne, vous pouvez gérer le périphérique principal avec Systems Manager.

Pour vérifier l'enregistrement de l'appareil principal (AWS CLI)

- Utilisez cette [DescribeInstanceInformation](#) opération pour obtenir la liste des nœuds gérés correspondant à un filtre que vous spécifiez. Exécutez la commande suivante pour vérifier si un périphérique principal est enregistré en tant que nœud géré. Remplacez *MyGreengrassCore* par le nom du périphérique principal à vérifier.

```
aws ssm describe-instance-information --filter  
Key=SourceIds,Values=MyGreengrassCore Key=SourceTypes,Values=AWS::IoT::Thing
```

La réponse contient la liste des nœuds gérés qui correspondent au filtre. Si la liste contient un nœud géré, le périphérique principal est enregistré en tant que nœud géré. Vous trouverez également d'autres informations sur le nœud géré du périphérique principal dans la réponse. Si c'est PingStatus le casOnline, vous pouvez gérer le périphérique principal à l'aide de Systems Manager.

Après avoir vérifié qu'un appareil principal est enregistré en tant que nœud géré dans Systems Manager, vous pouvez utiliser la console et l'API Systems Manager pour gérer cet appareil principal. Pour plus d'informations sur les fonctionnalités de Systems Manager que vous pouvez utiliser pour gérer les appareils principaux de Greengrass, consultez la section [Fonctionnalités de Systems Manager](#) dans le guide de l'AWS Systems Manager utilisateur.

Désinstallez l' AWS Systems Manager agent

Si vous ne souhaitez plus gérer un appareil Greengrass Core avec AWS Systems Manager, vous pouvez désenregistrer le périphérique principal de Systems Manager et désinstaller l'agent (AWS Systems Manager Systems Manager Agent) de l'appareil.

Vous pouvez réenregistrer un appareil principal à tout moment. Pour ce faire, déployez à nouveau le composant Systems Manager Agent, qui enregistre le périphérique principal auprès de Systems Manager lors de son installation. Systems Manager conserve l'historique des commandes d'un périphérique principal désenregistré pendant 30 jours.

Rubriques

- [Étape 1 : Désenregistrer le périphérique principal de Systems Manager](#)
- [Étape 2 : désinstallation du composant de l'agent Systems Manager](#)
- [Étape 3 : désinstallez le logiciel Systems Manager Agent](#)

Étape 1 : Désenregistrer le périphérique principal de Systems Manager

Vous pouvez utiliser la console ou l'API Systems Manager pour désenregistrer le périphérique principal. Pour plus d'informations, consultez la section [Désenregistrement des nœuds gérés](#) dans le guide de l'AWS Systems Manager utilisateur.

Étape 2 : désinstallation du composant de l'agent Systems Manager

Après avoir désenregistré le périphérique principal, désinstallez le [composant Systems Manager Agent](#) de l'appareil. Pour supprimer un composant d'un appareil principal Greengrass, revoyez le déploiement qui a installé le composant et supprimez-le du déploiement. Le logiciel AWS IoT Greengrass Core désinstalle un composant lorsqu'aucun des déploiements d'un périphérique principal ne spécifie ce composant. Pour de plus amples informations, veuillez consulter [Déployer AWS IoT Greengrass des composants sur des appareils](#).

Pour désinstaller le composant Systems Manager Agent (console)

1. Dans le menu de navigation de la [AWS IoT Greengrass console](#), choisissez Core devices.
2. Choisissez le périphérique principal sur lequel vous souhaitez désinstaller le composant Systems Manager Agent.
3. Sur la page des détails de l'appareil principal, choisissez l'onglet Déploiements.
4. Choisissez le déploiement qui déploie le composant Systems Manager Agent sur le périphérique principal.
5. Sur la page des détails du déploiement, choisissez Revise.
6. Dans le mode de déploiement Revise, choisissez Revise le déploiement.
7. À l'étape 1 : Spécifier la cible, choisissez Next.
8. À l'étape 2 : Sélection des composants, effacez la sélection pour `aws.greengrass.SystemsManagerAgentcomposant`, puis choisissez Next.
9. À l'étape 3 : Configuration des composants, choisissez Next.
10. À l'étape 4 : Configuration des paramètres avancés, choisissez Next.
11. À l'étape 5 : Révision, choisissez Déployer.

Pour désinstaller le composant Systems Manager Agent (CLI)

Pour désinstaller le composant Systems Manager Agent, revoyez le déploiement qui le déploie et supprimez-le du déploiement. Pour de plus amples informations, veuillez consulter [Réviser les déploiements](#).

L'exécution du déploiement peut prendre plusieurs minutes. Vous pouvez utiliser le AWS IoT Greengrass service pour vérifier l'état du déploiement. Pour de plus amples informations, veuillez consulter [Vérifier l'état du déploiement](#).

Étape 3 : désinstallez le logiciel Systems Manager Agent

Le logiciel Systems Manager Agent continue de s'exécuter sur le périphérique principal une fois que vous avez supprimé le composant Systems Manager Agent. Pour supprimer le logiciel Systems Manager Agent, vous pouvez exécuter des commandes sur le périphérique principal. Pour plus d'informations, consultez la section [Désinstaller l'agent Systems Manager des instances Linux](#) dans le Guide de AWS Systems Manager l'utilisateur.

Sécurité dans AWS IoT Greengrass

La sécurité du cloud AWS est la priorité absolue. En tant que AWS client, vous bénéficiez d'un centre de données et d'une architecture réseau conçus pour répondre aux exigences des entreprises les plus sensibles en matière de sécurité.

La sécurité est une responsabilité partagée entre vous 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 chargée de protéger l'infrastructure qui exécute les AWS services dans le AWS Cloud. AWS vous fournit également des services que vous pouvez utiliser en toute sécurité. Des auditeurs tiers testent et vérifient régulièrement l'efficacité de notre sécurité dans le cadre des programmes de [AWS conformité Programmes](#) de de conformité. Pour en savoir plus sur les programmes de conformité qui s'appliquent à AWS IoT Greengrass, voir [AWS Services concernés par programme de conformitéAWS](#) .
- **Sécurité dans le cloud** : votre responsabilité est déterminée par le service AWS 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 les utilisez AWS IoT Greengrass, vous êtes également responsable de la sécurisation de vos appareils, de votre connexion au réseau local et de vos clés privées.

Cette documentation vous aide à comprendre comment appliquer le modèle de responsabilité partagée lors de son utilisation AWS IoT Greengrass. Les rubriques suivantes expliquent comment procéder à la configuration AWS IoT Greengrass pour atteindre vos objectifs de sécurité et de conformité. Vous apprendrez également à utiliser d'autres AWS services qui vous aident à surveiller et à sécuriser vos AWS IoT Greengrass ressources.

Rubriques

- [Protection des données dans AWS IoT Greengrass](#)
- [Authentification et autorisation de l'appareil pour AWS IoT Greengrass](#)
- [Gestion des identités et des accès pour AWS IoT Greengrass](#)
- [Autoriser le trafic des appareils via un proxy ou un pare-feu](#)
- [Validation de conformité pour AWS IoT Greengrass](#)
- [Points de terminaison FIPS](#)

- [Résilience dans AWS IoT Greengrass](#)
- [Sécurité de l'infrastructure dans AWS IoT Greengrass](#)
- [Analyse de configuration et de vulnérabilité dans AWS IoT Greengrass](#)
- [Intégrité du code dans AWS IoT Greengrass V2](#)
- [AWS IoT Greengrass et points de terminaison VPC d'interface \(\)AWS PrivateLink](#)
- [Bonnes pratiques en matière de sécurité pour AWS IoT Greengrass](#)

Protection des données dans AWS IoT Greengrass

Le [modèle de responsabilité AWS partagée](#) de s'applique à la protection des données dans AWS IoT Greengrass. Comme décrit dans ce modèle, AWS est chargé de protéger l'infrastructure mondiale qui gère tous les AWS Cloud. 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 plus d'informations 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 d'AWS et RGPD \(Règlement général sur la protection des données\)](#) sur le Blog de sécuritéAWS .

À des fins de protection des données, nous vous recommandons de protéger les Compte AWS informations d'identification et de configurer les utilisateurs individuels avec AWS IAM Identity Center ou Gestion des identités et des accès AWS (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.
- SSL/TLS À utiliser pour communiquer avec AWS les ressources. Nous exigeons TLS 1.2 et recommandons TLS 1.3.
- Configurez l'API et la journalisation de l'activité des utilisateurs avec AWS CloudTrail. Pour plus d'informations sur l'utilisation des CloudTrail sentiers pour capturer AWS des activités, consultez la section [Utilisation des CloudTrail sentiers](#) dans le guide de AWS CloudTrail l'utilisateur.
- Utilisez des solutions de AWS chiffrement, ainsi que tous les contrôles de sécurité par défaut qu'ils contiennent 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 par la norme FIPS 140-3 pour accéder AWS via une interface de ligne de commande ou une API, utilisez un point de terminaison FIPS. Pour plus d'informations sur les points de terminaison FIPS disponibles, consultez [Norme FIPS \(Federal Information Processing Standard\) 140-3](#).

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 Nom. Cela inclut lorsque vous travaillez avec AWS IoT Greengrass ou d'autres Services AWS utilisateurs de la console, de l'API ou AWS SDKs. AWS CLI Toutes les données que vous entrez 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 dans AWS IoT Greengrass, consultez [the 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 Greengrass utilise le chiffrement pour protéger les données en transit (sur Internet ou sur un réseau local) et au repos (stockées dans le AWS Cloud).

Les appareils d'un AWS IoT Greengrass environnement collectent souvent des données qui sont envoyées aux AWS services pour un traitement ultérieur. Pour plus d'informations sur le chiffrement des données sur d'autres AWS services, consultez la documentation de sécurité de ce service.

Rubriques

- [Chiffrement en transit](#)
- [Chiffrement au repos](#)
- [Gestion des clés pour l'appareil principal Greengrass \(noyau\)](#)

Chiffrement en transit

AWS IoT Greengrass dispose de deux modes de communication lorsque les données sont en transit :

- [the section called “Données en transit sur Internet”](#). La communication entre un noyau Greengrass et AWS IoT Greengrass Internet est cryptée.
- [the section called “Données sur l'appareil principal \(noyau\)”](#). La communication entre les composants du périphérique principal de Greengrass n'est pas cryptée.

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 au AWS Cloud sont envoyées via une connexion TLS à l'aide des protocoles MQTT ou HTTPS. Elles sont donc sécurisées par défaut. AWS IoT Greengrass utilise le modèle AWS IoT de sécurité des transports. Pour de plus amples informations, veuillez consulter [Sécurité du transport](#) dans le Manuel du développeur AWS IoT Core .

Données sur l'appareil principal (noyau)

AWS IoT Greengrass ne chiffre pas les données échangées localement sur l'appareil principal de Greengrass car les données ne quittent pas l'appareil. Cela inclut la communication entre les composants définis par l'utilisateur, le SDK de l' AWS IoT appareil et les composants publics, tels que le gestionnaire de flux.

Chiffrement au repos

AWS IoT Greengrass stocke vos données :

- [the section called “Données au repos dans le AWS Cloud”](#). Ces données sont cryptées.
- [the section called “Données au repos sur le noyau Greengrass”](#). Ces données ne sont pas cryptées (sauf les copies locales de vos secrets).

Données au repos dans le AWS Cloud

AWS IoT Greengrass chiffre les données clients stockées dans le AWS Cloud. Ces données sont protégées à l'aide de AWS KMS clés 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 (s'il est activé) pour protéger les données au repos sur le cœur. Il est de votre responsabilité de sécuriser le système de fichiers et l'appareil.

Cependant, AWS IoT Greengrass crypte les copies locales de vos secrets récupérés à partir de AWS Secrets Manager. Pour plus d'informations, consultez le composant du [gestionnaire de secrets](#).

Gestion des clés pour l'appareil principal Greengrass (noyau)

Il est de la responsabilité du client de garantir le stockage sécurisé des clés cryptographiques (publiques et privées) sur l'appareil principal de Greengrass. AWS IoT Greengrass utilise des clés publiques et privées dans le scénario suivant :

- 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 de plus amples informations, veuillez consulter [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.

Un appareil Greengrass Core prend en charge le stockage de clés privées à l'aide des autorisations du système de fichiers ou d'un module [de sécurité matériel](#). Si vous utilisez des clés privées basées sur le système de fichiers, vous êtes responsable de leur stockage sécurisé sur l'appareil noyau.

Intégration de sécurité matérielle

Note

Cette fonctionnalité est disponible pour les versions 2.5.3 et ultérieures du composant [Greengrass](#) nucleus. AWS IoT Greengrass ne prend actuellement pas en charge cette fonctionnalité sur les appareils Windows principaux.

Vous pouvez configurer le logiciel AWS IoT Greengrass Core pour utiliser un module de sécurité matériel (HSM) via l'interface [PKCS #11](#). Cette fonctionnalité vous permet de stocker en toute

sécurité la clé privée et le certificat de l'appareil afin qu'ils ne soient pas exposés ou dupliqués dans le logiciel. Vous pouvez stocker la clé privée et le certificat sur un module matériel tel qu'un HSM ou un module de plateforme sécurisée (TPM).

Le logiciel AWS IoT Greengrass Core utilise une clé privée et un certificat X.509 pour authentifier les connexions aux services AWS IoT et AWS IoT Greengrass . Le [composant du gestionnaire de secrets](#) utilise cette clé privée pour chiffrer et déchiffrer en toute sécurité les secrets que vous déployez sur un appareil principal de Greengrass. Lorsque vous configurez un périphérique principal pour utiliser un HSM, ces composants utilisent la clé privée et le certificat que vous stockez dans le HSM.

Le [composant broker Moquette MQTT](#) stocke également une clé privée pour son certificat de serveur MQTT local. Ce composant stocke la clé privée sur le système de fichiers de l'appareil dans le dossier de travail du composant. Actuellement, AWS IoT Greengrass ne prend pas en charge le stockage de cette clé privée ou de ce certificat dans un HSM.

 Tip

Recherchez les appareils compatibles avec cette fonctionnalité dans le [catalogue d'appareils AWS partenaires](#).

Rubriques

- [Prérequis](#)
- [Bonnes pratiques en matière de sécurité matérielle](#)
- [Installation du logiciel AWS IoT Greengrass Core avec sécurité matérielle](#)
- [Configuration de la sécurité matérielle sur un périphérique principal existant](#)
- [Utiliser du matériel sans support PKCS #11](#)
- [Consultez aussi](#)

Prérequis

Vous devez satisfaire aux exigences suivantes pour utiliser un HSM sur un appareil principal Greengrass :

- [Greengrass nucleus](#) v2.5.3 ou version ultérieure installé sur le périphérique principal. Vous pouvez choisir une version compatible lorsque vous installez le logiciel AWS IoT Greengrass Core sur un appareil principal.
- Le [composant fournisseur PKCS #11](#) installé sur le périphérique principal. Vous pouvez télécharger et installer ce composant lorsque vous installez le logiciel AWS IoT Greengrass Core sur un appareil principal.
- Module de sécurité matérielle qui prend en charge le schéma de signature [PKCS #1 v1.5](#) et les clés RSA d'une taille de clé RSA-2048 (ou supérieure) ou les clés ECC.

 Note

Pour utiliser un module de sécurité matériel avec des clés ECC, vous devez utiliser [Greengrass](#) nucleus v2.5.6 ou version ultérieure.

Pour utiliser un module de sécurité matériel et un [gestionnaire de secrets](#), vous devez utiliser un module de sécurité matériel avec des clés RSA.

- Une bibliothèque de fournisseur PKCS #11 que le logiciel AWS IoT Greengrass Core peut charger au moment de l'exécution (à l'aide de libdl) pour appeler les fonctions PKCS #11. La bibliothèque du fournisseur PKCS #11 doit implémenter les opérations d'API PKCS #11 suivantes :
 - C_Initialize
 - C_Finalize
 - C_GetSlotList
 - C_GetSlotInfo
 - C_GetTokenInfo
 - C_OpenSession
 - C_GetSessionInfo
 - C_CloseSession
 - C_Login
 - C_Logout
 - C_GetAttributeValue
 - C_FindObjectsInit
 - C_FindObjects
 - C_FindObjectsFinal

- C_DecryptInit
- C_Decrypt
- C_DecryptUpdate
- C_DecryptFinal
- C_SignInit
- C_Sign
- C_SignUpdate
- C_SignFinal
- C_GetMechanismList
- C_GetMechanismInfo
- C_GetInfo
- C_GetFunctionList
- Le module matériel doit être résolu par étiquette d'emplacement, tel que défini dans la spécification PKCS#11.
- Vous devez stocker la clé privée et le certificat dans le HSM dans le même emplacement, et ils doivent utiliser la même étiquette d'objet et le même ID d'objet, si le HSM prend en charge l'objet. IDs
- Le certificat et la clé privée doivent pouvoir être résolus par des libellés d'objets.
- La clé privée doit disposer des autorisations suivantes :
 - sign
 - decrypt
- (Facultatif) Pour utiliser le [composant secret manager](#), vous devez utiliser la version 2.1.0 ou ultérieure, et la clé privée doit disposer des autorisations suivantes :
 - unwrap
 - wrap

Bonnes pratiques en matière de sécurité matérielle

Tenez compte des meilleures pratiques suivantes lorsque vous configurez la sécurité matérielle sur les appareils principaux de Greengrass.

- Générer des clés privées directement sur le HSM en utilisant le générateur de nombres aléatoires du matériel interne. Cette approche est plus sûre que l'importation d'une clé privée que vous générez ailleurs, car la clé privée reste dans le HSM.
- Configurez les clés privées pour qu'elles soient immuables et interdisez l'exportation.
- Utilisez l'outil de provisionnement recommandé par le fournisseur du matériel HSM pour générer une demande de signature de certificat (CSR) à l'aide de la clé privée protégée par le matériel, puis utilisez la AWS IoT console ou l'API pour générer un certificat client.

 Note

Les bonnes pratiques de sécurité relatives à la rotation des clés ne s'appliquent pas lorsque vous générez des clés privées sur un HSM.

Installation du logiciel AWS IoT Greengrass Core avec sécurité matérielle

Lorsque vous installez le logiciel AWS IoT Greengrass Core, vous pouvez le configurer pour utiliser une clé privée que vous générez dans un HSM. Cette approche suit les [meilleures pratiques de sécurité](#) pour générer la clé privée dans le HSM, afin que la clé privée reste dans le HSM.

Pour installer le logiciel AWS IoT Greengrass Core avec sécurité matérielle, procédez comme suit :

1. Générez une clé privée dans le HSM.
2. Créez une demande de signature de certificat (CSR) à partir de la clé privée.
3. Créez un certificat à partir du CSR. Vous pouvez créer un certificat signé par AWS IoT ou par une autre autorité de certification racine (CA). Pour plus d'informations sur l'utilisation d'une autre autorité de certification racine, voir [Création de vos propres certificats clients](#) dans le Guide du AWS IoT Core développeur.
4. Téléchargez le AWS IoT certificat et importez-le dans le HSM.
5. Installez le logiciel AWS IoT Greengrass Core à partir d'un fichier de configuration qui indique d'utiliser le composant fournisseur PKCS #11 ainsi que la clé privée et le certificat du HSM.

Vous pouvez choisir l'une des options d'installation suivantes pour installer le logiciel AWS IoT Greengrass Core en toute sécurité matérielle :

- Installation manuelle

Choisissez cette option pour créer manuellement les AWS ressources requises et configurer la sécurité matérielle. Pour de plus amples informations, veuillez consulter [Installation AWS IoT Greengrass du logiciel Core avec provisionnement manuel des ressources](#).

- Installation avec provisionnement personnalisé

Choisissez cette option pour développer une application Java personnalisée qui crée automatiquement les AWS ressources requises et configure la sécurité du matériel. Pour de plus amples informations, veuillez consulter [Installez le logiciel AWS IoT Greengrass Core avec un provisionnement personnalisé des ressources](#).

Actuellement, AWS IoT Greengrass ne prend pas en charge l'installation du logiciel AWS IoT Greengrass Core avec sécurité matérielle lorsque vous effectuez une [installation avec le provisionnement automatique des ressources ou le provisionnement](#) du [AWS IoT parc](#).

Configuration de la sécurité matérielle sur un périphérique principal existant

Vous pouvez importer la clé privée et le certificat d'un périphérique principal dans un HSM pour configurer la sécurité matérielle.

Considérations

- Vous devez disposer d'un accès root au système de fichiers de l'appareil principal.
- Dans cette procédure, vous arrêtez le logiciel AWS IoT Greengrass principal, de sorte que le périphérique principal est hors ligne et indisponible pendant que vous configurez la sécurité matérielle.

Pour configurer la sécurité matérielle sur un périphérique principal existant, procédez comme suit :

1. Initialisez le HSM.
2. Déployez le [composant fournisseur PKCS #11](#) sur le périphérique principal.
3. Arrêtez le logiciel AWS IoT Greengrass Core.
4. Importez la clé privée et le certificat de l'appareil principal dans le HSM.
5. Mettez à jour le fichier de configuration du logiciel AWS IoT Greengrass Core pour utiliser la clé privée et le certificat du HSM.
6. Démarrez le logiciel AWS IoT Greengrass Core.

Étape 1 : Initialisation du module de sécurité matérielle

Procédez comme suit pour initialiser le HSM sur votre appareil principal.

Pour initialiser le module de sécurité matérielle

- Initialisez un jeton PKCS #11 dans le HSM et enregistrez l'ID de slot et le code PIN utilisateur associés au jeton. Consultez la documentation de votre HSM pour savoir comment initialiser un jeton. Vous utiliserez l'ID de slot et le code PIN de l'utilisateur ultérieurement lorsque vous déployez et configurez le composant fournisseur PKCS #11.

Étape 2 : Déployer le composant fournisseur PKCS #11

Procédez comme suit pour déployer et configurer le [composant fournisseur PKCS #11](#). Vous pouvez déployer le composant sur un ou plusieurs appareils principaux.

Pour déployer le composant fournisseur PKCS #11 (console)

- Dans le menu de navigation de la [AWS IoT Greengrass console](#), sélectionnez Composants.
- Sur la page Composants, choisissez l'onglet Composants publics, puis choisissez `aws.greengrass.crypto.Pkcs11Provider`.
- Dans la page `aws.greengrass.crypto.Pkcs11Provider`, choisissez Déployer.
- Dans Ajouter au déploiement, choisissez un déploiement existant à réviser ou choisissez de créer un nouveau déploiement, puis choisissez Suivant.
- Si vous avez choisi de créer un nouveau déploiement, choisissez le périphérique principal ou le groupe d'objets cible pour le déploiement. Sur la page Spécifier la cible, sous Cible de déploiement, choisissez un périphérique principal ou un groupe d'objets, puis cliquez sur Suivant.
- Sur la page Sélectionner les composants, sous Composants publics, sélectionnez `aws.greengrass.crypto.Pkcs11Provider`, puis choisissez Next.
- Sur la page Configurer les composants, sélectionnez `aws.greengrass.crypto.Pkcs11Provider`, puis effectuez les opérations suivantes :
 - Choisissez Configure component (Configurer un composant).
 - Dans la configuration `aws.greengrass.crypto.Pkcs11Provider` modal, sous Mise à jour de la configuration, dans Configuration à fusionner, entrez la mise à jour de configuration suivante. Mettez à jour les paramètres de configuration suivants avec les valeurs des

équipements principaux cibles. Spécifiez l'ID du slot et le code PIN de l'utilisateur dans lesquels vous avez initialisé le jeton PKCS #11 plus tôt. Vous importez la clé privée et le certificat dans cet emplacement du HSM ultérieurement.

`name`

Nom de la configuration PKCS #11.

`library`

Le chemin de fichier absolu vers la bibliothèque de l'implémentation PKCS #11 que le logiciel AWS IoT Greengrass Core peut charger avec libdl.

`slot`

L'ID du slot qui contient la clé privée et le certificat de l'appareil. Cette valeur est différente de l'index ou de l'étiquette de l'emplacement.

`userPin`

Le code PIN de l'utilisateur à utiliser pour accéder au slot.

```
{
  "name": "softhsm_pkcs11",
  "library": "/usr/lib/softhsm/libsofthsm2.so",
  "slot": 1,
  "userPin": "1234"
}
```

- c. Choisissez Confirmer pour fermer le modal, puis cliquez sur Suivant.
8. Sur la page Configure advanced settings (Configurer les paramètres avancés), conservez les paramètres de configuration par défaut et choisissez Next (Suivant).
9. Sur la page Review (Révision), choisissez Deploy (Déployer).

Le déploiement peut prendre jusqu'à une minute.

Pour déployer le composant fournisseur PKCS #11 ()AWS CLI

Pour déployer le composant fournisseur PKCS #11, créez un document de déploiement incluant `aws.greengrass.crypto.Pkcs11Provider` l'componentsobjet et spécifiez la mise à jour

de configuration pour le composant. Suivez les instructions [Créer des déploiements](#) pour créer un nouveau déploiement ou modifier un déploiement existant.

L'exemple de document de déploiement partiel suivant indique de déployer et de configurer le composant fournisseur PKCS #11. Mettez à jour les paramètres de configuration suivants avec les valeurs des équipements principaux cibles. Enregistrez l'ID du slot et le code PIN de l'utilisateur pour les utiliser ultérieurement lors de l'importation de la clé privée et du certificat dans le HSM.

`name`

Nom de la configuration PKCS #11.

`library`

Le chemin de fichier absolu vers la bibliothèque de l'implémentation PKCS #11 que le logiciel AWS IoT Greengrass Core peut charger avec libdl.

`slot`

L'ID du slot qui contient la clé privée et le certificat de l'appareil. Cette valeur est différente de l'index ou de l'étiquette de l'emplacement.

`userPin`

Le code PIN de l'utilisateur à utiliser pour accéder au slot.

```
{
  "name": "softhsm_pkcs11",
  "library": "/usr/lib/softhsm/libsofthsm2.so",
  "slot": 1,
  "userPin": "1234"
}
```

```
{
  ...,
  "components": {
    ...,
    "aws.greengrass.crypto.Pkcs11Provider": {
      "componentVersion": "2.0.0",
      "configurationUpdate": {
        "merge": "{\"name\":\"softhsm_pkcs11\",\"library\":\"/usr/lib/softhsm/libsofthsm2.so\",\"slot\":1,\"userPin\":\"1234\"}"
      }
    }
  }
}
```

```
}  
}  
}  
}
```

L'exécution du déploiement peut prendre plusieurs minutes. Vous pouvez utiliser le AWS IoT Greengrass service pour vérifier l'état du déploiement. Vous pouvez consulter les journaux du logiciel AWS IoT Greengrass Core pour vérifier que le composant fournisseur PKCS #11 se déploie correctement. Pour plus d'informations, consultez les ressources suivantes :

- [Vérifier l'état du déploiement](#)
- [AWS IoT Greengrass Journaux de surveillance](#)

Si le déploiement échoue, vous pouvez résoudre les problèmes de déploiement sur chaque appareil principal. Pour de plus amples informations, veuillez consulter [Résolution des problèmes AWS IoT Greengrass V2](#).

Étape 3 : mise à jour de la configuration sur le périphérique principal

Le logiciel AWS IoT Greengrass Core utilise un fichier de configuration qui indique le mode de fonctionnement de l'appareil. Ce fichier de configuration indique où trouver la clé privée et le certificat que l'appareil utilise pour se connecter au AWS Cloud. Procédez comme suit pour importer la clé privée et le certificat du périphérique principal dans le HSM et mettre à jour le fichier de configuration pour utiliser le HSM.

Pour mettre à jour la configuration sur le périphérique principal afin d'utiliser la sécurité matérielle

1. Arrêtez le logiciel AWS IoT Greengrass Core. Si vous avez [configuré le logiciel AWS IoT Greengrass Core en tant que service système](#) avec systemd, vous pouvez exécuter la commande suivante pour arrêter le logiciel.

```
sudo systemctl stop greengrass.service
```

2. Trouvez la clé privée et les fichiers de certificat de l'appareil principal.
 - Si vous avez installé le logiciel AWS IoT Greengrass Core avec provisionnement [automatique ou provisionnement](#) de [flotte](#), la clé privée existe à `/greengrass/v2/privKey.key` l'adresse et le certificat à l'adresse. `/greengrass/v2/thingCert.crt`

- Si vous avez installé le logiciel AWS IoT Greengrass Core avec un [provisionnement manuel](#), la clé privée existe `/greengrass/v2/private.pem.key` par défaut et le certificat existe `/greengrass/v2/device.pem.crt` par défaut.

Vous pouvez également vérifier les `system.certificateFilePath` propriétés `system.privateKeyPath` et `/greengrass/v2/config/effectiveConfig.yaml` pour trouver l'emplacement de ces fichiers.

3. Importez la clé privée et le certificat dans le HSM. Consultez la documentation de votre HSM pour savoir comment y importer des clés privées et des certificats. Importez la clé privée et le certificat à l'aide de l'ID du slot et du code PIN utilisateur sur lesquels vous avez initialisé le jeton PKCS #11 plus tôt. Vous devez utiliser le même libellé et le même identifiant d'objet pour la clé privée et le même certificat. Enregistrez l'étiquette d'objet que vous spécifiez lors de l'importation de chaque fichier. Vous utiliserez cette étiquette ultérieurement lorsque vous mettrez à jour la configuration logicielle AWS IoT Greengrass principale pour utiliser la clé privée et le certificat du HSM.
4. Mettez à jour la configuration AWS IoT Greengrass principale pour utiliser la clé privée et le certificat dans le HSM. Pour mettre à jour la configuration, vous devez modifier le fichier de configuration AWS IoT Greengrass Core et exécuter le logiciel AWS IoT Greengrass Core avec le fichier de configuration mis à jour pour appliquer la nouvelle configuration.

Procédez comme suit :

- a. Créez une sauvegarde du fichier de configuration AWS IoT Greengrass Core. Vous pouvez utiliser cette sauvegarde pour restaurer le périphérique principal si vous rencontrez des problèmes lors de la configuration de la sécurité matérielle.

```
sudo cp /greengrass/v2/config/effectiveConfig.yaml ~/ggc-config-backup.yaml
```

- b. Ouvrez le fichier de configuration AWS IoT Greengrass Core dans un éditeur de texte. Par exemple, vous pouvez exécuter la commande suivante pour utiliser GNU nano pour modifier le fichier. Remplacez `/greengrass/v2` par le chemin d'accès au dossier racine de Greengrass.

```
sudo nano /greengrass/v2/config/effectiveConfig.yaml
```

- c. Remplacez la valeur de `system.privateKeyPath` par l'URI PKCS #11 pour la clé privée dans le HSM. `iotdevicekey` Remplacez-le par le libellé de l'objet dans lequel vous avez importé la clé privée et le certificat précédemment.

```
pkcs11:object=iotdevicekey;type=private
```

- d. Remplacez la valeur de `system.certificateFilePath` par l'URI PKCS #11 pour le certificat dans le HSM. `iotdevicekey` Remplacez-le par le libellé de l'objet dans lequel vous avez importé la clé privée et le certificat précédemment.

```
pkcs11:object=iotdevicekey;type=cert
```

Une fois ces étapes terminées, la `system` propriété du fichier de configuration AWS IoT Greengrass Core devrait ressembler à celle de l'exemple suivant.

```
system:
  certificateFilePath: "pkcs11:object=iotdevicekey;type=cert"
  privateKeyPath: "pkcs11:object=iotdevicekey;type=private"
  rootCaPath: "/greengrass/v2/rootCA.pem"
  rootpath: "/greengrass/v2"
  thingName: "MyGreengrassCore"
```

5. Appliquez la configuration dans le `effectiveConfig.yaml` fichier mis à jour. Exécutez `Greengrass.jar` avec le `--init-config` paramètre pour appliquer la configuration dans `effectiveConfig.yaml`. Remplacez `/greengrass/v2` par le chemin d'accès au dossier racine de Greengrass.

```
sudo java -Droot="/greengrass/v2" \
  -jar "/greengrass/v2/alts/current/distro/lib/Greengrass.jar" \
  --start false \
  --init-config "/greengrass/v2/config/effectiveConfig.yaml"
```

6. Démarrez le logiciel AWS IoT Greengrass Core. Si vous avez [configuré le logiciel AWS IoT Greengrass Core en tant que service système](#) avec `systemd`, vous pouvez exécuter la commande suivante pour démarrer le logiciel.

```
sudo systemctl start greengrass.service
```


Pour de plus amples informations, veuillez consulter [Exécutez le logiciel AWS IoT Greengrass Core](#).

7. Consultez les journaux du logiciel AWS IoT Greengrass Core pour vérifier que le logiciel démarre et se connecte au AWS Cloud. Le logiciel AWS IoT Greengrass Core utilise la clé privée et le certificat pour se connecter aux AWS IoT Greengrass services AWS IoT et.

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

Les messages du journal au niveau des informations suivants indiquent que le logiciel AWS IoT Greengrass Core se connecte correctement aux services AWS IoT et AWS IoT Greengrass .

```
2021-12-06T22:47:53.702Z [INFO] (Thread-3)
com.aws.greengrass.mqttclient.AwsIotMqttClient: Successfully connected to AWS IoT
Core. {clientId=MyGreengrassCore5, sessionPresent=false}
```

8. (Facultatif) Après avoir vérifié que le logiciel AWS IoT Greengrass Core fonctionne avec la clé privée et le certificat du HSM, supprimez les fichiers de clé privée et de certificat du système de fichiers de l'appareil. Exécutez la commande suivante et remplacez les chemins de fichiers par les chemins d'accès aux fichiers de clé privée et de certificat.

```
sudo rm /greengrass/v2/privKey.key
sudo rm /greengrass/v2/thingCert.crt
```

Utiliser du matériel sans support PKCS #11

La bibliothèque PKCS # 11 est généralement fournie par le fournisseur de matériel ou est en open source. Par exemple, avec du matériel conforme aux normes (tel que TPM1 .2), il peut être possible d'utiliser des logiciels open source existants. Toutefois, si votre matériel ne dispose pas d'une implémentation de bibliothèque PKCS #11 correspondante, ou si vous souhaitez créer un fournisseur PKCS #11 personnalisé, contactez votre représentant du support Amazon Web Services Enterprise pour toute question relative à l'intégration.

Consultez aussi

- [Guide d'utilisation de l'interface à jetons cryptographiques PKCS #11, version 2.4.0](#)
- [RFC 7512](#)
- [PKCS #1 : version de chiffrement RSA 1.5](#)

Authentification et autorisation de l'appareil pour AWS IoT Greengrass

Les appareils dans AWS IoT Greengrass les environnements utilisent des certificats X.509 pour l'authentification et des AWS IoT politiques d'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.

AWS IoT les politiques définissent l'ensemble des opérations autorisées pour les AWS IoT appareils. Plus précisément, ils autorisent et interdisent l'accès aux AWS IoT Core opérations du plan de AWS IoT Greengrass données, telles que la publication de messages MQTT et la récupération des ombres des appareils.

Tous les appareils nécessitent une entrée dans le AWS IoT Core registre et un certificat X.509 activé avec une AWS IoT politique associée. Les appareils se répartissent en deux catégories :

- Appareils Greengrass Core

Les appareils Greengrass Core utilisent des certificats et des AWS IoT politiques pour se connecter à AWS IoT Core et. AWS IoT Greengrass Les certificats et les politiques permettent également AWS IoT Greengrass de déployer des composants et des configurations sur les appareils principaux.

- Appareils clients

Les appareils clients MQTT 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 AWS IoT Greengrass cloud discovery pour rechercher et se connecter à un appareil principal de Greengrass. Un appareil client utilise le même certificat pour se connecter au service AWS IoT Core cloud et aux appareils principaux. Les appareils clients utilisent également les informations de découverte pour l'authentification mutuelle avec le périphérique principal. Pour de plus amples informations, veuillez consulter [Interagissez avec les appareils IoT locaux](#).

Certificats X.509

La communication entre les appareils principaux et les appareils 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 AWS IoT Greengrass environnement, les appareils utilisent des certificats avec des clés publiques et privées pour les connexions TLS (Transport Layer Security) suivantes :

- Le composant AWS IoT client de l'appareil principal de Greengrass qui se connecte à Internet AWS IoT Core et AWS IoT Greengrass via Internet.
- Appareils clients qui se AWS IoT Greengrass connectent via Internet pour découvrir les appareils principaux.
- Le composant broker MQTT sur le cœur de Greengrass se connectant aux appareils Greengrass du groupe via le réseau local.

AWS IoT Greengrass les appareils principaux stockent les certificats dans le dossier racine de Greengrass.

Certificats d'autorité de certification (CA)

Les appareils principaux et les appareils clients de Greengrass téléchargent un certificat CA racine utilisé pour l'authentification auprès des services AWS IoT Core et AWS IoT Greengrass . 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 .

Les appareils clients téléchargent également un certificat CA du périphérique principal Greengrass. Ils utilisent ce certificat pour valider le certificat du serveur MQTT sur le périphérique principal lors de l'authentification mutuelle.

Rotation des certificats sur le broker MQTT local

Lorsque vous [activez la prise en charge des appareils clients](#), les appareils principaux de Greengrass génèrent un certificat de serveur MQTT local que les appareils clients utilisent pour l'authentification mutuelle. Ce certificat est signé par le certificat CA de l'appareil principal, que celui-ci stocke dans le AWS IoT Greengrass cloud. Les appareils clients récupèrent le certificat CA du périphérique principal lorsqu'ils découvrent le périphérique principal. Ils utilisent le certificat CA du périphérique

principal pour vérifier le certificat du serveur MQTT du périphérique principal lorsqu'ils se connectent au périphérique principal. Le certificat CA de l'appareil principal expire au bout de 5 ans.

Le certificat du serveur MQTT expire tous les 7 jours par défaut, et vous pouvez configurer cette durée entre 2 et 10 jours. Cette période limitée est basée sur les bonnes pratiques en matière de sécurité. Cette rotation permet d'atténuer le risque qu'un attaquant vole le certificat du serveur MQTT et la clé privée pour se faire passer pour le périphérique principal de Greengrass.

Le périphérique principal de Greengrass fait pivoter le certificat du serveur MQTT 24 heures avant son expiration. Le périphérique principal Greengrass génère un nouveau certificat et redémarre le broker MQTT local. Dans ce cas, tous les appareils clients connectés au périphérique principal de Greengrass sont déconnectés. Les appareils clients peuvent se reconnecter au périphérique principal de Greengrass après un court laps de temps.

AWS IoT politiques relatives aux opérations du plan de données

Utilisez AWS IoT des politiques pour autoriser l'accès aux plans de AWS IoT Greengrass données AWS IoT Core et. Le plan de AWS IoT Core données fournit des opérations pour les appareils, les utilisateurs et les applications. Ces opérations incluent la possibilité de se connecter à des sujets AWS IoT Core et de s'y abonner. Le plan AWS IoT Greengrass de données fournit des opérations pour les appareils Greengrass. Pour de plus amples informations, veuillez consulter [AWS IoT Greengrass V2 actions politiques](#). Ces opérations incluent la capacité de résoudre les dépendances des composants et de télécharger des artefacts de composants publics.

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 IoT les 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 la ressource ARNs](#) du Guide de l'Gestion des identités et des accès AWS utilisateur.

Pour de plus amples informations, veuillez consulter [Stratégies AWS IoT](#) et [Actions de stratégie AWS IoT](#) dans le Manuel du développeur AWS IoT Core .

 Important

Les variables de politique des objets (`iot:Connection.Thing.*`) ne sont pas prises en charge dans AWS IoT les politiques relatives aux appareils principaux ou aux opérations du plan de données Greengrass. Vous pouvez plutôt utiliser un caractère générique correspondant à plusieurs appareils portant des noms similaires. Par exemple, vous pouvez spécifier `MyGreengrassDevice*` de correspondre `MyGreengrassDevice1` `MyGreengrassDevice2`, et ainsi de suite.

 Note

AWS IoT Core vous 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.

AWS IoT Greengrass V2 actions politiques

AWS IoT Greengrass V2 définit les actions politiques suivantes que les appareils principaux et clients de Greengrass peuvent utiliser dans AWS IoT les politiques. Pour spécifier une ressource pour une action de politique, vous utilisez le Amazon Resource Name (ARN) de la ressource.

Principales actions de l'appareil

`greengrass:GetComponentVersionArtifact`

Accorde l'autorisation d'obtenir une URL présignée pour télécharger un artefact de composant public ou un artefact de composant Lambda.

Cette autorisation est évaluée lorsqu'un périphérique principal reçoit un déploiement qui spécifie un composant public ou un Lambda contenant des artefacts. Si l'appareil principal possède déjà l'artefact, il ne le télécharge pas à nouveau.

Type de ressource : `componentVersion`

Format ARN de la ressource : `arn:aws:greengrass:region:account-id:components:component-name:versions:component-version`

`greengrass:ResolveComponentCandidates`

Accorde l'autorisation d'identifier une liste de composants répondant aux exigences relatives aux composants, à la version et à la plate-forme pour un déploiement. En cas de conflit entre les exigences ou s'il n'existe aucun composant répondant aux exigences, cette opération renvoie une erreur et le déploiement échoue sur le périphérique.

Cette autorisation est évaluée lorsqu'un périphérique principal reçoit un déploiement qui spécifie des composants.

Type de ressource : Aucune

Format ARN de la ressource : *

`greengrass:GetDeploymentConfiguration`

Accorde l'autorisation d'obtenir une URL présignée pour télécharger un document de déploiement volumineux.

Cette autorisation est évaluée lorsqu'un périphérique principal reçoit un déploiement qui spécifie un document de déploiement supérieur à 7 Ko (si le déploiement cible un objet) ou 31 Ko (si le déploiement cible un groupe d'objets). Le document de déploiement inclut les configurations des composants, les politiques de déploiement et les métadonnées de déploiement. Pour de plus amples informations, veuillez consulter [Déployer AWS IoT Greengrass des composants sur des appareils](#).

Cette fonctionnalité est disponible pour les versions 2.3.0 et ultérieures du composant [Greengrass](#) noyau.

Type de ressource : Aucune

Format ARN de la ressource : *

`greengrass:ListThingGroupsForCoreDevice`

Accorde l'autorisation d'obtenir la hiérarchie des groupes d'objets d'un appareil principal.

Cette autorisation est vérifiée lorsqu'un périphérique principal reçoit un déploiement de AWS IoT Greengrass. Le périphérique principal utilise cette action pour déterminer s'il a été supprimé d'un groupe d'objets depuis le dernier déploiement. Si le périphérique principal a été supprimé d'un groupe d'objets et que ce groupe d'objets est la cible d'un déploiement sur le périphérique principal, le périphérique principal supprime les composants installés par ce déploiement.

Cette fonctionnalité est utilisée par les versions 2.5.0 et ultérieures du composant [Greengrass nucleus](#).

Type de ressource : thing (appareil principal)

Format ARN de la ressource : `arn:aws:iot:region:account-id:thing/core-device-thing-name`

`greengrass:VerifyClientDeviceIdentity`

Accorde l'autorisation de vérifier l'identité d'un appareil client qui se connecte à un appareil principal.

Cette autorisation est évaluée lorsqu'un périphérique principal exécute le [composant d'authentification du périphérique client](#) et reçoit une connexion MQTT d'un périphérique client. L'appareil client présente son certificat d' AWS IoT appareil. Ensuite, le périphérique principal envoie le certificat de l'appareil au service AWS IoT Greengrass cloud pour vérifier l'identité de l'appareil client. Pour de plus amples informations, veuillez consulter [Interagissez avec les appareils IoT locaux](#).

Type de ressource : Aucune

Format ARN de la ressource : *

`greengrass:VerifyClientDeviceIoTCertificateAssociation`

Accorde l'autorisation de vérifier si un appareil client est associé à un AWS IoT certificat.

Cette autorisation est évaluée lorsqu'un périphérique principal exécute le [composant d'authentification du périphérique client](#) et autorise un périphérique client à se connecter via MQTT. Pour de plus amples informations, veuillez consulter [Interagissez avec les appareils IoT locaux](#).

 Note

Pour qu'un appareil principal puisse utiliser cette opération, le rôle de [service Greengrass doit être associé à votre rôle](#) `iot:DescribeCertificate` Compte AWS et autoriser cette autorisation.

Type de ressource : thing (appareil client)

Format ARN de la ressource : `arn:aws:iot:region:account-id:thing/client-device-thing-name`

`greengrass:PutCertificateAuthorities`

Accorde l'autorisation de télécharger des certificats d'autorité de certification (CA) que les appareils clients peuvent télécharger pour vérifier le périphérique principal.

Cette autorisation est évaluée lorsqu'un périphérique principal installe et exécute le composant d'[authentification du périphérique client](#). Ce composant crée une autorité de certification locale et utilise cette opération pour télécharger ses certificats CA. Les appareils clients téléchargent ces certificats CA lorsqu'ils utilisent l'opération [Discover](#) pour trouver les périphériques principaux auxquels ils peuvent se connecter. Lorsque les appareils clients se connectent à un courtier MQTT sur un périphérique principal, ils utilisent ces certificats CA pour vérifier l'identité du périphérique principal. Pour de plus amples informations, veuillez consulter [Interagissez avec les appareils IoT locaux](#).

Type de ressource : Aucune

Format de l'ARN : *

`greengrass:GetConnectivityInfo`

Accorde l'autorisation d'obtenir des informations de connectivité pour un appareil principal. Ces informations décrivent comment les appareils clients peuvent se connecter au périphérique principal.

Cette autorisation est évaluée lorsqu'un périphérique principal installe et exécute le composant d'[authentification du périphérique client](#). Ce composant utilise les informations de connectivité pour générer des certificats CA valides à télécharger sur le service AWS IoT Greengrass cloud lors de l'[PutCertificateAuthorities](#) opération. Les appareils clients utilisent ces certificats CA pour vérifier l'identité du périphérique principal. Pour de plus amples informations, veuillez consulter [Interagissez avec les appareils IoT locaux](#).

Vous pouvez également utiliser cette opération sur le plan AWS IoT Greengrass de contrôle pour afficher les informations de connectivité d'un périphérique principal. Pour plus d'informations, consultez [GetConnectivityInfo](#) dans la Référence d'API AWS IoT Greengrass V1 .

Type de ressource : thing (appareil principal)

Format ARN de la ressource : `arn:aws:iot:region:account-id:thing/core-device-thing-name`

greengrass:UpdateConnectivityInfo

Accorde l'autorisation de mettre à jour les informations de connectivité d'un appareil principal. Ces informations décrivent comment les appareils clients peuvent se connecter au périphérique principal.

Cette autorisation est évaluée lorsqu'un périphérique principal exécute le [composant de détection IP](#). Ce composant identifie les informations dont les appareils clients ont besoin pour se connecter au périphérique principal sur le réseau local. Ce composant utilise ensuite cette opération pour télécharger les informations de connectivité vers le service AWS IoT Greengrass cloud, afin que les appareils clients puissent récupérer ces informations avec l'opération [Discover](#). Pour de plus amples informations, veuillez consulter [Interagissez avec les appareils IoT locaux](#).

Vous pouvez également utiliser cette opération sur le plan de AWS IoT Greengrass contrôle pour mettre à jour manuellement les informations de connectivité d'un périphérique principal. Pour plus d'informations, consultez [UpdateConnectivityInfo](#) dans la Référence d'API AWS IoT Greengrass V1 .

Type de ressource : thing (appareil principal)

Format ARN de la ressource : `arn:aws:iot:region:account-id:thing/core-device-thing-name`

Actions relatives à l'appareil client

greengrass:Discover

Accorde l'autorisation de découvrir les informations de connectivité pour les appareils principaux auxquels un appareil client peut se connecter. Ces informations décrivent comment le périphérique client peut se connecter aux périphériques principaux. Un appareil client ne peut découvrir que les périphériques principaux auxquels vous l'avez associé à l'aide de cette [BatchAssociateClientDeviceWithCoreDevice](#) opération. Pour de plus amples informations, veuillez consulter [Interagissez avec les appareils IoT locaux](#).

Type de ressource : thing (appareil client)

Format ARN de la ressource : `arn:aws:iot:region:account-id:thing/client-device-thing-name`

Mettre à jour la AWS IoT politique d'un appareil principal

Vous pouvez utiliser les AWS IoT consoles AWS IoT Greengrass et ou l' AWS IoT API pour consulter et mettre à jour la AWS IoT politique d'un appareil principal.

Note

Si vous avez utilisé le [programme d'installation du logiciel AWS IoT Greengrass Core pour provisionner des ressources](#), votre appareil principal dispose d'une AWS IoT politique qui autorise l'accès à toutes les AWS IoT Greengrass actions (`greengrass:*`). Vous pouvez suivre ces étapes pour restreindre l'accès aux seules actions utilisées par un appareil principal.

Révision et mise à jour de la AWS IoT politique d'un appareil principal (console)

1. Dans le menu de navigation de la [AWS IoT Greengrass console](#), choisissez Core devices.
2. Sur la page des appareils principaux, choisissez le périphérique principal à mettre à jour.
3. Sur la page de détails de l'appareil principal, choisissez le lien vers l'objet de l'appareil principal. Ce lien ouvre la page des détails de l'objet dans la AWS IoT console.
4. Sur la page des détails de l'objet, sélectionnez Certificats.
5. Dans l'onglet Certificats, choisissez le certificat actif de l'objet.
6. Sur la page des détails du certificat, sélectionnez Politiques.
7. Dans l'onglet Politiques, choisissez la AWS IoT politique à revoir et à mettre à jour. Vous pouvez ajouter les autorisations requises à n'importe quelle politique associée au certificat actif de l'appareil principal.

Note

Si vous avez utilisé le [programme d'installation du logiciel AWS IoT Greengrass Core pour provisionner des ressources](#), vous avez deux AWS IoT règles. Nous vous recommandons de choisir la politique nommée `GreengrassV2IoTThingPolicy`, si elle existe. Les appareils principaux que vous créez avec le programme d'installation rapide utilisent ce nom de politique par défaut. Si vous ajoutez des autorisations à cette politique, vous les accordez également aux autres appareils principaux qui utilisent cette politique.

8. Dans l'aperçu des politiques, choisissez Modifier la version active.
9. Passez en revue la politique et ajoutez, supprimez ou modifiez des autorisations selon vos besoins.
10. 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.
11. Sélectionnez Enregistrer comme nouvelle version.

Révision et mise à jour de la AWS IoT politique d'un appareil principal (AWS CLI)

1. Énumérez les principes de base de l'appareil AWS IoT . Les principaux de l'objet peuvent être des certificats de périphérique X.509 ou d'autres identifiants. Exécutez la commande suivante et *MyGreengrassCore* remplacez-la par le nom du périphérique principal.

```
aws iot list-thing-principals --thing-name MyGreengrassCore
```

L'opération renvoie une réponse répertoriant les principaux éléments du périphérique principal.

```
{
  "principals": [
    "arn:aws:iot:us-west-2:123456789012:cert/certificateId"
  ]
}
```

2. Identifiez le certificat actif de l'appareil principal. Exécutez la commande suivante et remplacez-la *certificateId* par l'ID de chaque certificat de l'étape précédente jusqu'à ce que vous trouviez le certificat actif. L'ID du certificat est la chaîne hexadécimale à la fin de l'ARN du certificat. L'--query argument indique de n'afficher que le statut du certificat.

```
aws iot describe-certificate --certificate-id certificateId --query
'certificateDescription.status'
```

L'opération renvoie le statut du certificat sous forme de chaîne. Par exemple, si le certificat est actif, cette opération produit des résultats "ACTIVE".

3. AWS IoT Répertoriez les politiques associées au certificat. Exécutez la commande suivante et remplacez l'ARN du certificat par l'ARN du certificat.

```
aws iot list-principal-policies --principal arn:aws:iot:us-west-2:123456789012:cert/certificateId
```

L'opération renvoie une réponse répertoriant les AWS IoT politiques associées au certificat.

```
{
  "policies": [
    {
      "policyName":
        "GreengrassTESCertificatePolicyMyGreengrassCoreTokenExchangeRoleAlias",
      "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassTESCertificatePolicyMyGreengrassCoreTokenExchangeRoleAlias"
    },
    {
      "policyName": "GreengrassV2IoTThingPolicy",
      "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassV2IoTThingPolicy"
    }
  ]
}
```

4. Choisissez la politique à consulter et à mettre à jour.

Note

Si vous avez utilisé le [programme d'installation du logiciel AWS IoT Greengrass Core pour provisionner des ressources](#), vous avez deux AWS IoT règles. Nous vous recommandons de choisir la politique nommée GreengrassV2IoTThingPolicy, si elle existe. Les appareils principaux que vous créez avec le programme d'installation rapide utilisent ce nom de politique par défaut. Si vous ajoutez des autorisations à cette politique, vous les accordez également aux autres appareils principaux qui utilisent cette politique.

5. Obtenez le document de la politique. Exécutez la commande suivante et *GreengrassV2IoTThingPolicy* remplacez-la par le nom de la politique.

```
aws iot get-policy --policy-name GreengrassV2IoTThingPolicy
```

L'opération renvoie une réponse contenant le document de la politique et d'autres informations relatives à la politique. Le document de politique est un objet JSON sérialisé sous forme de chaîne.

```
{
  "policyName": "GreengrassV2IoTThingPolicy",
  "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/GreengrassV2IoTThingPolicy",
  "policyDocument": "{\\
    \\\"Version\\\": \\\"2012-10-17\\\",\\
    \\\"Statement\\\": [\\
      {\\
        \\\"Effect\\\": \\\"Allow\\\",\\
        \\\"Action\\\": [\\
          \\\"iot:Connect\\\",\\
          \\\"iot:Publish\\\",\\
          \\\"iot:Subscribe\\\",\\
          \\\"iot:Receive\\\",\\
          \\\"greengrass:*\\\"\\
        ],\\
        \\\"Resource\\\": \\\"*\\\"\\
      }\\
    ]\\
  }\",
  "defaultVersionId": "1",
  "creationDate": "2021-02-05T16:03:14.098000-08:00",
  "lastModifiedDate": "2021-02-05T16:03:14.098000-08:00",
  "generationId":
    "f19144b798534f52c619d44f771a354f1b957dfa2b850625d9f1d0fde530e75f"
}
```

6. Utilisez un convertisseur en ligne ou un autre outil pour convertir la chaîne du document de politique en objet JSON, puis enregistrez-la dans un fichier nommé `iot-policy.json`.

Par exemple, si l'outil [jq](#) est installé, vous pouvez exécuter la commande suivante pour obtenir le document de politique, le convertir en objet JSON et enregistrer le document de politique en tant qu'objet JSON.

```
aws iot get-policy --policy-name GreengrassV2IoTThingPolicy --query
'policyDocument' | jq fromjson >> iot-policy.json
```

7. Passez en revue le document de politique et ajoutez, supprimez ou modifiez des autorisations selon les besoins.

Par exemple, sur un système basé sur Linux, vous pouvez exécuter la commande suivante pour utiliser GNU nano pour ouvrir le fichier.

```
nano iot-policy.json
```

Lorsque vous aurez terminé, le document de politique peut ressembler à la [AWS IoT politique minimale pour les appareils principaux](#).

8. Enregistrez les modifications en tant que nouvelle version de la politique. Exécutez la commande suivante et *GreengrassV2IoTThingPolicy* remplacez-la par le nom de la politique.

```
aws iot create-policy-version --policy-name GreengrassV2IoTThingPolicy --policy-document file://iot-policy.json --set-as-default
```

L'opération renvoie une réponse similaire à l'exemple suivant en cas de succès.

```
{
  "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/GreengrassV2IoTThingPolicy",
  "policyDocument": "{\\
    \\\"Version\\\": \\\"2012-10-17\\\",\\
    \\\"Statement\\\": [\\
      {\\
        \\\"Effect\\\": \\\"Allow\\\",\\
        \\\"Action\\\": [\\
          \\\"iot:Connect\\\",\\
          \\\"iot:Publish\\\",\\
          \\\"iot:Subscribe\\\",\\
          \\\"iot:Receive\\\",\\
          \\\"greengrass:*\\\"\\
        ],\\
        \\\"Resource\\\": \\\"*\\\"\\
      }\\
    ]\\
  }\",
  "policyVersionId": "2",
  "isDefaultVersion": true
}
```

AWS IoT Politique minimale pour les appareils AWS IoT Greengrass V2 principaux

Important

Les versions ultérieures du [composant Greengrass nucleus](#) nécessitent des autorisations supplémentaires sur la politique minimale AWS IoT . Vous devrez peut-être [mettre à jour les AWS IoT politiques de vos appareils principaux](#) pour accorder des autorisations supplémentaires.

- Les appareils principaux qui exécutent Greengrass nucleus v2.5.0 et versions ultérieures utilisent l'`greengrass:ListThingGroupsForCoreDevice` autorisation de désinstaller des composants lorsque vous supprimez un périphérique principal d'un groupe d'objets.
- Les appareils principaux qui exécutent Greengrass nucleus v2.3.0 et versions ultérieures utilisent l'`greengrass:GetDeploymentConfiguration` autorisation de prendre en charge des documents de configuration de déploiement volumineux.

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 Connect politique inclut le * caractère générique après le nom de l'objet principal de l'appareil (par exemple, *core-device-thing-name**). L'appareil principal utilise le même certificat d'appareil pour souscrire plusieurs abonnements simultanés AWS IoT Core, mais l'ID client d'une connexion peut ne pas correspondre exactement au nom de l'appareil principal. Après les 50 premiers abonnements, l'appareil principal l'utilise *core-device-thing-name#number* comme identifiant client, qui *number* augmente pour chaque 50 abonnements supplémentaires. Par exemple, lorsqu'un périphérique principal nommé MyCoreDevice crée 150 abonnements simultanés, il utilise le client suivant IDs :
 - Abonnements 1 à 50 : MyCoreDevice
 - Abonnements 51 à 100 : MyCoreDevice#2
 - Abonnements 101 à 150 : MyCoreDevice#3

Le caractère générique permet au périphérique principal de se connecter lorsqu'il utilise ces clients IDs dotés d'un suffixe.

- 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 AWS IoT Core les composants Greengrass et les appareils clients, 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 politique autorise la publication dans la rubrique suivante pour les données de télémétrie.

```
$aws/things/core-device-thing-name/greengrass/health/json
```

Vous pouvez supprimer cette autorisation pour les appareils principaux sur lesquels vous désactivez la télémétrie. Pour de plus amples informations, veuillez consulter [Collectez les données de télémétrie relatives à l'état du système à partir des principaux appareils AWS IoT Greengrass](#).

- La politique accorde l'autorisation d'assumer un rôle IAM par le biais d'un alias de AWS IoT rôle. Le périphérique principal utilise ce rôle, appelé rôle d'échange de jetons, pour acquérir des AWS informations d'identification qu'il peut utiliser pour authentifier les AWS demandes. Pour de plus amples informations, veuillez consulter [Autoriser les appareils principaux à interagir avec les AWS services](#).

Lorsque vous installez le logiciel AWS IoT Greengrass Core, vous créez et joignez une deuxième AWS IoT politique qui inclut uniquement cette autorisation. Si vous incluez cette autorisation dans la AWS IoT politique principale de votre appareil principal, vous pouvez détacher et supprimer l'autre AWS IoT politique.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Connect"
      ],
      "Resource": "arn:aws:iot:us-east-1:123456789012:client/core-device-thing-name*"
    }
  ]
}
```



```

    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:Receive",
        "iot:Publish"
      ],
      "Resource": [
        "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/core-  
device-thing-name/greengrass/health/json",
        "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/core-  
device-thing-name/greengrassv2/health/json",
        "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/core-  
device-thing-name/jobs/*",
        "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/core-  
device-thing-name/shadow/*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:Subscribe"
      ],
      "Resource": [
        "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/things/core-  
device-thing-name/jobs/*",
        "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/things/core-  
device-thing-name/shadow/*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": "iot:AssumeRoleWithCertificate",
      "Resource": "arn:aws:iot:us-east-1:123456789012:rolealias/token-  
exchange-role-alias-name"
    },
    {
      "Effect": "Allow",
      "Action": [
        "greengrass:GetComponentVersionArtifact",
        "greengrass:ResolveComponentCandidates",
        "greengrass:GetDeploymentConfiguration",
        "greengrass:ListThingGroupsForCoreDevice"
      ]
    }
  ],

```

```
        "Resource": "*"
      }
    ]
  }
}
```

AWS IoT Politique minimale de prise en charge des appareils clients

L'exemple de politique suivant inclut l'ensemble minimal d'actions requises pour prendre en charge l'interaction avec les appareils clients sur un périphérique principal. Pour prendre en charge les appareils clients, un périphérique principal doit disposer des autorisations définies dans cette AWS IoT politique, en plus de la [AWS IoT politique minimale pour le fonctionnement de base](#).

- La politique permet au périphérique principal de mettre à jour ses propres informations de connectivité. Cette autorisation (`greengrass:UpdateConnectivityInfo`) n'est requise que si vous déployez le [composant de détection IP](#) sur le périphérique principal.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Publish"
      ],
      "Resource": [
        "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/core-  
device-thing-name-gci/shadow/get"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:Subscribe"
      ],
      "Resource": [
        "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/things/core-  
device-thing-name-gci/shadow/update/delta",

```

```

        "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/things/core-device-thing-name-gci/shadow/get/accepted"
    ],
    {
        "Effect": "Allow",
        "Action": [
            "iot:Receive"
        ],
        "Resource": [
            "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/core-device-thing-name-gci/shadow/update/delta",
            "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/core-device-thing-name-gci/shadow/get/accepted"
        ]
    },
    {
        "Effect": "Allow",
        "Action": [
            "greengrass:PutCertificateAuthorities",
            "greengrass:VerifyClientDeviceIdentity"
        ],
        "Resource": "*"
    },
    {
        "Effect": "Allow",
        "Action": [
            "greengrass:VerifyClientDeviceIoTCertificateAssociation"
        ],
        "Resource": "arn:aws:iot:us-east-1:123456789012:thing/*"
    },
    {
        "Effect": "Allow",
        "Action": [
            "greengrass:GetConnectivityInfo",
            "greengrass:UpdateConnectivityInfo"
        ],
        "Resource": [
            "arn:aws:iot:us-east-1:123456789012:thing/core-device-thing-name"
        ]
    }
]
}

```

AWS IoT Politique minimale pour les appareils clients

L'exemple de politique suivant inclut l'ensemble minimal d'actions requises pour qu'un appareil client découvre les périphériques principaux auxquels il se connecte et communique via MQTT. La AWS IoT politique de l'appareil client doit inclure l'`greengrass:Discover` action permettant à l'appareil de découvrir les informations de connectivité pour les appareils principaux Greengrass associés. Dans la `Resource` section, spécifiez l'Amazon Resource Name (ARN) de l'appareil client, et non l'ARN du périphérique principal Greengrass.

- La politique permet la communication sur tous les sujets du MQTT. Pour suivre les meilleures pratiques de sécurité, limitez les `iot:Publish`, `iot:Subscribe`, et `iot:Receive` les autorisations à l'ensemble minimal de sujets requis par un appareil client pour votre cas d'utilisation.
- La politique permet à l'objet de découvrir les principaux appareils pour AWS IoT tout. Pour suivre les meilleures pratiques de sécurité, limitez l'`greengrass:Discover` autorisation à l' AWS IoT objet de l'appareil client ou à un caractère générique correspondant à un ensemble d' AWS IoT éléments.

Important

[Les variables de politique des](#) objets (`iot:Connection.Thing.*`) ne sont pas prises en charge dans AWS IoT les politiques relatives aux appareils principaux ou aux opérations du plan de données Greengrass. Vous pouvez plutôt utiliser un caractère générique correspondant à plusieurs appareils portant des noms similaires. Par exemple, vous pouvez spécifier `MyGreengrassDevice*` de correspondre `MyGreengrassDevice1`, `MyGreengrassDevice2`, et ainsi de suite.

- La AWS IoT politique d'un appareil client n'exige généralement pas d'autorisations ou d'`iot:DeleteThingShadow`, `iot:GetThingShadow`, `iot:UpdateThingShadow`, car le périphérique principal de Greengrass gère les opérations de synchronisation parallèle pour les appareils clients. Pour permettre au périphérique principal de gérer les ombres des périphériques clients, vérifiez que la AWS IoT politique du périphérique principal autorise ces actions et que l'ARN de la `Resource` section inclut les périphériques clients.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Connect"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:Publish"
      ],
      "Resource": [
        "arn:aws:iot:us-east-1:123456789012:topic/*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:Subscribe"
      ],
      "Resource": [
        "arn:aws:iot:us-east-1:123456789012:topicfilter/*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:Receive"
      ],
      "Resource": [
        "arn:aws:iot:us-east-1:123456789012:topic/*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "greengrass:Discover"
      ]
    }
  ]
}
```

```
    ],
    "Resource": [
        "arn:aws:iot:us-east-1:123456789012:thing/*"
    ]
}
}
```

Gestion des identités et des accès pour AWS IoT Greengrass

Gestion des identités et des accès AWS (IAM) est un outil Service AWS qui permet à un administrateur de contrôler en toute sécurité l'accès aux AWS ressources. Les administrateurs IAM contrôlent qui peut être authentifié (connecté) et autorisé (autorisé) à utiliser AWS IoT Greengrass les ressources. IAM est un Service AWS outil 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 par AWS IoT Greengrass, consultez [the section called "Comment AWS IoT Greengrass fonctionne avec IAM"](#).

Public ciblé

La façon dont vous utilisez Gestion des identités et des accès AWS (IAM) varie en fonction de votre rôle :

- Utilisateur du service : demandez des autorisations à votre administrateur si vous ne pouvez pas accéder aux fonctionnalités (voir [Résolution des problèmes d'identité et d'accès pour AWS IoT Greengrass](#))
- Administrateur du service : déterminez l'accès des utilisateurs et soumettez les demandes d'autorisation (voir [Comment AWS IoT Greengrass fonctionne avec IAM](#))
- Administrateur IAM : rédigez des politiques pour gérer l'accès (voir [Exemples de politiques basées sur l'identité pour AWS IoT Greengrass](#))

Authentification par des identités

L'authentification est la façon dont vous vous connectez à AWS l'aide de vos informations d'identification. Vous devez être authentifié en tant qu'utilisateur IAM ou en assumant un rôle IAM. Utilisateur racine d'un compte AWS

Vous pouvez vous connecter en tant qu'identité fédérée à l'aide d'informations d'identification provenant d'une source d'identité telle que AWS IAM Identity Center (IAM Identity Center), d'une authentification unique ou d'informations d'identification. Google/Facebook Pour plus d'informations sur la connexion, consultez [Connexion à votre Compte AWS](#) dans le Guide de l'utilisateur Connexion à AWS .

Pour l'accès par programmation, AWS fournit un SDK et une CLI pour signer les demandes de manière cryptographique. Pour plus d'informations, consultez [Signature AWS Version 4 pour les demandes d'API](#) dans le Guide de l'utilisateur IAM.

Compte AWS utilisateur root

Lorsque vous créez un Compte AWS, vous commencez par une seule identité de connexion appelée utilisateur Compte AWS root qui dispose d'un accès complet à toutes Services AWS les ressources. Il est vivement déconseillé d'utiliser l'utilisateur racine pour vos tâches quotidiennes. Pour les tâches qui requièrent des informations d'identification de l'utilisateur racine, consultez [Tâches qui requièrent les informations d'identification de l'utilisateur racine](#) dans le Guide de l'utilisateur IAM.

Utilisateurs et groupes IAM

Un [utilisateur IAM](#) est une identité qui dispose d'autorisations spécifiques pour une seule personne ou application. Nous vous recommandons d'utiliser ces informations d'identification temporaires au lieu des utilisateurs IAM avec des informations d'identification à long terme. Pour plus d'informations, voir [Exiger des utilisateurs humains qu'ils utilisent la fédération avec un fournisseur d'identité pour accéder à AWS l'aide d'informations d'identification temporaires](#) dans le guide de l'utilisateur IAM.

[Les groupes IAM](#) spécifient une collection d'utilisateurs IAM et permettent de gérer plus facilement les autorisations pour de grands ensembles d'utilisateurs. Pour plus d'informations, consultez [Cas d'utilisation pour les utilisateurs IAM](#) dans le Guide de l'utilisateur IAM.

Rôles IAM

Un [rôle IAM](#) est une identité dotée d'autorisations spécifiques qui fournit des informations d'identification temporaires. Vous pouvez assumer un rôle en [passant d'un rôle d'utilisateur à un rôle](#)

[IAM \(console\)](#) ou en appelant une opération d' AWS API AWS CLI ou d'API. Pour plus d'informations, consultez [Méthodes pour endosser un rôle](#) dans le Guide de l'utilisateur IAM.

Les rôles IAM sont utiles pour l'accès des utilisateurs fédérés, les autorisations temporaires des utilisateurs IAM, les accès intercompte, les accès entre services et les applications exécutées sur Amazon EC2. Pour plus d'informations, consultez [Accès intercompte aux ressources dans IAM](#) dans le Guide de l'utilisateur IAM.

Gestion de l'accès à l'aide de politiques

Vous contrôlez l'accès en AWS créant des politiques et en les associant à AWS des identités ou à des ressources. Une politique définit les autorisations lorsqu'elles sont associées à une identité ou à une ressource. AWS évalue ces politiques lorsqu'un directeur fait une demande. La plupart des politiques sont stockées AWS sous forme de documents JSON. Pour plus d'informations les documents de politique JSON, consultez [Vue d'ensemble des politiques JSON](#) dans le Guide de l'utilisateur IAM.

À l'aide de politiques, les administrateurs précisent qui a accès à quoi en définissant 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. Un administrateur IAM crée des politiques IAM et les ajoute aux rôles, que les utilisateurs peuvent ensuite assumer. Les politiques IAM définissent les autorisations quelle que soit la méthode que vous utilisez pour exécuter l'opération.

Politiques basées sur l'identité

Les stratégies basées sur l'identité sont des documents de stratégie d'autorisations JSON que vous attachez à une identité (utilisateur, groupe ou rôle). Ces politiques contrôlent les actions que peuvent exécuter ces identités, sur quelles ressources et dans quelles conditions. Pour découvrir comment créer une politique basée sur l'identité, consultez [Définition d'autorisations IAM personnalisées avec des politiques gérées par le client](#) dans le Guide de l'utilisateur IAM.

Les politiques basées sur l'identité peuvent être des politiques intégrées (intégrées directement dans une seule identité) ou des politiques gérées (politiques autonomes associées à plusieurs identités). Pour découvrir comment choisir entre des politiques gérées et 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. Les exemples incluent les politiques de confiance de rôle IAM et les stratégies 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. Vous devez [spécifier un principal](#) dans une politique basée sur les ressources.

Les politiques basées sur les ressources sont des politiques en ligne situées dans ce service. Vous ne pouvez pas utiliser les politiques AWS gérées par IAM dans une stratégie basée sur les ressources.

Listes de contrôle d'accès (ACLs)

Les listes de contrôle d'accès (ACLs) contrôlent les principaux (membres du compte, utilisateurs ou rôles) autorisés à accéder à une ressource. ACLs sont similaires aux politiques basées sur les ressources, bien qu'elles n'utilisent pas le format de document de politique JSON.

Amazon S3 et AWS WAF Amazon VPC sont des exemples de services compatibles. ACLs Pour en savoir plus ACLs, consultez la [présentation de la liste de contrôle d'accès \(ACL\)](#) dans le guide du développeur Amazon Simple Storage Service.

Autres types de politique

AWS prend en charge des types de politiques supplémentaires qui peuvent définir les autorisations maximales accordées par les types de politiques les plus courants :

- Limites d'autorisations : une limite des autorisations définit le nombre maximum d'autorisations qu'une politique basée sur l'identité peut accorder à une entité IAM. Pour plus d'informations, consultez [Limites d'autorisations pour des entités IAM](#) dans le Guide de l'utilisateur IAM.
- Politiques de contrôle des services (SCPs) — Spécifiez les autorisations maximales pour une organisation ou une unité organisationnelle dans AWS Organizations. Pour plus d'informations, consultez [Politiques de contrôle de service](#) dans le Guide de l'utilisateur AWS Organizations .
- Politiques de contrôle des ressources (RCPs) : définissez le maximum d'autorisations disponibles pour les ressources de vos comptes. Pour plus d'informations, voir [Politiques de contrôle des ressources \(RCPs\)](#) dans le guide de AWS Organizations l'utilisateur.
- Politiques de session : politiques avancées que vous passez en tant que paramètre lorsque vous créez par programmation une session temporaire pour un rôle ou un utilisateur fédéré. Pour plus d'informations, consultez [Politiques de session](#) dans le Guide de l'utilisateur IAM.

Plusieurs types de politique

Lorsque plusieurs types de politiques s'appliquent à la requête, les autorisations en résultant sont plus compliquées à comprendre. Pour savoir comment AWS déterminer s'il faut autoriser une demande lorsque plusieurs types de politiques sont impliqués, consultez la section [Logique d'évaluation des politiques](#) dans le guide de l'utilisateur IAM.

Consultez aussi

- [the section called “Comment AWS IoT Greengrass fonctionne avec IAM”](#)
- [the section called “Exemples de politiques basées sur l'identité”](#)
- [the section called “Résolution des problèmes d'identité et d'accès”](#)

Comment AWS IoT Greengrass fonctionne avec IAM

Avant d'utiliser IAM pour gérer l'accès AWS IoT Greengrass, vous devez comprendre les fonctionnalités IAM que vous pouvez utiliser. 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 (ACLs)	Non
Autorisation basée sur les balises	Oui
Informations d'identification temporaires	Oui
Rôles liés à un service	Non
Rôles de service	Oui

Pour obtenir une vue d'ensemble de la façon dont les autres AWS services fonctionnent avec IAM, consultez la section [AWS Services compatibles avec IAM](#) dans le Guide de l'utilisateur d'IAM.

Stratégies 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 AWS JSON 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. Intégration d'actions dans une politique afin d'accorder l'autorisation d'exécuter les opérations associées.

Actions de stratégie pour AWS IoT Greengrass utiliser le greengrass : préfixe avant l'action. Par exemple, pour permettre à quelqu'un d'utiliser l'opération `ListCoreDevicesAPI` pour répertorier les principaux appareils dans sa politique Compte AWS, vous devez inclure `greengrass:ListCoreDevicesaction` dans sa politique. Les déclarations de politique doivent inclure un `NotAction` élément `Action` ou. AWS IoT Greengrass définit son propre ensemble d'actions décrivant 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 appliquer le principe du moindre privilège et à accorder des autorisations de portée limitée dans une stratégie. Pour de plus amples informations, veuillez consulter [the section called “Accorder le moins d'autorisations possibles”](#).

Pour la liste complète des AWS IoT Greengrass actions, voir [Actions définies par AWS IoT Greengrass](#) dans le guide de l'utilisateur IAM.

Ressources

Les administrateurs peuvent utiliser les politiques AWS JSON 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 auxquels l'action s'applique. Il est recommandé de définir une ressource à l'aide de son [Amazon Resource Name \(ARN\)](#). Pour les actions qui ne sont pas compatibles avec les autorisations de niveau ressource, utilisez un caractère générique (*) afin d'indiquer que l'instruction s'applique à toutes les ressources.

```
"Resource": "*"
```

Le tableau suivant contient la AWS IoT Greengrass ressource ARNs qui peut être utilisée dans l'élément `Resource` d'une déclaration de politique. Pour un mappage des autorisations prises en charge au niveau des ressources pour les AWS IoT Greengrass actions, voir [Actions définies par AWS IoT Greengrass](#) dans le guide de l'utilisateur IAM.

Certaines AWS IoT Greengrass actions (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 ressources ARNs dans une instruction, listez-les entre crochets ([]) et séparez-les par des virgules, comme suit :

```
"Resource": [  
  "resource-arn1",  
  "resource-arn2",  
  "resource-arn3"  
]
```

Pour plus d'informations sur les formats ARN, consultez [Amazon Resource Names \(ARNs\) et les espaces de noms de AWS service](#) dans le Référence générale d'Amazon Web Services.

Clés de condition

Les administrateurs peuvent utiliser les politiques AWS JSON 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 indique à quel moment les instructions sont exécutées en fonction de critères définis. 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. Pour voir toutes les clés de condition AWS globales, voir les clés de [contexte de condition AWS globales](#) dans le guide de l'utilisateur IAM.

Exemples

Pour consulter des exemples de politiques AWS IoT Greengrass basées sur l'identité, consultez [the section called "Exemples de politiques basées sur l'identité"](#)

Politiques basées sur les ressources pour AWS IoT Greengrass

AWS IoT Greengrass ne prend pas en charge les [politiques basées sur les ressources](#).

Listes de contrôle d'accès (ACLs)

AWS IoT Greengrass ne prend pas en charge [ACLs](#).

Autorisation basée sur les AWS IoT Greengrass tags

Vous pouvez associer des balises aux AWS IoT Greengrass ressources prises 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 de plus amples informations, veuillez consulter [Balisage de vos ressources](#).

Rôles IAM pour AWS IoT Greengrass

Un [rôle IAM](#) est une entité au sein de votre Compte AWS qui dispose d'autorisations spécifiques.

Utilisation d'informations d'identification temporaires avec AWS IoT Greengrass

Les informations d'identification temporaires sont utilisées pour se connecter à la fédération, assumer un rôle IAM ou assumer un rôle entre comptes. Vous obtenez des informations d'identification de sécurité temporaires en appelant des opérations d' AWS STS API telles que [AssumeRole](#) ou [GetFederationToken](#).

Sur le noyau de Greengrass, des informations d'identification temporaires pour le [rôle d'appareil](#) sont mises à la disposition des composants de Greengrass. Si vos composants 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](#).

Rôles 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 rôles de 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 les appareils principaux utilisent un rôle de service pour permettre aux composants Greengrass et aux fonctions Lambda d'accéder à certaines de vos AWS ressources en votre nom. Pour de plus amples informations, veuillez consulter [the section called "Autoriser les appareils principaux à interagir avec les AWS services"](#).

AWS IoT Greengrass utilise un rôle de service pour accéder à certaines de vos AWS ressources en votre nom. Pour de plus amples informations, veuillez consulter [Rôle de service Greengrass](#).

Exemples de politiques 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 effectuer de tâches à l'aide de l' AWS API AWS

Management Console AWS CLI, ou. 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 politiques basées sur l'identité déterminent si quelqu'un peut créer, accéder ou supprimer AWS IoT Greengrass des ressources 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 :

- Commencez AWS par les politiques gérées et passez aux autorisations du moindre privilège : pour commencer à accorder des autorisations à vos utilisateurs et à vos charges de travail, utilisez les politiques AWS gérées qui accordent des autorisations pour de nombreux cas d'utilisation courants. Ils sont disponibles dans votre Compte AWS. Nous vous recommandons de réduire davantage les autorisations en définissant des politiques gérées par le AWS client spécifiques à vos cas d'utilisation. Pour plus d'informations, consultez [politiques gérées par AWS](#) ou [politiques gérées par AWS pour les activités professionnelles](#) dans le Guide de l'utilisateur IAM.
- Accordez 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 d'IAM pour appliquer des autorisations, consultez [politiques et autorisations dans IAM](#) dans le Guide de l'utilisateur IAM.
- Utilisez 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 par le biais d'un service spécifique Service AWS, tel que CloudFormation. Pour plus d'informations, consultez [Conditions pour éléments de politique JSON IAM](#) dans le Guide de l'utilisateur IAM.
- Utilisez l'Analyseur d'accès IAM pour valider vos politiques IAM afin de garantir des autorisations sécurisées et fonctionnelles : l'Analyseur d'accès IAM 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 plus d'informations, consultez [Validation de politiques avec IAM Access Analyzer](#) dans le Guide de l'utilisateur IAM.

- Exiger l'authentification multifactorielle (MFA) : si vous avez un scénario qui nécessite des utilisateurs IAM ou un utilisateur root, activez l'authentification MFA pour une sécurité accrue. Compte AWS Pour exiger la MFA lorsque des opérations d'API sont appelées, ajoutez des conditions MFA à vos politiques. Pour plus d'informations, consultez [Sécurisation de l'accès aux API avec MFA](#) dans le Guide de l'utilisateur IAM.

Pour plus d'informations sur les bonnes pratiques dans IAM, consultez [Bonnes pratiques de sécurité dans IAM](#) dans le Guide de l'utilisateur IAM.

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, consultez [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 permettant d'effectuer cette action sur la console ou par programmation à l'aide de l'API AWS CLI or 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": "*"
}
]
}

```

Autoriser les appareils principaux à interagir avec les AWS services

AWS IoT Greengrass les appareils principaux utilisent le fournisseur AWS IoT Core d'informations d'identification pour autoriser les appels aux AWS services. Le fournisseur AWS IoT Core d'informations d'identification permet aux appareils d'utiliser leurs certificats X.509 comme identité d'appareil unique pour authentifier les demandes. AWS Il n'est donc plus nécessaire de stocker un identifiant de clé d' AWS accès et une clé d'accès secrète sur vos AWS IoT Greengrass principaux appareils. Pour plus d'informations, consultez la section [Autorisation des appels directs vers AWS des services](#) dans le Guide du AWS IoT Core développeur.

Lorsque vous exécutez le logiciel AWS IoT Greengrass Core, vous pouvez choisir de fournir les AWS ressources dont le périphérique principal a besoin. Cela inclut le rôle Gestion des identités et des accès AWS (IAM) que votre appareil principal assume par le biais du fournisseur AWS IoT Core d'informations d'identification. Utilisez l'--provision true argument pour configurer un rôle et des politiques permettant au périphérique principal d'obtenir des AWS informations d'identification temporaires. Cet argument configure également un alias de AWS IoT rôle qui pointe vers ce rôle IAM. Vous pouvez spécifier le nom du rôle IAM et l'alias de AWS IoT rôle à utiliser. Si vous spécifiez --

provision true sans ces autres paramètres de nom, le périphérique principal de Greengrass crée et utilise les ressources par défaut suivantes :

- Rôle IAM : GreengrassV2TokenExchangeRole

Ce rôle possède une politique nommée GreengrassV2TokenExchangeRoleAccess et une relation de confiance qui permet `credentials.iot.amazonaws.com` d'assumer le rôle. La politique inclut les autorisations minimales pour le périphérique principal.

 Important

Cette politique n'inclut pas l'accès aux fichiers dans les compartiments S3. Vous devez ajouter des autorisations au rôle pour permettre aux appareils principaux de récupérer les artefacts des composants à partir des compartiments S3. Pour de plus amples informations, veuillez consulter [Autoriser l'accès aux compartiments S3 pour les artefacts des composants](#).

- AWS IoT alias de rôle : GreengrassV2TokenExchangeRoleAlias

Cet alias de rôle fait référence au rôle IAM.

Pour de plus amples informations, veuillez consulter [Étape 3 : Installation du logiciel AWS IoT Greengrass principal](#).

Vous pouvez également définir l'alias de rôle pour un appareil principal existant. Pour ce faire, configurez le paramètre de `iotRoleAlias` configuration du composant [Greengrass noyau](#).

Vous pouvez obtenir des AWS informations d'identification temporaires pour ce rôle IAM afin d'effectuer AWS des opérations dans vos composants personnalisés. Pour de plus amples informations, veuillez consulter [Interagissez avec les AWS services](#).

Rubriques

- [Autorisations de rôle de service pour les appareils principaux](#)
- [Autoriser l'accès aux compartiments S3 pour les artefacts des composants](#)

Autorisations de rôle de service pour les appareils principaux

Le rôle permet au service suivant d'assumer le rôle :

- `credentials.iot.amazonaws.com`

Si vous utilisez le logiciel AWS IoT Greengrass Core pour créer ce rôle, il utilise la politique d'autorisation suivante pour autoriser les appareils principaux à se connecter et à envoyer des journaux à AWS. Le nom de la politique est par défaut le nom du rôle IAM se terminant par `Access`. Par exemple, si vous utilisez le nom de rôle IAM par défaut, le nom de cette politique est `GreengrassV2TokenExchangeRoleAccess`.

Greengrass nucleus v2.5.0 and later

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams",
        "s3:GetBucketLocation"
      ],
      "Resource": "*"
    }
  ]
}
```

v2.4.x

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
```

```
        "iot:DescribeCertificate",
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams",
        "s3:GetBucketLocation"
    ],
    "Resource": "*"
}
]
```

Earlier than v2.4.0

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:DescribeCertificate",
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams",
        "iot:Connect",
        "iot:Publish",
        "iot:Subscribe",
        "iot:Receive",
        "s3:GetBucketLocation"
      ],
      "Resource": "*"
    }
  ]
}
```

Autoriser l'accès aux compartiments S3 pour les artefacts des composants

Le rôle de périphérique principal par défaut ne permet pas aux appareils principaux d'accéder aux compartiments S3. Pour déployer des composants contenant des artefacts dans des compartiments S3, vous devez ajouter l'`s3:GetObject` autorisation permettant aux appareils principaux de télécharger des artefacts de composants. Vous pouvez ajouter une nouvelle politique au rôle principal de l'appareil pour accorder cette autorisation.

Pour ajouter une politique qui autorise l'accès aux artefacts des composants dans Amazon S3

1. Créez un fichier appelé `component-artifact-policy.json` et copiez-y le code JSON suivant. Cette politique permet d'accéder à tous les fichiers d'un compartiment S3. Remplacez `amzn-s3-demo-bucket` par le nom du compartiment S3 pour autoriser l'accès au périphérique principal.

JSON

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

2. Exécutez la commande suivante pour créer la politique à partir du document de stratégie dans `component-artifact-policy.json`.

Linux or Unix

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

Windows Command Prompt (CMD)

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

PowerShell

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

Copiez le nom Amazon Resource Name (ARN) de la politique à partir des métadonnées de la politique dans la sortie. Vous utilisez cet ARN pour associer cette politique au rôle principal de l'appareil à l'étape suivante.

3. Exécutez la commande suivante pour associer la politique au rôle principal de l'appareil. *GreengrassV2TokenExchangeRole* Remplacez-le par le nom du rôle que vous avez spécifié lors de l'exécution du logiciel AWS IoT Greengrass Core. Remplacez ensuite l'ARN de la politique par l'ARN de l'étape précédente.

Linux or Unix

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

Windows Command Prompt (CMD)

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

PowerShell

```
aws iam attach-role-policy `  
  --role-name GreengrassV2TokenExchangeRole `
```

```
--policy-arn
arn:aws:iam::123456789012:policy/MyGreengrassV2ComponentArtifactPolicy
```

Si la commande n'a aucune sortie, elle a réussi et votre appareil principal peut accéder aux artefacts que vous téléchargez dans ce compartiment S3.

Politique IAM minimale permettant au programme d'installation de provisionner les ressources

Lorsque vous installez le logiciel AWS IoT Greengrass Core, vous pouvez fournir les AWS ressources requises, telles qu'un AWS IoT objet ou un rôle IAM pour votre appareil. Vous pouvez également déployer des outils de développement locaux sur l'appareil. Le programme d'installation a besoin AWS d'informations d'identification pour pouvoir effectuer ces actions dans votre Compte AWS. Pour de plus amples informations, veuillez consulter [Installation du logiciel AWS IoT Greengrass de base](#).

L'exemple de politique suivant inclut l'ensemble minimal d'actions dont le programme d'installation a besoin pour provisionner ces ressources. Ces autorisations sont requises si vous spécifiez l'`--provision` argument du programme d'installation. `account-id` Remplacez-le par votre Compte AWS ID et remplacez-le `GreengrassV2TokenExchangeRole` par le nom du rôle d'échange de jetons que vous spécifiez avec l'[argument du --tes-role-name programme d'installation](#).

Note

La déclaration `DeployDevTools` de politique n'est requise que si vous spécifiez l'`--deploy-dev-tools` argument du programme d'installation.

Greengrass nucleus v2.5.0 and later

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
```

```

        "Sid": "CreateTokenExchangeRole",
        "Effect": "Allow",
        "Action": [
            "iam:AttachRolePolicy",
            "iam:CreatePolicy",
            "iam:CreateRole",
            "iam:GetPolicy",
            "iam:GetRole",
            "iam:PassRole"
        ],
        "Resource": [
            "arn:aws:iam::123456789012:role/GreengrassV2TokenExchangeRole",
            "arn:aws:iam::123456789012:policy/GreengrassV2TokenExchangeRoleAccess",
            "arn:aws:iam::aws:policy/GreengrassV2TokenExchangeRoleAccess"
        ]
    },
    {
        "Sid": "CreateIoTResources",
        "Effect": "Allow",
        "Action": [
            "iot:AddThingToThingGroup",
            "iot:AttachPolicy",
            "iot:AttachThingPrincipal",
            "iot:CreateKeysAndCertificate",
            "iot:CreatePolicy",
            "iot:CreateRoleAlias",
            "iot:CreateThing",
            "iot:CreateThingGroup",
            "iot:DescribeEndpoint",
            "iot:DescribeRoleAlias",
            "iot:DescribeThingGroup",
            "iot:GetPolicy"
        ],
        "Resource": "*"
    },
    {
        "Sid": "DeployDevTools",
        "Effect": "Allow",
        "Action": [
            "greengrass:CreateDeployment",
            "iot:CancelJob",
            "iot:CreateJob",

```



```

        "iot:DeleteThingShadow",
        "iot:DescribeJob",
        "iot:DescribeThing",
        "iot:DescribeThingGroup",
        "iot:GetThingShadow",
        "iot:UpdateJob",
        "iot:UpdateThingShadow"
    ],
    "Resource": "*"
}
]
}

```

Earlier than v2.5.0

JSON

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "CreateTokenExchangeRole",
      "Effect": "Allow",
      "Action": [
        "iam:AttachRolePolicy",
        "iam:CreatePolicy",
        "iam:CreateRole",
        "iam:GetPolicy",
        "iam:GetRole",
        "iam:PassRole"
      ],
      "Resource": [
        "arn:aws:iam::123456789012:role/GreengrassV2TokenExchangeRole",
        "arn:aws:iam::123456789012:policy/GreengrassV2TokenExchangeRoleAccess",
        "arn:aws:iam::aws:policy/GreengrassV2TokenExchangeRoleAccess"
      ]
    },
    {
      "Sid": "CreateIoTResources",
      "Effect": "Allow",
      "Action": [
        "iot:AddThingToThingGroup",

```

```

        "iot:AttachPolicy",
        "iot:AttachThingPrincipal",
        "iot:CreateKeysAndCertificate",
        "iot:CreatePolicy",
        "iot:CreateRoleAlias",
        "iot:CreateThing",
        "iot:CreateThingGroup",
        "iot:DescribeEndpoint",
        "iot:DescribeRoleAlias",
        "iot:DescribeThingGroup",
        "iot:GetPolicy"
    ],
    "Resource": "*"
  },
  {
    "Sid": "DeployDevTools",
    "Effect": "Allow",
    "Action": [
      "greengrass:CreateDeployment",
      "iot:CancelJob",
      "iot:CreateJob",
      "iot>DeleteThingShadow",
      "iot:DescribeJob",
      "iot:DescribeThing",
      "iot:DescribeThingGroup",
      "iot:GetThingShadow",
      "iot:UpdateJob",
      "iot:UpdateThingShadow"
    ],
    "Resource": "*"
  }
]
}

```

Rôle de service Greengrass

Le rôle de service Greengrass est un rôle de service Gestion des identités et des accès AWS (IAM) qui autorise l'accès AWS IoT Greengrass aux ressources des AWS services en votre nom. Ce rôle permet de vérifier l'identité AWS IoT Greengrass des appareils clients et de gérer les informations de connectivité de base des appareils.

 Note

AWS IoT Greengrass V1 utilise également ce rôle pour effectuer des tâches essentielles. Pour plus d'informations, voir le [rôle de service Greengrass](#) dans le Guide du AWS IoT Greengrass V1 développeur.

Pour autoriser AWS IoT Greengrass l'accès à vos ressources, le rôle de service Greengrass doit être associé à votre rôle Compte AWS et spécifié en AWS IoT Greengrass tant qu'entité de confiance. Le rôle doit inclure la politique [AWSGreengrassResourceAccessRolePolicy](#) gérée ou une politique personnalisée qui définit des autorisations équivalentes pour les AWS IoT Greengrass fonctionnalités que vous utilisez. AWS applique cette politique, qui définit l'ensemble des autorisations AWS IoT Greengrass utilisées pour accéder à vos AWS ressources. Pour de plus amples informations, veuillez consulter [AWS politique gérée : AWSGreengrass ResourceAccessRolePolicy](#).

Vous pouvez réutiliser le même rôle de service Greengrass partout Régions AWS, mais vous devez l'associer à votre compte partout Région AWS où vous l'utilisez. AWS IoT Greengrass Si le rôle de service n'est pas configuré actuellement Région AWS, les appareils principaux ne vérifient pas les appareils clients et ne mettent pas à jour les informations de connectivité.

Les sections suivantes décrivent comment créer et gérer le rôle de service Greengrass avec le AWS Management Console ou. AWS CLI

Rubriques

- [Gérer le rôle de service Greengrass \(console\)](#)
- [Gérer le rôle de service Greengrass \(CLI\)](#)
- [Consultez aussi](#)

 Note

Outre le rôle de service qui autorise l'accès au niveau de service, vous attribuez un rôle d'échange de jetons aux appareils principaux de Greengrass. Le rôle d'échange de jetons est un rôle IAM distinct qui contrôle la manière dont les composants Greengrass et les fonctions Lambda du périphérique principal peuvent accéder aux services. AWS Pour de plus amples informations, veuillez consulter [Autoriser les appareils principaux à interagir avec les AWS services](#).

Gérer le rôle de service Greengrass (console)

La AWS IoT console facilite la gestion de votre rôle dans le service Greengrass. Par exemple, lorsque vous configurez la découverte des appareils clients pour un appareil principal, la console vérifie si vous êtes Compte AWS actuellement attaché à un rôle de service Greengrass. Région AWS Si ce n'est pas le cas, la console peut créer et configurer un rôle de service pour vous. Pour de plus amples informations, veuillez consulter [the section called “Créer le rôle de service Greengrass”](#).

Vous pouvez utiliser la console pour les tâches de gestion des rôles suivantes :

Rubriques

- [Rechercher votre rôle de service Greengrass \(console\)](#)
- [Créer le rôle de service Greengrass \(console\)](#)
- [Modifier le rôle de service Greengrass \(console\)](#)
- [Détacher le rôle de service Greengrass \(console\)](#)

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)

Suivez les étapes ci-dessous pour trouver le rôle de service AWS IoT Greengrass utilisé dans le courant Région AWS.

1. Accédez à la [console AWS IoT](#).
2. Dans le panneau de navigation, sélectionnez Settings (Paramètres).
3. 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 aucun rôle de service n'apparaît, la console peut en créer ou en configurer un pour vous. Pour de plus amples informations, veuillez consulter [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 vous créez ce rôle à l'aide du [script de configuration de l'AWS IoT Greengrass V1 appareil](#), le nom du rôle est `GreengrassServiceRole_`*random-string*.

Lorsque vous configurez la découverte des appareils clients pour un appareil principal, la console vérifie si un rôle de service Greengrass est associé à votre rôle Compte AWS dans le service actuel. Région AWS Si ce n'est pas le cas, la console vous invite AWS IoT Greengrass à autoriser les AWS services à lire et à écrire en votre nom.

Si vous accordez l'autorisation, la console vérifie si un rôle nommé `Greengrass_ServiceRole` existe dans votre Compte AWS.

- Si le rôle existe, la console attache le rôle de service à votre rôle Compte AWS dans le courant Région AWS.
- Si le rôle n'existe pas, la console crée un rôle de service Greengrass par défaut et l'attache à votre rôle Compte AWS dans le service actuel. Région AWS

Note

Si vous souhaitez créer un rôle de service avec des politiques de rôle personnalisées, utilisez la console IAM pour créer ou modifier le rôle. Pour plus d'informations, voir [Création d'un rôle pour déléguer des autorisations à un AWS service](#) ou [Modification d'un rôle](#) dans le

Guide de l'utilisateur IAM. 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 clés `aws:SourceArn` contextuelles et les clés de contexte de condition `aws:SourceAccount` globale dans votre politique de confiance afin d'éviter le problème de sécurité secondaire confus. Les clés contextuelles de condition limitent l'accès pour autoriser uniquement les demandes provenant du compte spécifié et de l'espace de travail Greengrass. Pour de plus amples informations sur le problème de l'adjoint confus, veuillez consulter [Prévention du problème de l'adjoint confus entre services](#).

Si vous créez un rôle de service, retournez à la AWS IoT console et associez le rôle à votre Compte AWS. Vous pouvez le faire sous le rôle de service Greengrass sur la page Paramètres.

Modifier le rôle de service Greengrass (console)

Utilisez la procédure suivante pour choisir un autre rôle de service Greengrass à associer à votre rôle Compte AWS dans le rôle Région AWS actuellement sélectionné dans la console.

1. Accédez à la [console AWS IoT](#).
2. Dans le panneau de navigation, sélectionnez Settings (Paramètres).
3. Sous le rôle de service Greengrass, sélectionnez Modifier le rôle.

La boîte de dialogue Mettre à jour le rôle de service Greengrass s'ouvre et indique les rôles IAM Compte AWS que vous définissez AWS IoT Greengrass comme une entité de confiance.

4. Choisissez le rôle de service Greengrass à associer.
5. Choisissez Attacher un rôle.

Détacher le rôle de service Greengrass (console)

Suivez la procédure ci-dessous pour dissocier le rôle de service Greengrass de AWS votre compte à l'heure actuelle. Région AWS Cela révoque les autorisations AWS IoT Greengrass d'accès aux AWS services actuels Région AWS.

Important

Le détachement du rôle de service peut interrompre les opérations actives.

1. Accédez à la [console AWS IoT](#).
2. Dans le panneau de navigation, sélectionnez Settings (Paramètres).
3. Sous le rôle de service Greengrass, choisissez Detach role.
4. 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 vous AWS IoT Greengrass permettre d'accéder à vos ressources. Pour trouver tous les rôles qui permettent AWS IoT Greengrass d'assumer des autorisations en votre nom, dans la console IAM, sur la page Rôles, recherchez les rôles incluant AWS service : greengrass dans la colonne Entités fiables.

Gérer le rôle de service Greengrass (CLI)

Dans les procédures suivantes, nous partons du principe que le AWS Command Line Interface est installé et configuré pour utiliser votre Compte AWS. Pour plus d'informations, consultez les [sections Installation, mise à jour et désinstallation du AWS CLI](#) et [Configuration du AWS CLI dans le](#) guide de l'AWS Command Line Interface utilisateur.

Vous pouvez utiliser le AWS CLI pour les tâches de gestion des rôles suivantes :

Rubriques

- [Obtenir le rôle de service Greengrass \(interface de ligne de commande\)](#)
- [Créer le rôle de service Greengrass \(interface de ligne de commande\)](#)
- [Supprimer le rôle de service Greengrass \(interface de ligne de commande\)](#)

Obtenir le rôle de service Greengrass (interface de ligne de commande)

Utilisez la procédure suivante pour savoir si un rôle de service Greengrass vous est associé Compte AWS dans un. Région AWS

- Obtenez le rôle de service. *region* Remplacez-le par votre Région AWS (par exemple, us-west-2).

```
aws greengrassv2 get-service-role-for-account --region region
```

Si un rôle de service Greengrass est déjà associé à votre compte, la demande renvoie les métadonnées de rôle suivantes.

```
{
  "associatedAt": "timestamp",
  "roleArn": "arn:aws:iam::account-id:role/path/role-name"
}
```

Si la demande ne renvoie pas les métadonnées du rôle, vous devez créer le rôle de service (s'il n'existe pas) et l'associer à votre compte dans le Région AWS.

Créer le rôle de service Greengrass (interface de ligne de commande)

Suivez les étapes ci-dessous pour créer un rôle et l'associer à votre Compte AWS.

Pour créer le rôle de service à l'aide d'IAM

1. Créez un rôle avec une politique de confiance qui permet 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 clés `aws:SourceArn` contextuelles et les clés de contexte de condition `aws:SourceAccount` globale dans votre politique de confiance afin d'éviter le problème de sécurité secondaire confus. Les clés contextuelles de condition limitent l'accès pour autoriser uniquement les demandes provenant du compte spécifié et de l'espace de travail Greengrass. Pour de plus amples informations sur le problème de l'adjoint confus, veuillez consulter [Prévention du problème de l'adjoint confus entre services](#).

Linux or Unix

```
aws iam create-role --role-name Greengrass_ServiceRole --assume-role-policy-
document '{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
```



```

        "Service": "greengrass.amazonaws.com"
    },
    "Action": "sts:AssumeRole",
    "Condition": {
        "ArnLike": {
            "aws:SourceArn": "arn:aws:greengrass:region:account-id:*"
        },
        "StringEquals": {
            "aws:SourceAccount": "account-id"
        }
    }
}
]
}'

```

Windows Command Prompt (CMD)

```

aws iam create-role --role-name Greengrass_ServiceRole --assume-role-policy-document "{\"Version\":\"2012-10-17\", \"Statement\": [{\"Effect\": \"Allow\", \"Principal\": {\"Service\": \"greengrass.amazonaws.com\"}, \"Action\": \"sts:AssumeRole\", \"Condition\": {\"ArnLike\": {\"aws:SourceArn\": \"arn:aws:greengrass:region:account-id:*\"}, \"StringEquals\": {\"aws:SourceAccount\": \"account-id\"}}}]}"

```

PowerShell

```

aws iam create-role --role-name Greengrass_ServiceRole --assume-role-policy-document '{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "greengrass.amazonaws.com"
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "ArnLike": {
                    "aws:SourceArn": "arn:aws:greengrass:region:account-id:*"
                },
                "StringEquals": {
                    "aws:SourceAccount": "account-id"
                }
            }
        }
    ]
}'

```

```
}
}
]
}'
```

2. 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 votre Compte AWS

- Associez le rôle à votre compte. `role-arn` Remplacez-le par l'ARN du rôle `region` de service et par votre Région AWS (par exemple, `us-west-2`).

```
aws greengrassv2 associate-service-role-to-account --role-arn role-arn --
region region
```

En cas de succès, la demande renvoie la réponse suivante.

```
{
  "associatedAt": "timestamp"
}
```

Supprimer le rôle de service Greengrass (interface de ligne de commande)

Suivez les étapes ci-dessous pour dissocier le rôle de service Greengrass de votre. Compte AWS

- Dissociez le rôle de service de votre compte. `region` Remplacez-le par votre Région AWS (par exemple, `us-west-2`).

```
aws greengrassv2 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 aucun d'entre eux Région 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.

Consultez aussi

- [Création d'un rôle pour déléguer des autorisations à un AWS service](#) dans le guide de l'utilisateur IAM
- [Modification d'un rôle](#) dans le guide de l'utilisateur IAM
- [Suppression de rôles ou de profils d'instance](#) dans le guide de l'utilisateur IAM
- AWS IoT Greengrass commandes dans la référence des AWS CLI commandes
 - [associate-service-role-to-compte](#)
 - [disassociate-service-role-from-compte](#)
 - [get-service-role-for-compte](#)
- Commandes IAM dans la référence des AWS CLI commandes
 - [attach-role-policy](#)
 - [create-role](#)
 - [delete-role](#)
 - [delete-role-policy](#)

AWS politiques gérées pour AWS IoT Greengrass

Une politique AWS gérée est une politique autonome créée et administrée par AWS. AWS les politiques gérées sont conçues pour fournir des autorisations pour de nombreux cas d'utilisation courants afin que vous puissiez commencer à attribuer des autorisations aux utilisateurs, aux groupes et aux rôles.

N'oubliez pas que les politiques AWS gérées peuvent ne pas accorder d'autorisations de moindre privilège pour vos cas d'utilisation spécifiques, car elles sont accessibles à tous les AWS clients. Nous vous recommandons de réduire encore les autorisations en définissant des [politiques gérées par le client](#) qui sont propres à vos cas d'utilisation.

Vous ne pouvez pas modifier les autorisations définies dans les politiques AWS gérées. Si les autorisations définies dans une politique AWS gérée sont AWS mises à jour, la mise à jour affecte toutes les identités principales (utilisateurs, groupes et rôles) auxquelles la politique est attachée. AWS est le plus susceptible de mettre à jour une politique AWS gérée lorsqu'une nouvelle politique Service AWS est lancée ou lorsque de nouvelles opérations d'API sont disponibles pour les services existants.

Pour plus d'informations, consultez [Politiques gérées par AWS](#) dans le Guide de l'utilisateur IAM.

Rubriques

- [AWS politique gérée : AWSGreengrass FullAccess](#)
- [AWS politique gérée : AWSGreengrass ReadOnlyAccess](#)
- [AWS politique gérée : AWSGreengrass ResourceAccessRolePolicy](#)
- [AWS IoT Greengrass mises à jour des politiques AWS gérées](#)

AWS politique gérée : AWSGreengrass FullAccess

Vous pouvez associer la politique AWSGreengrassFullAccess à vos identités IAM.

Cette politique accorde des autorisations administratives qui permettent au principal d'avoir un accès complet à toutes les AWS IoT Greengrass actions.

Détails de l'autorisation

Cette politique inclut les autorisations suivantes :

- greengrass— Permet aux principaux un accès complet à toutes les AWS IoT Greengrass actions.

JSON

```
{
```

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

AWS politique gérée : AWSGreengrass ReadOnlyAccess

Vous pouvez associer la politique `AWSGreengrassReadOnlyAccess` à vos identités IAM.

Cette politique accorde des autorisations en lecture seule qui permettent au principal de consulter, mais pas de modifier, les informations contenues dans AWS IoT Greengrass. Par exemple, les principaux disposant de ces autorisations peuvent consulter la liste des composants déployés sur un appareil principal Greengrass, mais ne peuvent pas créer de déploiement pour modifier les composants qui s'exécutent sur cet appareil.

Détails de l'autorisation

Cette politique inclut les autorisations suivantes :

- `greengrass`— Permet aux principaux d'effectuer des actions qui renvoient soit une liste d'éléments, soit des informations sur un élément. Cela inclut les opérations d'API qui commencent par `List` ou `Get`.

JSON

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

```
    ],  
    "Resource": "*"    
  }  
]  
}
```

AWS politique gérée : AWSGreengrass ResourceAccessRolePolicy

Vous pouvez associer la `AWSGreengrassResourceAccessRolePolicy` politique à vos entités IAM. AWS IoT Greengrass associe également cette politique à un rôle de service qui permet AWS IoT Greengrass d'effectuer des actions en votre nom. Pour de plus amples informations, veuillez consulter [Rôle de service Greengrass](#).

Cette politique accorde des autorisations administratives qui permettent d' AWS IoT Greengrass effectuer des tâches essentielles, telles que la récupération de vos fonctions Lambda, la AWS IoT gestion des ombres des appareils et la vérification des appareils clients Greengrass.

Détails de l'autorisation

Cette politique inclut les autorisations suivantes.

- `greengrass`— Gérez les ressources de Greengrass.
- `iot(*Shadow)` — Gérez les AWS IoT ombres dont le nom contient les identifiants spéciaux suivants. Ces autorisations sont nécessaires pour AWS IoT Greengrass pouvoir communiquer avec les appareils principaux.
 - `*-gci`— AWS IoT Greengrass utilise cette ombre pour stocker les informations de connectivité des appareils principaux, afin que les appareils clients puissent découvrir les appareils principaux et s'y connecter.
 - `*-gcm`— La AWS IoT Greengrass V1 utilise cette ombre pour informer le périphérique principal que le certificat d'autorité de certification (CA) du groupe Greengrass a changé.
 - `*-gda`— La AWS IoT Greengrass V1 utilise cette ombre pour informer le périphérique principal d'un déploiement.
 - `GG_*`— Non utilisé.
- `iot(DescribeThingGetDescribeCertificate)` — Récupérez des informations sur AWS IoT des objets et des certificats. Ces autorisations sont nécessaires pour vérifier les AWS IoT Greengrass appareils clients qui se connectent à un périphérique principal. Pour de plus amples informations, veuillez consulter [Interagissez avec les appareils IoT locaux](#).

- **lambda**— Récupère des informations sur AWS Lambda les fonctions. Cette autorisation est requise pour que la AWS IoT Greengrass V1 puisse déployer des fonctions Lambda sur les cœurs de Greengrass. Pour plus d'informations, consultez la section [Exécuter la fonction Lambda sur le AWS IoT Greengrass noyau dans le Guide du](#) développeur de la AWS IoT Greengrass V1.
- **secretsmanager**— Récupère la valeur des AWS Secrets Manager secrets dont le nom commence greengrass- par Cette autorisation est requise pour que la AWS IoT Greengrass V1 puisse déployer les secrets de Secrets Manager sur les cœurs de Greengrass. Pour plus d'informations, consultez la section [Déployer des secrets vers le AWS IoT Greengrass cœur](#) dans le Guide du développeur de la AWS IoT Greengrass V1.
- **s3**— Récupère des objets de fichiers dans des compartiments S3 dont les noms contiennent greengrass ousagemaker. Ces autorisations sont requises pour que AWS IoT Greengrass V1 puisse déployer des ressources d'apprentissage automatique que vous stockez dans des compartiments S3. Pour plus d'informations, consultez les [ressources d'apprentissage automatique](#) dans le guide du développeur de la AWS IoT Greengrass V1.
- **sagemaker**— Récupérez des informations sur les modèles d'inférence d'apprentissage automatique Amazon SageMaker AI. Cette autorisation est requise pour que la AWS IoT Greengrass V1 puisse déployer des modèles ML sur les cœurs de Greengrass. Pour plus d'informations, consultez la section [Effectuer une inférence d'apprentissage automatique](#) dans le guide du développeur de la AWS IoT Greengrass V1.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowGreengrassAccessToShadows",
      "Action": [
        "iot:DeleteThingShadow",
        "iot:GetThingShadow",
        "iot:UpdateThingShadow"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:iot:*:*:thing/GG_*",
        "arn:aws:iot:*:*:thing/*-gcm",
        "arn:aws:iot:*:*:thing/*-gda",
        "arn:aws:iot:*:*:thing/*-gci"
      ]
    }
  ]
}
```

```
    ],
  },
  {
    "Sid": "AllowGreengrassToDescribeThings",
    "Action": [
      "iot:DescribeThing"
    ],
    "Effect": "Allow",
    "Resource": "arn:aws:iot:*:*:thing/*"
  },
  {
    "Sid": "AllowGreengrassToDescribeCertificates",
    "Action": [
      "iot:DescribeCertificate"
    ],
    "Effect": "Allow",
    "Resource": "arn:aws:iot:*:*:cert/*"
  },
  {
    "Sid": "AllowGreengrassToCallGreengrassServices",
    "Action": [
      "greengrass:*"
    ],
    "Effect": "Allow",
    "Resource": "*"
  },
  {
    "Sid": "AllowGreengrassToGetLambdaFunctions",
    "Action": [
      "lambda:GetFunction",
      "lambda:GetFunctionConfiguration"
    ],
    "Effect": "Allow",
    "Resource": "*"
  },
  {
    "Sid": "AllowGreengrassToGetGreengrassSecrets",
    "Action": [
      "secretsmanager:GetSecretValue"
    ],
    "Effect": "Allow",
    "Resource": "arn:aws:secretsmanager:*:*:secret:greengrass-*"
  },
  {
```



```

        "Sid": "AllowGreengrassAccessToS3Objects",
        "Action": [
            "s3:GetObject"
        ],
        "Effect": "Allow",
        "Resource": [
            "arn:aws:s3:::*Greengrass*",
            "arn:aws:s3:::*GreenGrass*",
            "arn:aws:s3:::*greengrass*",
            "arn:aws:s3:::*Sagemaker*",
            "arn:aws:s3:::*SageMaker*",
            "arn:aws:s3:::*sagemaker*"
        ]
    },
    {
        "Sid": "AllowGreengrassAccessToS3BucketLocation",
        "Action": [
            "s3:GetBucketLocation"
        ],
        "Effect": "Allow",
        "Resource": "*"
    },
    {
        "Sid": "AllowGreengrassAccessToSageMakerTrainingJobs",
        "Action": [
            "sagemaker:DescribeTrainingJob"
        ],
        "Effect": "Allow",
        "Resource": [
            "arn:aws:sagemaker:*:*:training-job/*"
        ]
    }
]
}

```

AWS IoT Greengrass mises à jour des politiques AWS gérées

Vous pouvez consulter les informations relatives aux mises à jour des politiques AWS gérées à AWS IoT Greengrass partir du moment où ce service a commencé à suivre ces modifications. Pour recevoir des alertes automatiques concernant les modifications apportées à cette page, abonnez-vous au flux RSS sur la [page d'historique du document AWS IoT Greengrass V2](#).

Modifier	Description	Date
AWS IoT Greengrass a commencé à suivre les modifications	AWS IoT Greengrass a commencé à suivre les modifications apportées AWS à ses politiques gérées.	2 juillet 2021

Prévention du problème de l'adjoint confus entre services

Le problème de délégué confus est un problème de sécurité dans lequel une entité qui n'est pas autorisée à effectuer une action peut contraindre une entité plus privilégiée à le faire. En AWS, l'usurpation d'identité interservices peut entraîner la confusion des adjoints. 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 recommandons d'utiliser les clés de contexte de condition [aws:SourceAccount](#) globale [aws:SourceArn](#) et les clés contextuelles dans les politiques de ressources afin de limiter les autorisations qui AWS IoT Greengrass accordent un autre service à la ressource. 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 stratégie.

La valeur de `aws:SourceArn` doit être la ressource client Greengrass associée à la `sts:AssumeRole` demande.

Le moyen le plus efficace de se protéger contre le problème de délégué 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::account-id:*`.

Pour obtenir un exemple de politique utilisant les clés de contexte de condition `aws:SourceAccount` globale `aws:SourceArn` et les clés contextuelles, consultez [Créer le rôle de service Greengrass](#).

Résolution des problèmes d'identité et d'accès pour AWS IoT Greengrass

Utilisez les informations suivantes pour vous aider à diagnostiquer et à résoudre les problèmes courants que vous pouvez rencontrer lorsque vous travaillez avec AWS IoT Greengrass IAM.

Problèmes

- [Je ne suis pas autorisé à effectuer une action dans AWS IoT Greengrass](#)
- [Je ne suis pas autorisé à effectuer iam : PassRole](#)
- [Je suis administrateur et je souhaite autoriser d'autres personnes à accéder AWS IoT Greengrass](#)
- [Je souhaite permettre à des personnes extérieures Compte AWS à moi d'accéder à mes AWS IoT Greengrass ressources](#)

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 l'utilisateur mateojackson IAM essaie d'afficher les détails d'un appareil principal, mais ne dispose pas des greengrass:GetCoreDevice autorisations nécessaires.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to
perform: greengrass:GetCoreDevice on resource: arn:aws:greengrass:us-
west-2:123456789012:coreDevices/MyGreengrassCore
```

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:coreDevices/MyGreengrassCore` à l'aide de l'action `greengrass:GetCoreDevice`.

Les problèmes généraux liés à l'IAM que vous pouvez rencontrer lors de votre utilisation sont les AWS IoT Greengrass suivants.

Je ne suis pas autorisé à effectuer iam : PassRole

Si vous recevez une erreur selon laquelle vous n'êtes pas autorisé à exécuter `iam:PassRole` l'action, vos stratégies doivent être mises à jour afin de vous permettre de transmettre un rôle à AWS IoT Greengrass.

Certains services AWS permettent de transmettre un rôle existant à ce service au lieu de créer un nouveau rôle de service ou un 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 de 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 besoin d'aide, contactez votre AWS administrateur. Votre administrateur vous a fourni vos informations d'identification de connexion.

Je suis administrateur et je souhaite autoriser d'autres personnes à accéder AWS IoT Greengrass

Pour autoriser d'autres personnes à y accéder AWS IoT Greengrass, vous devez accorder l'autorisation aux personnes ou aux applications qui ont besoin d'y accéder. Si vous utilisez AWS IAM Identity Center pour gérer des personnes et des applications, vous attribuez des ensembles d'autorisations aux utilisateurs ou aux groupes afin de définir leur niveau d'accès. Les ensembles d'autorisations créent et attribuent automatiquement des politiques IAM aux rôles IAM associés à la personne ou à l'application. Pour plus d'informations, consultez la section [Ensembles d'autorisations](#) dans le guide de AWS IAM Identity Center l'utilisateur.

Si vous n'utilisez pas IAM Identity Center, vous devez créer des entités IAM (utilisateurs ou rôles) pour les personnes ou les applications qui ont besoin d'un accès. Vous devez ensuite associer une politique à l'entité qui leur accorde les autorisations appropriées dans AWS IoT Greengrass.

Une fois les autorisations accordées, fournissez les informations d'identification à l'utilisateur ou au développeur de l'application. Ils utiliseront ces informations d'identification pour y accéder AWS. Pour en savoir plus sur la création d'utilisateurs, de groupes, de politiques et d'autorisations [IAM, consultez la section Identités, politiques et autorisations IAM dans le guide de l'utilisateur IAM.](#)

Je souhaite permettre à des personnes extérieures Compte AWS à moi d'accéder à mes AWS IoT Greengrass ressources

Vous pouvez créer un rôle IAM que les utilisateurs d'autres comptes ou des personnes extérieures à votre organisation peuvent utiliser pour accéder à vos AWS ressources. Vous pouvez spécifier qui est approuvé pour assumer le rôle. Pour plus d'informations, consultez les [sections Fourniture d'accès à un utilisateur IAM dans un autre utilisateur Compte AWS que vous possédez](#) et [Fourniture d'accès à Compte AWS des utilisateurs appartenant à des tiers](#) dans le Guide de l'utilisateur IAM.

AWS IoT Greengrass ne prend pas en charge l'accès entre comptes basé sur des politiques basées sur les ressources ou des listes de contrôle d'accès (). ACLs

Autoriser le trafic des appareils via un proxy ou un pare-feu

Les appareils principaux de Greengrass et les composants de Greengrass exécutent les demandes sortantes adressées aux services et à d'autres sites Web AWS . Par mesure de sécurité, vous pouvez limiter le trafic sortant à un petit nombre de terminaux et de ports. Vous pouvez utiliser les informations suivantes sur les points de terminaison et les ports pour limiter le trafic des appareils via un proxy, un pare-feu ou un groupe de [sécurité Amazon VPC](#). Pour plus d'informations sur la configuration d'un périphérique principal pour utiliser un proxy, consultez [Connexion au port 443 ou via un proxy réseau](#).

Rubriques

- [Points de terminaison pour le fonctionnement de base](#)
- [Points de terminaison pour l'installation avec provisionnement automatique](#)
- [Points de terminaison pour les composants AWS fournis](#)

Points de terminaison pour le fonctionnement de base

Les appareils Greengrass Core utilisent les points de terminaison et les ports suivants pour un fonctionnement de base.

Récupérer des points de AWS IoT terminaison

Obtenez les AWS IoT points de terminaison qui vous Compte AWS conviennent et enregistrez-les pour les utiliser ultérieurement. Votre appareil utilise ces points de terminaison pour se connecter à AWS IoT. Procédez comme suit :

1. Obtenez le point de terminaison de AWS IoT données pour votre Compte AWS.

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

La réponse ressemble à l'exemple suivant, si la demande aboutit.

```
{
  "endpointAddress": "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
}
```

2. Obtenez le point de terminaison des informations d' AWS IoT identification pour votre Compte AWS.

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

La réponse ressemble à l'exemple suivant, si la demande aboutit.

```
{
  "endpointAddress": "device-credentials-prefix.credentials.iot.us-
west-2.amazonaws.com"
}
```

Point de terminaison	Port	Obligatoire	Description
greengrass-ats.iot . <i>region</i> .amazonaws.com	8443 ou 443	Oui	Utilisé pour les opérations du plan de données, telles que l'install ation de

Point de terminaison	Port	Obligatoire	Description
			déploiements et l'utilisation de périphériques clients.
<i>device-data-prefix</i> -ats.iot. <i>region</i> .amazonaws.com	MQTT : 8883 ou 443 HTTPS : 8443 ou 443	Oui	Utilisé pour les opérations du plan de données destinées à la gestion des appareils, telles que la communication MQTT et la synchronisation des ombres avec AWS IoT Core.

Point de terminaison	Port	Obligatoire	Description
<i>device-credentials</i> <i>-prefix</i> .credentials.iot. <i>region</i> .amazonaws.com	443	Oui	Utilisé pour acquérir des AWS informations d'identification, que l'appareil principal utilise pour télécharger des artefacts de composants depuis Amazon S3 et effectuer d'autres opérations. Pour de plus amples informations, veuillez consulter Autoriser les appareils principaux à interagir avec les

Point de terminaison	Port	Obligatoire	Description
			AWS services.
<code>*.s3.amazonaws.com</code> <code>*.s3.<i>region</i>.amazonaws.com</code>	443	Oui	Utilisé pour les déploiements. Ce format inclut le * caractère, car les préfixes des points de terminaison sont contrôlés en interne et peuvent changer à tout moment.

Point de terminaison	Port	Obligatoire	Description
data.iot. <i>region</i> .amazonaws.com	443	Non	Nécessaire si le périphérique principal exécute une version du noyau Greengrass antérieure à la version 2.4.0 et est configuré pour utiliser un proxy réseau. Le périphérique principal utilise ce point de terminaison pour communiquer avec le MQTT AWS IoT Core lorsqu'il est derrière un proxy. Pour

Point de terminaison	Port	Obligatoire	Description
			de plus amples informati ons, veuillez consulter Configura tion d'un proxy réseau.

Points de terminaison pour l'installation avec provisionnement automatique

Les appareils Greengrass Core utilisent les points de terminaison et les ports suivants lorsque vous [installez le logiciel AWS IoT Greengrass Core avec provisionnement automatique des ressources.](#)

Point de terminaison	Port	Obligatoire	Description
iot. <i>region</i> .amazonaws.com	443	Oui	Utilisé pour créer des AWS IoT ressource s et récupérer des informati ons sur les AWS IoT ressources existantes.
iam.amazonaws.com	443	Oui	Utilisé pour créer des ressource s IAM et

Point de terminaison	Port	Obligatoire	Description
			recupérer des informations sur les ressources IAM existantes.
<code>sts.<i>region</i>.amazonaws.com</code>	443	Oui	Utilisé pour obtenir l'identifiant de votre Compte AWS.
<code>greengrass.<i>region</i>.amazonaws.com</code>	443	Non	Obligatoire si vous utilisez l'--deploy-dev-tools argument pour déployer le composant Greengrass CLI sur le périphérique principal.

Points de terminaison pour les composants AWS fournis

Les appareils Greengrass Core utilisent des points de terminaison supplémentaires en fonction des composants logiciels qu'ils exécutent. Vous trouverez les points de terminaison requis par chaque

composant AWS fourni dans la section Exigences de la page de chaque composant de ce guide du développeur. Pour de plus amples informations, veuillez consulter [AWS-composants fournis](#).

Validation de 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. Pour plus d'informations sur votre responsabilité en matière de conformité lors de l'utilisation Services AWS, consultez [AWS la documentation de sécurité](#).

Points de terminaison FIPS

AWS IoT Greengrass prend en charge l'utilisation des points de terminaison FIPS ([Federal Information Processing Standard \(FIPS\) 140-2](#)). Lorsque le mode FIPS est activé, toutes les transmissions de données, y compris les protocoles HTTP et MQTT, vers les AWS Cloud services doivent appeler et établir des connexions avec les points de terminaison compatibles FIPS correspondants (FIPS - [Amazon Web Services](#) ()).AWS

Communications MQTT pour AWS IoT utiliser le point de terminaison FIPS du plan de données IoT ([connexion aux points de terminaison AWS IoT FIPS - AWS IoT Core](#)) et la bibliothèque cryptographique compatible FIPS développée par AWS aws-lc.

Pour les communications HTTP dans Greengrass :

- Pour les composants du noyau et du plugin, tous les clients HTTP du SDK sont configurés avec des points de terminaison FIPS en définissant la propriété système sur true ;
AWS_USE_FIPS_ENDPOINT
- Pour les composants génériques, tous les composants commencent par une propriété système AWS_USE_FIPS_ENDPOINT définie sur true. Ce processus garantit que les clients HTTP du SDK utilisés par ces composants génériques envoient des demandes aux points de terminaison conformes à la norme FIPS.

 Note

Dans le cas du gestionnaire de flux, Nucleus transmet la variable d'environnement `AWS_GG_FIPS_MODE`. Cette variable d'environnement permet aux clients HTTP utilisés dans le Stream Manager d'identifier et de se connecter au point de terminaison compatible FIPS correspondant.

AWS IoT Greengrass propose deux méthodes pour activer le mode FIPS : le provisionnement et le déploiement. Pour activer le mode FIPS, vous devez définir le paramètre `fipsMode` de configuration sur `true`, puis Nucleus définit la propriété système sur `AWS_USE_FIPS_ENDPOINT true` et la propage en tant que variable d'environnement à tous les autres composants. En outre, AWS IoT Greengrass téléchargera un certificat CA racine (CA3) et l'ajoutera au fichier `RootCA.pem` (ou `.pem`) existant. AmazonRoot CA1 Si vous activez FIPS via un nouveau déploiement, Nucleus redémarrera pour garantir que les propriétés du système prennent effet après l'activation du mode FIPS.

Outre la configuration du `fipsMode` paramètre, vous devez également configurer les `greengrassDataEndpoint` paramètres `iotDataEndpoint`, `iotCredEndpoint` et. Pour plus d'informations, consultez le document correspondant ci-dessous.

Activez les points de terminaison FIPS grâce au déploiement

Obtenez les AWS IoT points de terminaison qui vous Compte AWS conviennent et enregistrez-les pour les utiliser ultérieurement. Votre appareil utilise ces points de terminaison pour se connecter à AWS IoT. Deux points de terminaison sont requis, le `iotDataEndpoint` et le `iotCredEndpoint`. Procédez comme suit :

1. Obtenez le point de terminaison de données FIPS de votre région dans les points de terminaison du [plan de données AWS IoT Core FIPS](#). Le point de terminaison de données FIPS pour vous Compte AWS devrait ressembler à ceci : `data.iot-fips.us-west-2.amazonaws.com`
2. Obtenez le point de terminaison des informations d'identification FIPS de votre région dans les points de terminaison du [plan de données AWS IoT Core FIPS](#). Le point de terminaison de vos informations d'identification FIPS Compte AWS devrait ressembler à ceci : `data.credentials.iot-fips.us-west-2.amazonaws.com`

Ensuite, pour activer FIPS lors d'un déploiement, vous devez appliquer la configuration suivante à Nucleus. La configuration à fusionner lors du déploiement est la suivante.

Console

Configuration à fusionner

```
{
  "fipsMode": "true",
  "iotDataEndpoint": "data.iot-fips.us-west-2.amazonaws.com",
  "greengrassDataPlaneEndpoint": "iotData",
  "iotCredEndpoint": "data.credentials.iot-fips.us-west-2.amazonaws.com"
}
```

AWS CLI

La commande suivante crée un déploiement sur un périphérique principal.

```
aws greengrassv2 create-deployment --cli-input-json file://dashboard-deployment.json
```

Le `dashboard-deployment.json` fichier contient le document JSON suivant.

```
{
  "targetArn": "arn:aws:iot:us-west-2:123456789012:thing/MyGreengrassCore",
  "deploymentName": "Deployment for MyGreengrassCore",
  "components": {
    "aws.greengrass.Nucleus": {
      "componentVersion": "2.13.0",
      "configurationUpdate": {
        "merge": {"fipsMode": "true", "iotDataEndpoint": "data.iot-fips.us-west-2.amazonaws.com", "greengrassDataPlaneEndpoint": "iotData", "iotCredEndpoint": "data.credentials.iot-fips.us-west-2.amazonaws.com"}
      }
    }
  }
}
```

Greengrass CLI

La commande [Greengrass CLI](#) suivante crée un déploiement local sur un périphérique principal.

```
sudo greengrass-cli deployment create \
  --recipeDir recipes \
  --artifactDir artifacts \
```

```
--merge "aws.greengrass.Nucleus=2.13.0" \  
--update-config dashboard-configuration.json
```

Le `dashboard-configuration.json` fichier contient le document JSON suivant.

```
{  
  "aws.greengrass.Nucleus": {  
    "MERGE": {  
      "fipsMode": "true",  
      "iotDataEndpoint": "data.iot-fips.us-west-2.amazonaws.com",  
      "greengrassDataPlaneEndpoint": "iotData",  
      "iotCredEndpoint": "data.credentials.iot-fips.us-west-2.amazonaws.com"  
    }  
  }  
}
```

Installation de Nucleus avec des points de terminaison FIPS avec provisionnement manuel des ressources

Approvisionnement manuel AWS des ressources pour les appareils AWS IoT Greengrass V2 principaux dotés de points de terminaison FIPS

Important

Avant de télécharger le logiciel AWS IoT Greengrass Core, vérifiez que votre appareil principal répond à la [configuration requise](#) pour installer et exécuter le logiciel AWS IoT Greengrass Core v2.0.

Rubriques

- [Récupérer des points de AWS IoT terminaison](#)
- [Créez n'importe AWS IoT quoi](#)
- [Créez le certificat d'objet](#)
- [Création d'un rôle d'échange de jetons](#)
- [Télécharger les certificats sur l'appareil](#)
- [Configuration de l'environnement de l'appareil](#)

- [Téléchargez le logiciel AWS IoT Greengrass de base](#)
- [Installation du logiciel AWS IoT Greengrass Core](#)

Récupérer des points de AWS IoT terminaison

Obtenez les AWS IoT points de terminaison qui vous Compte AWS conviennent et enregistrez-les pour les utiliser ultérieurement. Votre appareil utilise ces points de terminaison pour se connecter à AWS IoT. Deux points de terminaison sont requis, le `iotDataEndpoint` et le `iotCredEndpoint`. Procédez comme suit :

1. Obtenez le point de terminaison de données FIPS de votre région dans les points de terminaison du [plan de données AWS IoT Core FIPS](#). Le point de terminaison de données FIPS pour vous Compte AWS devrait ressembler à ceci : `data.iot-fips.us-west-2.amazonaws.com`
2. Obtenez le point de terminaison des informations d'identification FIPS de votre région dans les points de terminaison du [plan de données AWS IoT Core FIPS](#). Le point de terminaison de vos informations d'identification FIPS Compte AWS devrait ressembler à ceci : `data.credentials.iot-fips.us-west-2.amazonaws.com`

Créez n'importe AWS IoT quoi

AWS IoT les objets représentent les appareils et les entités logiques auxquels ils se connectent AWS IoT. Les appareils Greengrass Core sont AWS IoT des objets. Lorsque vous enregistrez un appareil en tant qu' AWS IoT objet, celui-ci peut utiliser un certificat numérique pour s'authentifier. AWS

Dans cette section, vous allez créer un AWS IoT objet qui représente votre appareil.

Pour créer AWS IoT quelque chose

1. Créez AWS IoT quelque chose pour votre appareil. Sur votre ordinateur de développement, exécutez la commande suivante.
 - Remplacez `MyGreengrassCore` par le nom de l'objet à utiliser. Ce nom est également le nom de votre appareil principal Greengrass.

Note

Le nom de l'objet ne peut pas contenir de caractères deux-points (:).

```
aws iot create-thing --thing-name MyGreengrassCore
```

La réponse ressemble à l'exemple suivant, si la demande aboutit.

```
{
  "thingName": "MyGreengrassCore",
  "thingArn": "arn:aws:iot:us-west-2:123456789012:thing/MyGreengrassCore",
  "thingId": "8cb4b6cd-268e-495d-b5b9-1713d71dbf42"
}
```

2. (Facultatif) Ajoutez l' AWS IoT objet à un nouveau groupe d'objets ou à un groupe d'objets existant. Vous utilisez des groupes d'objets pour gérer des flottes d'appareils principaux de Greengrass. Lorsque vous déployez des composants logiciels sur vos appareils, vous pouvez cibler des appareils individuels ou des groupes d'appareils. Vous pouvez ajouter un appareil à un groupe d'objets avec un déploiement Greengrass actif pour déployer les composants logiciels de ce groupe d'objets sur l'appareil. Procédez comme suit :

- a. (Facultatif) Créez un AWS IoT groupe d'objets.

- Remplacez *MyGreengrassCoreGroup* par le nom du groupe d'objets à créer.

 Note

Le nom du groupe d'objets ne peut pas contenir de deux-points (:).

```
aws iot create-thing-group --thing-group-name MyGreengrassCoreGroup
```

La réponse ressemble à l'exemple suivant, si la demande aboutit.

```
{
  "thingGroupName": "MyGreengrassCoreGroup",
  "thingGroupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/MyGreengrassCoreGroup",
  "thingGroupId": "4df721e1-ff9f-4f97-92dd-02db4e3f03aa"
}
```

- b. Ajoutez l' AWS IoT objet à un groupe d'objets.

- Remplacez *MyGreengrassCore* par le nom de votre AWS IoT objet.
- Remplacez *MyGreengrassCoreGroup* par le nom du groupe d'objets.

```
aws iot add-thing-to-thing-group --thing-name MyGreengrassCore --thing-group-name MyGreengrassCoreGroup
```

La commande n'a aucune sortie si la demande aboutit.

Créez le certificat d'objet

Lorsque vous enregistrez un appareil en tant qu' AWS IoT objet, celui-ci peut utiliser un certificat numérique pour s'authentifier. AWS Ce certificat permet à l'appareil de communiquer avec AWS IoT et AWS IoT Greengrass.

Dans cette section, vous allez créer et télécharger des certificats que votre appareil peut utiliser pour se connecter AWS.

Si vous souhaitez configurer le logiciel AWS IoT Greengrass Core pour utiliser un module de sécurité matériel (HSM) pour stocker en toute sécurité la clé privée et le certificat, suivez les étapes pour créer le certificat à partir d'une clé privée dans un HSM. Sinon, suivez les étapes pour créer le certificat et la clé privée dans le AWS IoT service. La fonctionnalité de sécurité matérielle n'est disponible que sur les appareils Linux. Pour plus d'informations sur la sécurité matérielle et les conditions requises pour l'utiliser, consultez [Intégration de sécurité matérielle](#).

Créez le certificat et la clé privée dans le AWS IoT service

Pour créer le certificat d'objet

1. Créez un dossier dans lequel vous téléchargerez les certificats de l' AWS IoT objet.

```
mkdir greengrass-v2-certs
```

2. Créez et téléchargez les certificats AWS IoT correspondants.

```
aws iot create-keys-and-certificate --set-as-active --certificate-pem-outfile greengrass-v2-certs/device.pem.crt --public-key-outfile greengrass-v2-certs/public.pem.key --private-key-outfile greengrass-v2-certs/private.pem.key
```

La réponse ressemble à l'exemple suivant, si la demande aboutit.

```
{
  "certificateArn": "arn:aws:iot:us-west-2:123456789012:cert/
aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4",
  "certificateId":
"aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4",
  "certificatePem": "-----BEGIN CERTIFICATE-----
MIICiTCCAfICCQD6m7oRw0uX0jANBgkqhkiG9w
0BAQUFADCBiDELMAkGA1UEBhMCVVMxCzAJBgNVBAGTAldBMRAwDgYDVQQHEwdTZ
WF0dGx1MQ8wDQYDVQQKEwZBbWF6b24xFDASBgNVBASTC0lBTSBDb25zb2x1MRIw
EAYDVQQDEw1UZXRhbnQ2L2sYWMxH2AdBgkqhkiG9w0BCQEWEG5vb25lQGfYXpvi5
jb20wHhcNMTEwNDI1MjA0NTIxWhcNMTEwNDI1MjA0NTIxWjCBiDELMAkGA1UEBh
MCVVMxCzAJBgNVBAGTAldBMRAwDgYDVQQHEwdTZWF0dGx1MQ8wDQYDVQQKEwZBb
WF6b24xFDASBgNVBASTC0lBTSBDb25zb2x1MRIwEAYDVQQDEw1UZXRhbnQ2L2sYWMx
H2AdBgkqhkiG9w0BCQEWEG5vb25lQGfYXpvi5jb20wgZ8wDQYJKoZIhvcNAQE
BBQADgY0AMIGJAoGBAMaK0dn+a4GmWIWJ21uUSfwfEvySWtC2XADZ4nB+BLygVI
k60CpiwsZ3G93vUEIO3IyNoH/f0wYK8m9TrDHudUZg3qX4waLG5M43q7Wgc/MbQ
ITx0USQv7c7ugFFDzQGBzZswY6786m86gpEIbb30hjZnzcVQAaRHhd1QWIMm2nr
AgMBAAEwDQYJKoZIhvcNAQEFBQADgYEAtCu4nUhVVxYUntneD9+h8Mg9q6q+auN
KyExzyLwaxlAoo7TJHidbtS4J5iNmZgXL0FkbFFBjvSfpJI1J00zbhNYS5f6Guo
EDmFJl0ZxBHjJnyp3780D8uTs7fLvjx79LjStbNYiytVbZPQUQ5Yaxu2jXnimvw
3rrszlaEXAMPLE=
-----END CERTIFICATE-----",
  "keyPair": {
    "PublicKey": "-----BEGIN PUBLIC KEY-----\
MIIBIjANBgkqhkiG9w0BAQ0CAQ8AMIIBCgKCAQEAEXAMPLE1nnyJwKSMHw4h\
MMEXAMPLEEuuN/dMAS3fyce8DW/4+EXAMPLEYjmoF/YVF/gHr99VEEXAMPLE5VF13\
59VK7cEXAMPLE67GK+y+jikqX0gHh/xJTWO
+sGpWEXAMPLEDz18x0d2ka4tCzuWEXAMPLEEahJbYkCPUBSU8opVkr7qkEXAMPLE1DR6sx2Hocli00Ltu6Fkw91swQWE
\\GB3ZPrNh0PzQYvjUStZeccyNCx2EXAMPLEvp9mQ0UXP6plfgxwKRX2fEXAMPLEDa\
hJLXkX3rHU2xbxJSq7D+XEXAMPLEcw+LyFhI5mgFR188eGdsAEXAMPLE1nI9EesG\
FQIDAQAB\
-----END PUBLIC KEY-----\
",
    "PrivateKey": "-----BEGIN RSA PRIVATE KEY-----\
key omitted for security reasons\
-----END RSA PRIVATE KEY-----\
"
  }
}
```

Enregistrez le nom de ressource Amazon (ARN) du certificat afin de l'utiliser pour configurer le certificat ultérieurement.

Créez le certificat à partir d'une clé privée dans un HSM

 Note

Cette fonctionnalité est disponible pour les versions 2.5.3 et ultérieures du composant [Greengrass](#) nucleus. AWS IoT Greengrass ne prend actuellement pas en charge cette fonctionnalité sur les appareils Windows principaux.

Pour créer le certificat d'objet

1. Sur le périphérique principal, initialisez un jeton PKCS #11 dans le HSM et générez une clé privée. La clé privée doit être une clé RSA de taille RSA-2048 (ou supérieure) ou une clé ECC.

 Note

Pour utiliser un module de sécurité matériel avec des clés ECC, vous devez utiliser [Greengrass](#) nucleus v2.5.6 ou version ultérieure.

Pour utiliser un module de sécurité matériel et un [gestionnaire de secrets](#), vous devez utiliser un module de sécurité matériel doté de clés RSA.

Consultez la documentation de votre HSM pour savoir comment initialiser le jeton et générer la clé privée. Si votre HSM prend en charge l'objet IDs, spécifiez un ID d'objet lorsque vous générez la clé privée. Enregistrez l'ID d'emplacement, le code PIN utilisateur, le libellé de l'objet, l'ID d'objet (si votre HSM en utilise un) que vous spécifiez lorsque vous initialisez le jeton et générez la clé privée. Vous utiliserez ces valeurs ultérieurement lorsque vous importez le certificat d'objet dans le HSM et que vous configurez le logiciel AWS IoT Greengrass principal.

2. Créez une demande de signature de certificat (CSR) à partir de la clé privée. AWS IoT utilise ce CSR pour créer un certificat d'objet pour la clé privée que vous avez générée dans le HSM. Pour plus d'informations sur la création d'un CSR à partir de la clé privée, consultez la documentation de votre HSM. Le CSR est un fichier, tel que `iotdevicekey.csr`.

3. Copiez le CSR de l'appareil vers votre ordinateur de développement. Si SSH et SCP sont activés sur l'ordinateur de développement et le périphérique, vous pouvez utiliser la `scp` commande de votre ordinateur de développement pour transférer le CSR. Remplacez *device-ip-address* par l'adresse IP de votre appareil, puis *~/iotdevicekey.csr* par le chemin d'accès au fichier CSR sur l'appareil.

```
scp device-ip-address:~/iotdevicekey.csr iotdevicekey.csr
```

4. Sur votre ordinateur de développement, créez un dossier dans lequel vous téléchargerez le certificat AWS IoT correspondant.

```
mkdir greengrass-v2-certs
```

5. Utilisez le fichier CSR pour créer et télécharger le certificat correspondant à l' AWS IoT objet sur votre ordinateur de développement.

```
aws iot create-certificate-from-csr --set-as-active --certificate-signing-request=file:///iotdevicekey.csr --certificate-pem-outfile greengrass-v2-certs/device.pem.crt
```

La réponse ressemble à l'exemple suivant, si la demande aboutit.

```
{
  "certificateArn": "arn:aws:iot:us-west-2:123456789012:cert/aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4",
  "certificateId":
    "aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4",
  "certificatePem": "-----BEGIN CERTIFICATE-----
MIICiTCCAfICCD6m7oRw0uX0jANBgkqhkiG9w
0BAQUFADCBiDELMAkGA1UEBhMCVVMxCzAJBgNVBAGTAldBMRAwDgYDVQQHEwdTZ
WF0dGx1MQ8wDQYDVQQKEwZBbWF6b24xFDASBgNVBA5TC0lBTSBDb25zb2x1MRIw
EAYDVQQDEw1UZXR0Q21sYWxhZG9w0BCQEWEG5vb25lQGFTYXpvi5
jb20wHhcNMTEwNDI1MjA0NTIxWhcNMTEwNDI1MjA0NTIxWjCBiDELMAkGA1UEBh
MCVVMxCzAJBgNVBAGTAldBMRAwDgYDVQQHEwdTZWF0dGx1MQ8wDQYDVQQKEwZBb
WF6b24xFDASBgNVBA5TC0lBTSBDb25zb2x1MRIwEAYDVQQDEw1UZXR0Q21sYWxh
ZG9w0BCQEWEG5vb25lQGFTYXpvi5jb20wgZ8wDQYJKoZIhvcNAQEE
BBQADgY0AMIGJAoGBAMaK0dn+a4GmWIWJ21uUSfwfEvySWtC2XADZ4nB+BLygVI
k60CpiwsZ3G93vUEIO3IyNoH/f0wYK8m9TrDHudUZg3qX4waLG5M43q7Wgc/MbQ
ITx0USQv7c7ugFFDzQGBzZswY6786m86gpEibb30hjZnzcVQAaRHhd1QWIMm2nr
AgMBAAEwDQYJKoZIhvcNAQEFBQADgYEAtCu4nUhVVxYUntneD9+h8Mg9q6q+auN
KyExzyLwax1Aoo7TJHidbtS4J5iNmZgXL0FkbFFBjvSfpJI1J00zbhNYS5f6Guo
-----"
```

```
EDmFJl0ZxBHjJnyp3780D8uTs7fLvJx79LjSTbNYiytVbZPQUQ5Yaxu2jXnimvw
3rrszlaEXAMPLE=
-----END CERTIFICATE-----"
}
```

Enregistrez l'ARN du certificat afin de l'utiliser pour le configurer ultérieurement.

Configurez ensuite le certificat d'objet. Pour de plus amples informations, veuillez consulter [Configurer le certificat d'objet](#).

Création d'un rôle d'échange de jetons

Les appareils principaux de Greengrass utilisent un rôle de service IAM, appelé rôle d'échange de jetons, pour autoriser les appels aux services. AWS L'appareil utilise le fournisseur AWS IoT d'informations d'identification pour obtenir des AWS informations d'identification temporaires pour ce rôle, ce qui lui permet d'interagir avec Amazon Logs AWS IoT, d'envoyer des journaux à Amazon CloudWatch Logs et de télécharger des artefacts de composants personnalisés depuis Amazon S3. Pour de plus amples informations, veuillez consulter [Autoriser les appareils principaux à interagir avec les AWS services](#).

Vous utilisez un alias de AWS IoT rôle pour configurer le rôle d'échange de jetons pour les appareils principaux de Greengrass. Les alias de rôle vous permettent de modifier le rôle d'échange de jetons d'un appareil tout en conservant la même configuration de l'appareil. Pour plus d'informations, consultez la section [Autorisation des appels directs vers AWS des services](#) dans le Guide du AWS IoT Core développeur.

Dans cette section, vous allez créer un rôle IAM d'échange de jetons et un alias de AWS IoT rôle pointant vers le rôle. Si vous avez déjà configuré un appareil principal Greengrass, vous pouvez utiliser son rôle d'échange de jetons et son alias de rôle au lieu d'en créer de nouveaux. Ensuite, vous configurez l'appareil pour AWS IoT qu'il utilise ce rôle et cet alias.

Pour créer un rôle IAM d'échange de jetons

1. Créez un rôle IAM que votre appareil peut utiliser comme rôle d'échange de jetons. Procédez comme suit :
 - a. Créez un fichier contenant le document de politique de confiance requis par le rôle d'échange de jetons.

Par exemple, sur un système basé sur Linux, vous pouvez exécuter la commande suivante pour utiliser GNU nano pour créer le fichier.

```
nano device-role-trust-policy.json
```

Copiez le code JSON suivant dans le fichier.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "credentials.iot.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

- b. Créez le rôle d'échange de jetons avec le document de politique de confiance.
- Remplacez *GreengrassV2TokenExchangeRole* par le nom du rôle IAM à créer.

```
aws iam create-role --role-name GreengrassV2TokenExchangeRole --assume-role-policy-document file://device-role-trust-policy.json
```

La réponse ressemble à l'exemple suivant, si la demande aboutit.

```
{
  "Role": {
    "Path": "/",
    "RoleName": "GreengrassV2TokenExchangeRole",
    "RoleId": "AR0AZ2YMUHYHK50KM77FB",
    "Arn": "arn:aws:iam::123456789012:role/GreengrassV2TokenExchangeRole",
    "CreateDate": "2021-02-06T00:13:29+00:00",
    "AssumeRolePolicyDocument": {
      "Version": "2012-10-17",
      "Statement": [
        {
```



```
        "Effect": "Allow",
        "Principal": {
            "Service": "credentials.iot.amazonaws.com"
        },
        "Action": "sts:AssumeRole"
    }
]
}
```

- c. Créez un fichier contenant le document de politique d'accès requis par le rôle d'échange de jetons.

Par exemple, sur un système basé sur Linux, vous pouvez exécuter la commande suivante pour utiliser GNU nano pour créer le fichier.

```
nano device-role-access-policy.json
```

Copiez le code JSON suivant dans le fichier.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams",
        "s3:GetBucketLocation"
      ],
      "Resource": "*"
    }
  ]
}
```

 Note

Cette politique d'accès n'autorise pas l'accès aux artefacts des composants dans les compartiments S3. Pour déployer des composants personnalisés qui définissent

des artefacts dans Amazon S3, vous devez ajouter des autorisations au rôle afin de permettre à votre appareil principal de récupérer les artefacts des composants. Pour de plus amples informations, veuillez consulter [Autoriser l'accès aux compartiments S3 pour les artefacts des composants](#).

Si vous ne possédez pas encore de compartiment S3 pour les artefacts des composants, vous pouvez ajouter ces autorisations ultérieurement après avoir créé un compartiment.

d. Créez la politique IAM à partir du document de stratégie.

- Remplacez *GreengrassV2TokenExchangeRoleAccess* par le nom de la politique IAM à créer.

```
aws iam create-policy --policy-name GreengrassV2TokenExchangeRoleAccess --  
policy-document file://device-role-access-policy.json
```

La réponse ressemble à l'exemple suivant, si la demande aboutit.

```
{  
  "Policy": {  
    "PolicyName": "GreengrassV2TokenExchangeRoleAccess",  
    "PolicyId": "ANPAZ2YMUHYHACI7C5Z66",  
    "Arn": "arn:aws:iam::123456789012:policy/  
GreengrassV2TokenExchangeRoleAccess",  
    "Path": "/",  
    "DefaultVersionId": "v1",  
    "AttachmentCount": 0,  
    "PermissionsBoundaryUsageCount": 0,  
    "IsAttachable": true,  
    "CreateDate": "2021-02-06T00:37:17+00:00",  
    "UpdateDate": "2021-02-06T00:37:17+00:00"  
  }  
}
```

e. Associez la politique IAM au rôle d'échange de jetons.

- Remplacez *GreengrassV2TokenExchangeRole* par le nom du rôle IAM.
- Remplacez l'ARN de la stratégie par l'ARN de la stratégie IAM que vous avez créée à l'étape précédente.

```
aws iam attach-role-policy --role-name GreengrassV2TokenExchangeRole --policy-arn arn:aws:iam::123456789012:policy/GreengrassV2TokenExchangeRoleAccess
```

La commande n'a aucune sortie si la demande aboutit.

2. Créez un alias de AWS IoT rôle qui pointe vers le rôle d'échange de jetons.

- Remplacez *GreengrassCoreTokenExchangeRoleAlias* par le nom de l'alias de rôle à créer.
- Remplacez l'ARN du rôle par l'ARN du rôle IAM que vous avez créé à l'étape précédente.

```
aws iot create-role-alias --role-alias GreengrassCoreTokenExchangeRoleAlias --role-arn arn:aws:iam::123456789012:role/GreengrassV2TokenExchangeRole
```

La réponse ressemble à l'exemple suivant, si la demande aboutit.

```
{
  "roleAlias": "GreengrassCoreTokenExchangeRoleAlias",
  "roleAliasArn": "arn:aws:iot:us-west-2:123456789012:rolealias/GreengrassCoreTokenExchangeRoleAlias"
}
```

 Note

Pour créer un alias de rôle, vous devez être autorisé à transmettre le rôle IAM d'échange de jetons à AWS IoT. Si vous recevez un message d'erreur lorsque vous essayez de créer un alias de rôle, vérifiez que votre AWS utilisateur dispose de cette autorisation. Pour plus d'informations, consultez la section [Octroi à un utilisateur des autorisations lui permettant de transférer un rôle à un AWS service](#) dans le Guide de Gestion des identités et des accès AWS l'utilisateur.

3. Créez et attachez une AWS IoT politique qui permet à votre appareil principal Greengrass d'utiliser l'alias de rôle pour assumer le rôle d'échange de jetons. Si vous avez déjà configuré un appareil principal Greengrass, vous pouvez associer sa AWS IoT politique d'alias de rôle au lieu d'en créer une nouvelle. Procédez comme suit :

- a. (Facultatif) Créez un fichier contenant le document AWS IoT de politique requis par l'alias de rôle.

Par exemple, sur un système basé sur Linux, vous pouvez exécuter la commande suivante pour utiliser GNU nano pour créer le fichier.

```
nano greengrass-v2-iot-role-alias-policy.json
```

Copiez le code JSON suivant dans le fichier.

- Remplacez l'ARN de la ressource par l'ARN de votre alias de rôle.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iot:AssumeRoleWithCertificate",
      "Resource": "arn:aws:iot:us-west-2:123456789012:rolealias/
GreengrassCoreTokenExchangeRoleAlias"
    }
  ]
}
```

- b. Créez une AWS IoT politique à partir du document de stratégie.

- Remplacez *GreengrassCoreTokenExchangeRoleAliasPolicy* par le nom de la AWS IoT politique à créer.

```
aws iot create-policy --policy-name GreengrassCoreTokenExchangeRoleAliasPolicy
--policy-document file://greengrass-v2-iot-role-alias-policy.json
```

La réponse ressemble à l'exemple suivant, si la demande aboutit.

```
{
  "policyName": "GreengrassCoreTokenExchangeRoleAliasPolicy",
  "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassCoreTokenExchangeRoleAliasPolicy",
}
```

```

"policyDocument": "{
  \"Version\": \"2012-10-17\",
  \"Statement\": [
    {
      \"Effect\": \"Allow\",
      \"Action\": \"iot:AssumeRoleWithCertificate\",
      \"Resource\": \"arn:aws:iot:us-west-2:123456789012:rolealias/
GreengrassCoreTokenExchangeRoleAlias\"
    }
  ]
},
\"policyVersionId\": \"1\"
}"

```

c. Joignez la AWS IoT politique au certificat de l' AWS IoT objet.

- Remplacez *GreengrassCoreTokenExchangeRoleAliasPolicy* par le nom de la AWS IoT politique d'alias de rôle.
- Remplacez l'ARN cible par l'ARN du certificat associé à votre AWS IoT objet.

```

aws iot attach-policy --policy-name GreengrassCoreTokenExchangeRoleAliasPolicy
--target arn:aws:iot:us-west-2:123456789012:cert/
aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4

```

La commande n'a aucune sortie si la demande aboutit.

Télécharger les certificats sur l'appareil

Auparavant, vous avez téléchargé le certificat de votre appareil sur votre ordinateur de développement. Dans cette section, vous allez copier le certificat sur votre appareil principal pour configurer celui-ci avec les certificats auxquels il se connecte AWS IoT. Vous téléchargez également le certificat de l'autorité de certification racine (CA) Amazon. Si vous utilisez un HSM, vous importez également le fichier de certificat dans le HSM dans cette section.

- Si vous avez créé le certificat d'objet et la clé privée dans le AWS IoT service précédemment, suivez les étapes pour télécharger les certificats avec la clé privée et les fichiers de certificat.
- Si vous avez créé le certificat d'objet à partir d'une clé privée dans un module de sécurité matériel (HSM) plus tôt, suivez les étapes pour télécharger les certificats avec la clé privée et le certificat dans un HSM.

Téléchargez des certificats avec clé privée et fichiers de certificats

Pour télécharger des certificats sur l'appareil

1. Copiez le certificat d' AWS IoT objet de votre ordinateur de développement sur l'appareil. Si SSH et SCP sont activés sur l'ordinateur de développement et le périphérique, vous pouvez utiliser la scp commande de votre ordinateur de développement pour transférer le certificat. Remplacez *device-ip-address* par l'adresse IP de votre appareil.

```
scp -r greengrass-v2-certs/ device-ip-address:~
```

2. Créez le dossier racine Greengrass sur l'appareil. Vous installerez ultérieurement le logiciel AWS IoT Greengrass Core dans ce dossier.

Note

Pour Windows, la limitation de la longueur du chemin est de 260 caractères. Si vous utilisez Windows, utilisez un dossier racine tel que C:\greengrass\v2 ou D:\greengrass\v2 pour conserver les chemins des composants Greengrass en dessous de la limite de 260 caractères.

Linux or Unix

- */greengrass/v2* Remplacez-le par le dossier à utiliser.

```
sudo mkdir -p /greengrass/v2
```

Windows Command Prompt

- *C:\greengrass\v2* Remplacez-le par le dossier à utiliser.

```
mkdir C:\greengrass\v2
```

PowerShell

- *C:\greengrass\v2* Remplacez-le par le dossier à utiliser.

```
mkdir C:\greengrass\v2
```

3. (Linux uniquement) Définissez les autorisations du parent du dossier racine de Greengrass.

- Remplacez `/greengrass` par le parent du dossier racine.

```
sudo chmod 755 /greengrass
```

4. Copiez les AWS IoT certificats d'objets dans le dossier racine de Greengrass.

Linux or Unix

- `/greengrass/v2` Remplacez-le par le dossier racine de Greengrass.

```
sudo cp -R ~/greengrass-v2-certs/* /greengrass/v2
```

Windows Command Prompt

- `C:\greengrass\v2` Remplacez-le par le dossier à utiliser.

```
robocopy %USERPROFILE%\greengrass-v2-certs C:\greengrass\v2 /E
```

PowerShell

- `C:\greengrass\v2` Remplacez-le par le dossier à utiliser.

```
cp -Path ~\greengrass-v2-certs\* -Destination C:\greengrass\v2
```

5. Téléchargez le certificat de l'autorité de certification racine (CA) Amazon. AWS IoT les certificats sont associés par défaut au certificat de l'autorité de certification racine d'Amazon. Téléchargez le CA1 certificat et le [CA3certificat](#).

Linux or Unix

- Remplacez `/greengrass/v2` ou `C:\greengrass\v2` par le dossier racine de Greengrass.

```
sudo curl -o /greengrass/v2/AmazonRootCA1.pem https://www.amazontrust.com/
repository/AmazonRootCA1.pem
sudo curl -o - https://www.amazontrust.com/repository/AmazonRootCA3.pem >> /
greengrass/v2/AmazonRootCA1.pem
```

Windows Command Prompt (CMD)

```
curl -o C:\greengrass\v2\AmazonRootCA1.pem https://www.amazontrust.com/
repository/AmazonRootCA1.pem
```

PowerShell

```
iwr -Uri https://www.amazontrust.com/repository/AmazonRootCA1.pem -OutFile C:
\greengrass\v2\AmazonRootCA1.pem
```

Téléchargez les certificats avec la clé privée et le certificat dans un HSM

Note

Cette fonctionnalité est disponible pour les versions 2.5.3 et ultérieures du composant [Greengrass](#) nucleus. AWS IoT Greengrass ne prend actuellement pas en charge cette fonctionnalité sur les appareils Windows principaux.

Pour télécharger des certificats sur l'appareil

1. Copiez le certificat d'AWS IoT objet de votre ordinateur de développement sur l'appareil. Si SSH et SCP sont activés sur l'ordinateur de développement et le périphérique, vous pouvez utiliser la scp commande de votre ordinateur de développement pour transférer le certificat. Remplacez *device-ip-address* par l'adresse IP de votre appareil.

```
scp -r greengrass-v2-certs/ device-ip-address:~
```

2. Créez le dossier racine Greengrass sur l'appareil. Vous installerez ultérieurement le logiciel AWS IoT Greengrass Core dans ce dossier.

 Note

Pour Windows, la limitation de la longueur du chemin est de 260 caractères. Si vous utilisez Windows, utilisez un dossier racine tel que C:\greengrass\v2 ou D:\greengrass\v2 pour conserver les chemins des composants Greengrass en dessous de la limite de 260 caractères.

Linux or Unix

- `/greengrass/v2` Remplacez-le par le dossier à utiliser.

```
sudo mkdir -p /greengrass/v2
```

Windows Command Prompt

- `C:\greengrass\v2` Remplacez-le par le dossier à utiliser.

```
mkdir C:\greengrass\v2
```

PowerShell

- `C:\greengrass\v2` Remplacez-le par le dossier à utiliser.

```
mkdir C:\greengrass\v2
```

3. (Linux uniquement) Définissez les autorisations du parent du dossier racine de Greengrass.

- Remplacez `/greengrass` par le parent du dossier racine.

```
sudo chmod 755 /greengrass
```

4. Importez le fichier de certificat d'~/greengrass-v2-certs/device.pem.crt objet dans le HSM. Consultez la documentation de votre HSM pour savoir comment y importer des certificats. Importez le certificat en utilisant le jeton, l'ID de slot, le code PIN utilisateur, le même libellé

d'objet et le même ID d'objet (si votre HSM en utilise un) où vous avez généré la clé privée dans le HSM précédemment.

Note

Si vous avez généré la clé privée plus tôt sans ID d'objet et que le certificat possède un ID d'objet, définissez l'ID d'objet de la clé privée sur la même valeur que le certificat. Consultez la documentation de votre HSM pour savoir comment définir l'ID d'objet pour l'objet clé privée.

5. (Facultatif) Supprimez le fichier de certificat d'objet afin qu'il n'existe que dans le HSM.

```
rm ~/greengrass-v2-certs/device.pem.crt
```

6. Téléchargez le certificat de l'autorité de certification racine (CA) Amazon. AWS IoT les certificats sont associés par défaut au certificat de l'autorité de certification racine d'Amazon. Téléchargez à la fois le CA1 et le [CA3certificat](#).

Linux or Unix

- Remplacez */greengrass/v2* ou *C:\greengrass\v2* par le dossier racine de Greengrass.

```
sudo curl -o /greengrass/v2/AmazonRootCA1.pem https://www.amazontrust.com/
repository/AmazonRootCA1.pem
sudo curl -o - https://www.amazontrust.com/repository/AmazonRootCA3.pem >> /
greengrass/v2/AmazonRootCA1.pem
```

Windows Command Prompt (CMD)

```
curl -o C:\greengrass\v2\AmazonRootCA1.pem https://www.amazontrust.com/
repository/AmazonRootCA1.pem
```

PowerShell

```
iwr -Uri https://www.amazontrust.com/repository/AmazonRootCA1.pem -OutFile C:
\greengrass\v2\AmazonRootCA1.pem
```

Configuration de l'environnement de l'appareil

Suivez les étapes décrites dans cette section pour configurer un appareil Linux ou Windows à utiliser comme périphérique AWS IoT Greengrass principal.

Configuration d'un appareil Linux

Pour configurer un appareil Linux pour AWS IoT Greengrass V2

1. Installez le moteur d'exécution Java, dont le logiciel AWS IoT Greengrass Core a besoin pour fonctionner. Nous vous recommandons d'utiliser les versions de support à long terme d'[Amazon Corretto](#) ou d'OpenJDK. La version 8 ou supérieure est requise. Les commandes suivantes vous montrent comment installer OpenJDK sur votre appareil.

- Pour les distributions basées sur Debian ou Ubuntu :

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

- Pour les distributions basées sur Red Hat :

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

- Dans Amazon Linux 2 :

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

- Pour Amazon Linux 2023 :

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

Lorsque l'installation est terminée, exécutez la commande suivante pour vérifier que Java s'exécute sur votre appareil Linux.

```
java -version
```

La commande affiche la version de Java exécutée sur le périphérique. Par exemple, sur une distribution basée sur Debian, le résultat peut ressembler à celui de l'exemple suivant.

```
openjdk version "11.0.9.1" 2020-11-04
```

```
OpenJDK Runtime Environment (build 11.0.9.1+1-post-Debian-1deb10u2)
OpenJDK 64-Bit Server VM (build 11.0.9.1+1-post-Debian-1deb10u2, mixed mode)
```

2. (Facultatif) Créez l'utilisateur système et le groupe par défaut qui exécutent les composants sur le périphérique. Vous pouvez également choisir de laisser le programme d'installation du logiciel AWS IoT Greengrass Core créer cet utilisateur et ce groupe lors de l'installation avec l'argument `--component-default-user` installer. Pour de plus amples informations, veuillez consulter [Arguments d'installation](#).

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

3. Vérifiez que l'utilisateur qui exécute le logiciel AWS IoT Greengrass Core (généralement `root`) est autorisé à exécuter `sudo` avec n'importe quel utilisateur et n'importe quel groupe.

- a. Exécutez la commande suivante pour ouvrir le `/etc/sudoers` fichier.

```
sudo visudo
```

- b. Vérifiez que l'autorisation accordée à l'utilisateur ressemble à l'exemple suivant.

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

4. (Facultatif) Pour [exécuter des fonctions Lambda conteneurisées](#), vous devez activer [cgroups](#) v1, et vous devez activer et monter les cgroups de mémoire et de périphériques. Si vous ne prévoyez pas d'exécuter des fonctions Lambda conteneurisées, vous pouvez ignorer cette étape.

Pour activer ces options cgroups, démarrez le périphérique avec les paramètres du noyau Linux suivants.

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

Pour plus d'informations sur l'affichage et la définition des paramètres du noyau de votre appareil, consultez la documentation de votre système d'exploitation et de votre chargeur de démarrage. Suivez les instructions pour définir définitivement les paramètres du noyau.

5. Installez toutes les autres dépendances requises sur votre appareil, comme indiqué dans la liste des exigences figurant dans [Exigences relatives aux dispositifs](#).

Configuration d'un appareil Windows

Note

Cette fonctionnalité est disponible pour les versions 2.5.0 et ultérieures du composant [Greengrass](#) nucleus.

Pour configurer un appareil Windows pour AWS IoT Greengrass V2

1. Installez le moteur d'exécution Java, dont le logiciel AWS IoT Greengrass Core a besoin pour fonctionner. Nous vous recommandons d'utiliser les versions de support à long terme d'[Amazon Corretto](#) ou d'OpenJDK. La version 8 ou supérieure est requise.
2. Vérifiez si Java est disponible sur la variable système [PATH](#), et ajoutez-le dans le cas contraire. Le LocalSystem compte exécute le logiciel AWS IoT Greengrass Core. Vous devez donc ajouter Java à la variable système PATH au lieu de la variable utilisateur PATH pour votre utilisateur. Procédez comme suit :
 - a. Appuyez sur la touche Windows pour ouvrir le menu de démarrage.
 - b. Tapez **environment variables** pour rechercher les options du système dans le menu Démarrer.
 - c. Dans les résultats de recherche du menu Démarrer, choisissez Modifier les variables d'environnement du système pour ouvrir la fenêtre des propriétés du système.
 - d. Choisissez les variables d'environnement... pour ouvrir la fenêtre des variables d'environnement.
 - e. Sous Variables système, sélectionnez Chemin, puis Modifier. Dans la fenêtre Modifier la variable d'environnement, vous pouvez afficher chaque chemin sur une ligne distincte.
 - f. Vérifiez si le chemin d'accès au bin dossier d'installation de Java est présent. Le chemin peut ressembler à celui de l'exemple suivant.

```
C:\\Program Files\\Amazon Corretto\\jdk11.0.13_8\\bin
```
 - g. Si le bin dossier de l'installation Java est absent de Path, choisissez Nouveau pour l'ajouter, puis OK.
3. Ouvrez l'invite de commande Windows (cmd.exe) en tant qu'administrateur.
4. Créez l'utilisateur par défaut dans le LocalSystem compte sur l'appareil Windows.
password Remplacez-le par un mot de passe sécurisé.

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

Tip

En fonction de votre configuration Windows, le mot de passe de l'utilisateur peut être configuré pour expirer à une date ultérieure. Pour vous assurer que vos applications Greengrass continuent de fonctionner, suivez la date d'expiration du mot de passe et mettez-le à jour avant son expiration. Vous pouvez également définir le mot de passe de l'utilisateur pour qu'il n'expire jamais.

- Pour vérifier la date d'expiration d'un utilisateur et de son mot de passe, exécutez la commande suivante.

```
net user ggc_user | findstr /C:expires
```

- Pour définir le mot de passe d'un utilisateur de manière à ce qu'il n'expire jamais, exécutez la commande suivante.

```
wmic UserAccount where "Name='ggc_user'" set PasswordExpires=False
```

- Si vous utilisez Windows 10 ou une version ultérieure où la [wmiccommande est obsolète](#), exécutez la commande suivante PowerShell .

```
Get-CimInstance -Query "SELECT * from Win32_UserAccount WHERE name = 'ggc_user'" | Set-CimInstance -Property @{PasswordExpires="False"}
```

5. Téléchargez et installez l'[PsExecutilitaire](#) de Microsoft sur l'appareil.
6. Utilisez l' PsExec utilitaire pour stocker le nom d'utilisateur et le mot de passe de l'utilisateur par défaut dans l'instance Credential Manager du LocalSystem compte. *password* Remplacez-le par le mot de passe utilisateur que vous avez défini précédemment.

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

S'il PsExec License Agreements'ouvre, choisissez Acceptd'accepter la licence et exécutez la commande.

 Note

Sur les appareils Windows, le LocalSystem compte exécute le noyau Greengrass, et vous devez utiliser l' PsExec utilitaire pour stocker les informations utilisateur par défaut dans le LocalSystem compte. L'application Credential Manager stocke ces informations dans le compte Windows de l'utilisateur actuellement connecté, plutôt que dans le LocalSystem compte.

Téléchargez le logiciel AWS IoT Greengrass de base

Vous pouvez télécharger la dernière version du logiciel AWS IoT Greengrass Core à l'adresse suivante :

- <https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest.zip>

 Note

Vous pouvez télécharger une version spécifique du logiciel AWS IoT Greengrass Core à l'emplacement suivant. Remplacez *version* par la version à télécharger.

```
https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-version.zip
```

Pour télécharger le logiciel AWS IoT Greengrass Core

1. Sur votre appareil principal, téléchargez le logiciel AWS IoT Greengrass Core dans un fichier nommé `greengrass-nucleus-latest.zip`.

Linux or Unix

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

Windows Command Prompt (CMD)

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

PowerShell

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

Si vous téléchargez ce logiciel, vous acceptez le [contrat de licence du logiciel Greengrass Core](#).

2. (Facultatif) Pour vérifier la signature du logiciel Greengrass Nucleus

Note

Cette fonctionnalité est disponible avec Greengrass nucleus version 2.9.5 et versions ultérieures.

- a. Utilisez la commande suivante pour vérifier la signature de votre artefact Greengrass nucleus :

Linux or Unix

```
jarsigner -verify -certs -verbose greengrass-nucleus-latest.zip
```

Windows Command Prompt (CMD)

Le nom du fichier peut être différent selon la version du JDK que vous installez.
jdk17.0.6_10 Remplacez-le par la version du JDK que vous avez installée.

```
"C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe" -verify -certs -verbose greengrass-nucleus-latest.zip
```


PowerShell

Le nom du fichier peut être différent selon la version du JDK que vous installez.
jdk17.0.6_10 Remplacez-le par la version du JDK que vous avez installée.

```
'C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe' -  
verify -certs -verbose greengrass-nucleus-latest.zip
```

- b. L'`jarsigner` invocation produit une sortie qui indique les résultats de la vérification.
 - i. Si le fichier zip Greengrass nucleus est signé, le résultat contient l'instruction suivante :

```
jar verified.
```

- ii. Si le fichier zip Greengrass nucleus n'est pas signé, le résultat contient l'instruction suivante :

```
jar is unsigned.
```

- c. Si vous avez fourni l'`-certs` option Jarsigner en même temps que les `-verbose` options `-verify` et, le résultat inclut également des informations détaillées sur le certificat du signataire.
- 3. Décompressez le logiciel AWS IoT Greengrass Core dans un dossier de votre appareil.
GreengrassInstaller Remplacez-le par le dossier que vous souhaitez utiliser.

Linux or Unix

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

Windows Command Prompt (CMD)

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

PowerShell

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

```
rm greengrass-nucleus-latest.zip
```

4. (Facultatif) Exécutez la commande suivante pour voir la version du logiciel AWS IoT Greengrass Core.

```
java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
```

Important

Si vous installez une version du noyau Greengrass antérieure à la version 2.4.0, ne supprimez pas ce dossier après avoir installé le logiciel Core. AWS IoT Greengrass Le logiciel AWS IoT Greengrass Core utilise les fichiers de ce dossier pour s'exécuter.

Si vous avez téléchargé la dernière version du logiciel, vous devez installer la version 2.4.0 ou ultérieure, et vous pouvez supprimer ce dossier après avoir installé le logiciel AWS IoT Greengrass Core.

Installation du logiciel AWS IoT Greengrass Core

Exécutez le programme d'installation avec des arguments qui spécifient les actions suivantes :

- Effectuez l'installation à partir d'un fichier de configuration partiel qui indique d'utiliser les AWS ressources et les certificats que vous avez créés précédemment. Le logiciel AWS IoT Greengrass Core utilise un fichier de configuration qui spécifie la configuration de chaque composant Greengrass de l'appareil. Le programme d'installation crée un fichier de configuration complet à partir du fichier de configuration partiel que vous fournissez.
- Spécifiez l'utilisation de l'utilisateur `ggc_user` du système pour exécuter les composants logiciels sur le périphérique principal. Sur les appareils Linux, cette commande indique également d'utiliser le groupe `ggc_group` système, et le programme d'installation crée l'utilisateur et le groupe système pour vous.
- Configurez le logiciel AWS IoT Greengrass Core en tant que service système qui s'exécute au démarrage. Sur les appareils Linux, cela nécessite le [système d'initialisation Systemd](#).

 Important

Sur les appareils Windows Core, vous devez configurer le logiciel AWS IoT Greengrass Core en tant que service système.

Pour plus d'informations sur les arguments que vous pouvez spécifier, consultez [Arguments d'installation](#).

 Note

Si vous utilisez AWS IoT Greengrass un appareil dont la mémoire est limitée, vous pouvez contrôler la quantité de mémoire utilisée par le logiciel AWS IoT Greengrass Core. Pour contrôler l'allocation de mémoire, vous pouvez définir les options de taille de segment de mémoire JVM dans le paramètre de `jvmOptions` configuration de votre composant Nucleus. Pour de plus amples informations, veuillez consulter [Contrôlez l'allocation de mémoire grâce aux options JVM](#).

- Si vous avez créé le certificat d'objet et la clé privée dans le AWS IoT service précédemment, suivez les étapes pour installer le logiciel AWS IoT Greengrass Core avec les fichiers de clé privée et de certificat.
- Si vous avez créé le certificat d'objet à partir d'une clé privée dans un module de sécurité matériel (HSM) plus tôt, suivez les étapes pour installer le logiciel AWS IoT Greengrass Core avec la clé privée et le certificat dans un HSM.

Installez le logiciel AWS IoT Greengrass Core avec les fichiers de clé privée et de certificat

Pour installer le logiciel AWS IoT Greengrass Core

1. Vérifiez la version du logiciel AWS IoT Greengrass Core.
 - *GreengrassInstaller* Remplacez-le par le chemin d'accès au dossier contenant le logiciel.

```
java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
```

2. Utilisez un éditeur de texte pour créer un fichier de configuration nommé `config.yaml` à fournir au programme d'installation.

Par exemple, sur un système basé sur Linux, vous pouvez exécuter la commande suivante pour utiliser GNU nano pour créer le fichier.

```
nano GreengrassInstaller/config.yaml
```

Copiez le contenu YAML suivant dans le fichier. Ce fichier de configuration partiel spécifie les paramètres du système et les paramètres du noyau Greengrass.

```
---
system:
  certificateFilePath: "/greengrass/v2/device.pem.crt"
  privateKeyPath: "/greengrass/v2/private.pem.key"
  rootCaPath: "/greengrass/v2/AmazonRootCA1.pem"
  rootpath: "/greengrass/v2"
  thingName: "MyGreengrassCore"
services:
  aws.greengrass.Nucleus:
    componentType: "NUCLEUS"
    version: "2.16.1"
    configuration:
      awsRegion: "us-west-2"
      iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
      fipsMode: "true"
      iotDataEndpoint: "data.iot-fips.us-west-2.amazonaws.com"
      greengrassDataPlaneEndpoint: "iotData"
      iotCredEndpoint: "data.credentials.iot-fips.us-west-2.amazonaws.com"
```

Ensuite, procédez comme suit :

- Remplacez chaque instance de */greengrass/v2* par le dossier racine de Greengrass.
- Remplacez *MyGreengrassCore* par le nom de l' AWS IoT objet.
- *2.16.1* Remplacez-le par la version du logiciel AWS IoT Greengrass Core.
- Remplacez *us-west-2* par l' Région AWS endroit où vous avez créé les ressources.
- Remplacez *GreengrassCoreTokenExchangeRoleAlias* par le nom de l'alias du rôle d'échange de jetons.
- Remplacez le *iotDataEndpoint* par votre point de terminaison de AWS IoT données.

- Remplacez le point de terminaison *iotCredEndpoint* par vos AWS IoT informations d'identification.
3. Exécutez le programme d'installation et spécifiez `--init-config` de fournir le fichier de configuration.
- Remplacez */greengrass/v2* ou *C:\greengrass\v2* par le dossier racine de Greengrass.
 - Remplacez chaque instance de *GreengrassInstaller* par le dossier dans lequel vous avez décompressé le programme d'installation.

Linux or Unix

```
sudo -E java -Droot="/greengrass/v2" -Dlog.store=FILE \  
-jar ./GreengrassInstaller/lib/Greengrass.jar \  
--init-config ./GreengrassInstaller/config.yaml \  
--component-default-user ggc_user:ggc_group \  
--setup-system-service true
```

Windows Command Prompt (CMD)

```
java -Droot="C:\greengrass\v2" "-Dlog.store=FILE" ^  
-jar ./GreengrassInstaller/lib/Greengrass.jar ^  
--init-config ./GreengrassInstaller/config.yaml ^  
--component-default-user ggc_user ^  
--setup-system-service true
```

PowerShell

```
java -Droot="C:\greengrass\v2" "-Dlog.store=FILE" `\  
-jar ./GreengrassInstaller/lib/Greengrass.jar `\  
--init-config ./GreengrassInstaller/config.yaml `\  
--component-default-user ggc_user `\  
--setup-system-service true
```

Important

Sur les appareils Windows Core, vous `--setup-system-service true` devez spécifier de configurer le logiciel AWS IoT Greengrass Core en tant que service système.

Si vous le spécifiez `--setup-system-service true`, le programme d'installation affiche `Successfully set up Nucleus as a system service` s'il a configuré et exécuté le logiciel en tant que service système. Dans le cas contraire, le programme d'installation n'affiche aucun message s'il installe le logiciel avec succès.

 Note

Vous ne pouvez pas utiliser l'`deploy-dev-tools` argument pour déployer des outils de développement locaux lorsque vous exécutez le programme d'installation sans l'`--provision true` argument. Pour plus d'informations sur le déploiement de la CLI Greengrass directement sur votre appareil, consultez. [Interface de ligne de commande Greengrass](#)

4. Vérifiez l'installation en consultant les fichiers du dossier racine.

Linux or Unix

```
ls /greengrass/v2
```

Windows Command Prompt (CMD)

```
dir C:\greengrass\v2
```

PowerShell

```
ls C:\greengrass\v2
```

Si l'installation a réussi, le dossier racine contient plusieurs dossiers, tels que `configpackages`, `etlogs`.

Installez le logiciel AWS IoT Greengrass principal avec la clé privée et le certificat dans un HSM

 Note

Cette fonctionnalité est disponible pour les versions 2.5.3 et ultérieures du composant [Greengrass](#) nucleus. AWS IoT Greengrass ne prend actuellement pas en charge cette fonctionnalité sur les appareils Windows principaux.

Pour installer le logiciel AWS IoT Greengrass Core

1. Vérifiez la version du logiciel AWS IoT Greengrass Core.
 - *GreengrassInstaller* Remplacez-le par le chemin d'accès au dossier contenant le logiciel.

```
java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
```

2. Pour permettre au logiciel AWS IoT Greengrass Core d'utiliser la clé privée et le certificat du HSM, installez le [composant fournisseur PKCS #11](#) lors de l'installation du logiciel AWS IoT Greengrass Core. Le composant fournisseur PKCS #11 est un plugin que vous pouvez configurer lors de l'installation. Vous pouvez télécharger la dernière version du composant fournisseur PKCS #11 depuis l'emplacement suivant :

- <https://d2s8p88vqu9w66.cloudfront.net/releases/Pkcs11Provider/aws.greengrass.crypto.pkcs11Provider-Latest.jar>

Téléchargez le plugin du fournisseur PKCS #11 dans un fichier nommé `aws.greengrass.crypto.Pkcs11Provider.jar`.

GreengrassInstaller Remplacez-le par le dossier que vous souhaitez utiliser.

```
curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/Pkcs11Provider/
aws.greengrass.crypto.Pkcs11Provider-latest.jar > GreengrassInstaller/
aws.greengrass.crypto.Pkcs11Provider.jar
```

Si vous téléchargez ce logiciel, vous acceptez le [contrat de licence du logiciel Greengrass Core](#).

3. Utilisez un éditeur de texte pour créer un fichier de configuration nommé `config.yaml` à fournir au programme d'installation.

Par exemple, sur un système basé sur Linux, vous pouvez exécuter la commande suivante pour utiliser GNU nano pour créer le fichier.

```
nano GreengrassInstaller/config.yaml
```

Copiez le contenu YAML suivant dans le fichier. Ce fichier de configuration partiel spécifie les paramètres système, les paramètres du noyau Greengrass et les paramètres du fournisseur PKCS #11.

```
---
system:
  certificateFilePath: "/greengrass/v2/device.pem.crt"
  privateKeyPath: "/greengrass/v2/private.pem.key"
  rootCaPath: "/greengrass/v2/AmazonRootCA1.pem"
  rootpath: "/greengrass/v2"
  thingName: "MyGreengrassCore"
services:
  aws.greengrass.Nucleus:
    componentType: "NUCLEUS"
    version: "2.16.1"
    configuration:
      awsRegion: "us-west-2"
      iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
      fipsMode: "true"
      iotDataEndpoint: "data.iot-fips.us-west-2.amazonaws.com"
      greengrassDataPlaneEndpoint: "iotData"
      iotCredEndpoint: "data.credentials.iot-fips.us-west-2.amazonaws.com"
```

Ensuite, procédez comme suit :

- Remplacez chaque instance du PKCS #11 par le libellé URIs de l'objet *iotdevicekey* dans lequel vous avez créé la clé privée et importé le certificat.
- Remplacez chaque instance de */greengrass/v2* par le dossier racine de Greengrass.
- Remplacez *MyGreengrassCore* par le nom de l' AWS IoT objet.
- *2.16.1* Remplacez-le par la version du logiciel AWS IoT Greengrass Core.
- Remplacez *us-west-2* par l' Région AWS endroit où vous avez créé les ressources.
- Remplacez *GreengrassCoreTokenExchangeRoleAlias* par le nom de l'alias du rôle d'échange de jetons.

- Remplacez le `iotDataEndpoint` par votre point de terminaison de AWS IoT données.
 - Remplacez le point de terminaison `iotCredEndpoint` par vos AWS IoT informations d'identification.
 - Remplacez les paramètres de configuration du `aws.greengrass.crypto.Pkcs11Provider` composant par les valeurs de la configuration HSM sur le périphérique principal.
4. Exécutez le programme d'installation et spécifiez `--init-config` de fournir le fichier de configuration.
- `/greengrass/v2` Remplacez-le par le dossier racine de Greengrass.
 - Remplacez chaque instance de `GreengrassInstaller` par le dossier dans lequel vous avez décompressé le programme d'installation.

```
sudo -E java -Droot="/greengrass/v2" -Dlog.store=FILE \  
-jar ./GreengrassInstaller/lib/Greengrass.jar \  
--trusted-plugin ./GreengrassInstaller/aws.greengrass.crypto.Pkcs11Provider.jar \  
--init-config ./GreengrassInstaller/config.yaml \  
--component-default-user ggc_user:ggc_group \  
--setup-system-service true
```

Important

Sur les appareils Windows Core, vous `--setup-system-service true` devez spécifier de configurer le logiciel AWS IoT Greengrass Core en tant que service système.

Si vous le spécifiez `--setup-system-service true`, le programme d'installation affiche `Successfully set up Nucleus as a system service` s'il a configuré et exécuté le logiciel en tant que service système. Dans le cas contraire, le programme d'installation n'affiche aucun message s'il installe le logiciel avec succès.

Note

Vous ne pouvez pas utiliser l'`deploy-dev-tools` argument pour déployer des outils de développement locaux lorsque vous exécutez le programme d'installation sans l'`--provision true` argument. Pour plus d'informations sur le déploiement de la CLI

Greengrass directement sur votre appareil, consultez. [Interface de ligne de commande Greengrass](#)

5. Vérifiez l'installation en consultant les fichiers du dossier racine.

Linux or Unix

```
ls /greengrass/v2
```

Windows Command Prompt (CMD)

```
dir C:\greengrass\v2
```

PowerShell

```
ls C:\greengrass\v2
```

Si l'installation a réussi, le dossier racine contient plusieurs dossiers, tels que configpackages, etlogs.

Si vous avez installé le logiciel AWS IoT Greengrass Core en tant que service système, le programme d'installation exécute le logiciel pour vous. Dans le cas contraire, vous devez exécuter le logiciel manuellement. Pour de plus amples informations, veuillez consulter [Exécutez le logiciel AWS IoT Greengrass Core](#).

Pour plus d'informations sur la configuration et l'utilisation du logiciel AWS IoT Greengrass, consultez les rubriques suivantes :

- [Configuration du logiciel AWS IoT Greengrass de base](#)
- [Développer des AWS IoT Greengrass composants](#)
- [Déployer AWS IoT Greengrass des composants sur des appareils](#)
- [Interface de ligne de commande Greengrass](#)

Installation de points de terminaison FIPS avec provisionnement de flotte

Cette fonctionnalité est disponible pour les versions 2.4.0 et ultérieures du composant [Greengrass](#) nucleus.

Installez des points de terminaison FIPS sur votre logiciel AWS IoT Greengrass principal en provisionnant un AWS IoT parc pour vos appareils principaux.

Note

Le plug-in de provisionnement de flotte ne prend actuellement pas en charge le stockage de clés privées et de fichiers de certificats dans un module de sécurité matérielle (HSM). Pour utiliser un HSM, [installez le logiciel AWS IoT Greengrass Core avec un provisionnement manuel](#).

Pour installer le logiciel AWS IoT Greengrass Core avec provisionnement de AWS IoT flotte, vous devez configurer les ressources utilisées pour approvisionner Compte AWS les AWS IoT appareils principaux de Greengrass. Ces ressources incluent un modèle de provisionnement, des certificats de réclamation et un rôle [IAM d'échange de jetons](#). Après avoir créé ces ressources, vous pouvez les réutiliser pour approvisionner plusieurs appareils principaux dans un parc. Pour de plus amples informations, veuillez consulter [Configurer le provisionnement du AWS IoT parc pour les appareils principaux de Greengrass](#).

Important

Avant de télécharger le logiciel AWS IoT Greengrass Core, vérifiez que votre appareil principal répond à la [configuration requise](#) pour installer et exécuter le logiciel AWS IoT Greengrass Core v2.0.

Rubriques

- [Conditions préalables](#)
- [Récupérer des points de AWS IoT terminaison](#)
- [Télécharger les certificats sur l'appareil](#)
- [Configuration de l'environnement de l'appareil](#)
- [Téléchargez le logiciel AWS IoT Greengrass de base](#)

- [Téléchargez le plugin de provisionnement de AWS IoT flotte](#)
- [Installer le logiciel AWS IoT Greengrass Core](#)

Conditions préalables

Pour installer le logiciel AWS IoT Greengrass Core avec le provisionnement du AWS IoT parc, vous devez d'abord [configurer le provisionnement du AWS IoT parc pour les appareils principaux de Greengrass](#). Après avoir effectué ces étapes une fois, vous pouvez utiliser le provisionnement du parc pour installer le logiciel AWS IoT Greengrass Core sur autant d'appareils que vous le souhaitez.

Récupérer des points de AWS IoT terminaison

Procurez-vous les points de terminaison FIPS qui vous Compte AWS conviennent et enregistrez-les pour les utiliser ultérieurement. Votre appareil utilise ces points de terminaison pour se connecter à AWS IoT. Procédez comme suit :

1. Obtenez le point de terminaison de données FIPS de votre région dans les points de terminaison du [plan de données AWS IoT Core FIPS](#). Le point de terminaison de données FIPS pour vous Compte AWS devrait ressembler à ceci : *data.iot-fips.us-west-2.amazonaws.com*
2. Obtenez le point de terminaison des informations d'identification FIPS de votre région dans les points de terminaison du [plan de données AWS IoT Core FIPS](#). Le point de terminaison de vos informations d'identification FIPS Compte AWS devrait ressembler à ceci : *data.credentials.iot-fips.us-west-2.amazonaws.com*

Télécharger les certificats sur l'appareil

L'appareil utilise un certificat de réclamation et une clé privée pour authentifier sa demande de fourniture de AWS ressources et acquérir un certificat de périphérique X.509. Vous pouvez intégrer le certificat de réclamation et la clé privée dans l'appareil pendant la fabrication, ou copier le certificat et la clé sur l'appareil lors de l'installation. Dans cette section, vous devez copier le certificat de réclamation et la clé privée sur l'appareil. Vous téléchargez également le certificat Amazon Root Certificate Authority (CA) sur l'appareil.

Important

L'approvisionnement des clés privées des réclamations doit être sécurisé à tout moment, y compris sur les appareils principaux de Greengrass. Nous vous recommandons d'utiliser

CloudWatch les statistiques et les journaux Amazon pour détecter les signes d'utilisation abusive, tels que l'utilisation non autorisée du certificat de réclamation pour approvisionner des appareils. Si vous détectez une utilisation abusive, désactivez le certificat de demande d'approvisionnement afin qu'il ne puisse pas être utilisé pour le provisionnement des appareils. Pour plus d'informations, consultez la section [Surveillance AWS IoT](#) dans le guide du AWS IoT Core développeur.

Pour vous aider à mieux gérer le nombre d'appareils et les appareils qui s'enregistrent dans le votre Compte AWS, vous pouvez spécifier un crochet de préapprovisionnement lorsque vous créez un modèle de provisionnement de flotte. Un hook de pré-provisionnement est une AWS Lambda fonction qui valide les paramètres du modèle fournis par les appareils lors de l'enregistrement. Par exemple, vous pouvez créer un hook de pré-provisionnement qui compare l'identifiant d'un appareil à une base de données afin de vérifier que l'appareil est autorisé à le provisionner. Pour plus d'informations, consultez la section [Pre-provisioning hooks](#) dans le Guide du AWS IoT Core développeur.

Pour télécharger les certificats de réclamation sur l'appareil

1. Copiez le certificat de réclamation et la clé privée sur l'appareil. Si SSH et SCP sont activés sur l'ordinateur de développement et le périphérique, vous pouvez utiliser la `scp` commande sur votre ordinateur de développement pour transférer le certificat de réclamation et la clé privée. L'exemple de commande suivant transfère ces fichiers vers le périphérique dans un dossier nommé `claim-certs` sur votre ordinateur de développement. Remplacez *device-ip-address* par l'adresse IP de votre appareil.

```
scp -r claim-certs/ device-ip-address:~
```

2. Créez le dossier racine Greengrass sur l'appareil. Vous installerez ultérieurement le logiciel AWS IoT Greengrass Core dans ce dossier.

Note

Pour Windows, la limitation de la longueur du chemin est de 260 caractères. Si vous utilisez Windows, utilisez un dossier racine tel que `C:\greengrass\v2` ou `D:\greengrass\v2` pour conserver les chemins des composants Greengrass en dessous de la limite de 260 caractères.

Linux or Unix

- `/greengrass/v2` Remplacez-le par le dossier à utiliser.

```
sudo mkdir -p /greengrass/v2
```

Windows Command Prompt

- `C:\greengrass\v2` Remplacez-le par le dossier à utiliser.

```
mkdir C:\greengrass\v2
```

PowerShell

- `C:\greengrass\v2` Remplacez-le par le dossier à utiliser.

```
mkdir C:\greengrass\v2
```

3. (Linux uniquement) Définissez les autorisations du parent du dossier racine de Greengrass.

- Remplacez `/greengrass` par le parent du dossier racine.

```
sudo chmod 755 /greengrass
```

4. Déplacez les certificats de réclamation vers le dossier racine de Greengrass.

- Remplacez `/greengrass/v2` ou `C:\greengrass\v2` par le dossier racine de Greengrass.

Linux or Unix

```
sudo mv ~/claim-certs /greengrass/v2
```

Windows Command Prompt (CMD)

```
move %USERPROFILE%\claim-certs C:\greengrass\v2
```

PowerShell

```
mv -Path ~\claim-certs -Destination C:\greengrass\v2
```

5. Téléchargez à la fois le CA1 certificat et le [CA3certificat](#).

Linux or Unix

```
sudo curl -o - https://www.amazontrust.com/repository/AmazonRootCA3.pem >> /  
greengrass/v2/AmazonRootCA1.pem
```

Windows Command Prompt (CMD)

```
curl -o C:\greengrass\v2\AmazonRootCA1.pem https://www.amazontrust.com/  
repository/AmazonRootCA1.pem
```

PowerShell

```
iwr -Uri https://www.amazontrust.com/repository/AmazonRootCA1.pem -OutFile C:  
\greengrass\v2\AmazonRootCA1.pem
```

Configuration de l'environnement de l'appareil

Suivez les étapes décrites dans cette section pour configurer un appareil Linux ou Windows à utiliser comme périphérique AWS IoT Greengrass principal.

Configuration d'un appareil Linux

Pour configurer un appareil Linux pour AWS IoT Greengrass V2

1. Installez le moteur d'exécution Java, dont le logiciel AWS IoT Greengrass Core a besoin pour fonctionner. Nous vous recommandons d'utiliser les versions de support à long terme d'[Amazon Corretto](#) ou d'OpenJDK. La version 8 ou supérieure est requise. Les commandes suivantes vous montrent comment installer OpenJDK sur votre appareil.

- Pour les distributions basées sur Debian ou Ubuntu :

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

- Pour les distributions basées sur Red Hat :

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

- Dans Amazon Linux 2 :

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

- Pour Amazon Linux 2023 :

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

Lorsque l'installation est terminée, exécutez la commande suivante pour vérifier que Java s'exécute sur votre appareil Linux.

```
java -version
```

La commande affiche la version de Java exécutée sur le périphérique. Par exemple, sur une distribution basée sur Debian, le résultat peut ressembler à celui de l'exemple suivant.

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

2. (Facultatif) Créez l'utilisateur système et le groupe par défaut qui exécutent les composants sur le périphérique. Vous pouvez également choisir de laisser le programme d'installation du logiciel AWS IoT Greengrass Core créer cet utilisateur et ce groupe lors de l'installation avec l'argument `--component-default-user` installer. Pour de plus amples informations, veuillez consulter [Arguments d'installation](#).

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


3. Vérifiez que l'utilisateur qui exécute le logiciel AWS IoT Greengrass Core (généralement `root`) est autorisé à exécuter `sudo` avec n'importe quel utilisateur et n'importe quel groupe.

- a. Exécutez la commande suivante pour ouvrir le `/etc/sudoers` fichier.

```
sudo visudo
```

- b. Vérifiez que l'autorisation accordée à l'utilisateur ressemble à l'exemple suivant.

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

4. (Facultatif) Pour [exécuter des fonctions Lambda conteneurisées](#), vous devez activer [cgroups v1](#), et vous devez activer et monter les cgroups de mémoire et de périphériques. Si vous ne prévoyez pas d'exécuter des fonctions Lambda conteneurisées, vous pouvez ignorer cette étape.

Pour activer ces options cgroups, démarrez le périphérique avec les paramètres du noyau Linux suivants.

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

Pour plus d'informations sur l'affichage et la définition des paramètres du noyau de votre appareil, consultez la documentation de votre système d'exploitation et de votre chargeur de démarrage. Suivez les instructions pour définir définitivement les paramètres du noyau.

5. Installez toutes les autres dépendances requises sur votre appareil, comme indiqué dans la liste des exigences figurant dans [Exigences relatives aux dispositifs](#).

Configuration d'un appareil Windows

Note

Cette fonctionnalité est disponible pour les versions 2.5.0 et ultérieures du composant [Greengrass](#) nucleus.

Pour configurer un appareil Windows pour AWS IoT Greengrass V2

1. Installez le moteur d'exécution Java, dont le logiciel AWS IoT Greengrass Core a besoin pour fonctionner. Nous vous recommandons d'utiliser les versions de support à long terme d'[Amazon Corretto](#) ou d'OpenJDK. La version 8 ou supérieure est requise.

2. Vérifiez si Java est disponible sur la variable système [PATH](#), et ajoutez-le dans le cas contraire. Le LocalSystem compte exécute le logiciel AWS IoT Greengrass Core. Vous devez donc ajouter Java à la variable système PATH au lieu de la variable utilisateur PATH pour votre utilisateur. Procédez comme suit :
 - a. Appuyez sur la touche Windows pour ouvrir le menu de démarrage.
 - b. Tapez **environment variables** pour rechercher les options du système dans le menu Démarrer.
 - c. Dans les résultats de recherche du menu Démarrer, choisissez Modifier les variables d'environnement du système pour ouvrir la fenêtre des propriétés du système.
 - d. Choisissez les variables d'environnement... pour ouvrir la fenêtre des variables d'environnement.
 - e. Sous Variables système, sélectionnez Chemin, puis Modifier. Dans la fenêtre Modifier la variable d'environnement, vous pouvez afficher chaque chemin sur une ligne distincte.
 - f. Vérifiez si le chemin d'accès au bin dossier d'installation de Java est présent. Le chemin peut ressembler à celui de l'exemple suivant.

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

- g. Si le bin dossier de l'installation Java est absent de Path, choisissez Nouveau pour l'ajouter, puis OK.
3. Ouvrez l'invite de commande Windows (cmd.exe) en tant qu'administrateur.
4. Créez l'utilisateur par défaut dans le LocalSystem compte sur l'appareil Windows.
password Remplacez-le par un mot de passe sécurisé.

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

Tip

En fonction de votre configuration Windows, le mot de passe de l'utilisateur peut être configuré pour expirer à une date ultérieure. Pour vous assurer que vos applications Greengrass continuent de fonctionner, suivez la date d'expiration du mot de passe et mettez-le à jour avant son expiration. Vous pouvez également définir le mot de passe de l'utilisateur pour qu'il n'expire jamais.

- Pour vérifier la date d'expiration d'un utilisateur et de son mot de passe, exécutez la commande suivante.

```
net user ggc_user | findstr /C:expires
```

- Pour définir le mot de passe d'un utilisateur de manière à ce qu'il n'expire jamais, exécutez la commande suivante.

```
wmic UserAccount where "Name='ggc_user'" set PasswordExpires=False
```

- Si vous utilisez Windows 10 ou une version ultérieure où la [wmic commande est obsolète](#), exécutez la commande suivante PowerShell .

```
Get-CimInstance -Query "SELECT * from Win32_UserAccount WHERE name = 'ggc_user'" | Set-CimInstance -Property @{PasswordExpires="False"}
```

5. Téléchargez et installez l'[PsExec utilitaire](#) de Microsoft sur l'appareil.
6. Utilisez l' PsExec utilitaire pour stocker le nom d'utilisateur et le mot de passe de l'utilisateur par défaut dans l'instance Credential Manager du LocalSystem compte. *password* Remplacez-le par le mot de passe utilisateur que vous avez défini précédemment.

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

S'il PsExec License Agreements'ouvre, choisissez Acceptd'accepter la licence et exécutez la commande.

Note

Sur les appareils Windows, le LocalSystem compte exécute le noyau Greengrass, et vous devez utiliser l' PsExec utilitaire pour stocker les informations utilisateur par défaut dans le LocalSystem compte. L'application Credential Manager stocke ces informations dans le compte Windows de l'utilisateur actuellement connecté, plutôt que dans le LocalSystem compte.

Téléchargez le logiciel AWS IoT Greengrass de base

Vous pouvez télécharger la dernière version du logiciel AWS IoT Greengrass Core à l'adresse suivante :

- <https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest.zip>

Note

Vous pouvez télécharger une version spécifique du logiciel AWS IoT Greengrass Core à l'emplacement suivant. Remplacez *version* par la version à télécharger.

```
https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-version.zip
```

Pour télécharger le logiciel AWS IoT Greengrass Core

1. Sur votre appareil principal, téléchargez le logiciel AWS IoT Greengrass Core dans un fichier nommé `greengrass-nucleus-latest.zip`.

Linux or Unix

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

Windows Command Prompt (CMD)

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

PowerShell

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

Si vous téléchargez ce logiciel, vous acceptez le [contrat de licence du logiciel Greengrass Core](#).

2. (Facultatif) Pour vérifier la signature du logiciel Greengrass Nucleus

 Note

Cette fonctionnalité est disponible avec Greengrass nucleus version 2.9.5 et versions ultérieures.

- a. Utilisez la commande suivante pour vérifier la signature de votre artefact Greengrass nucleus :

Linux or Unix

```
jarsigner -verify -certs -verbose greengrass-nucleus-latest.zip
```

Windows Command Prompt (CMD)

Le nom du fichier peut être différent selon la version du JDK que vous installez.
jdk17.0.6_10 Remplacez-le par la version du JDK que vous avez installée.

```
"C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe" -  
verify -certs -verbose greengrass-nucleus-latest.zip
```

PowerShell

Le nom du fichier peut être différent selon la version du JDK que vous installez.
jdk17.0.6_10 Remplacez-le par la version du JDK que vous avez installée.

```
'C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe' -  
verify -certs -verbose greengrass-nucleus-latest.zip
```

- b. L'`jarsigner` invocation produit une sortie qui indique les résultats de la vérification.
 - i. Si le fichier zip Greengrass nucleus est signé, le résultat contient l'instruction suivante :

```
jar verified.
```

- ii. Si le fichier zip Greengrass nucleus n'est pas signé, le résultat contient l'instruction suivante :

```
jar is unsigned.
```

- c. Si vous avez fourni l'-certsoption Jarsigner en même temps que les -verbose options -verify et, le résultat inclut également des informations détaillées sur le certificat du signataire.
3. Décompressez le logiciel AWS IoT Greengrass Core dans un dossier de votre appareil. **GreengrassInstaller** Remplacez-le par le dossier que vous souhaitez utiliser.

Linux or Unix

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

Windows Command Prompt (CMD)

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

PowerShell

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

4. (Facultatif) Exécutez la commande suivante pour voir la version du logiciel AWS IoT Greengrass Core.

```
java -jar ./ GreengrassInstaller/lib/Greengrass.jar --version
```

Important

Si vous installez une version du noyau Greengrass antérieure à la version 2.4.0, ne supprimez pas ce dossier après avoir installé le logiciel Core. AWS IoT Greengrass Le logiciel AWS IoT Greengrass Core utilise les fichiers de ce dossier pour s'exécuter.

Si vous avez téléchargé la dernière version du logiciel, vous devez installer la version 2.4.0 ou ultérieure, et vous pouvez supprimer ce dossier après avoir installé le logiciel AWS IoT Greengrass Core.

Téléchargez le plugin de provisionnement de AWS IoT flotte

Vous pouvez télécharger la dernière version du plugin de provisionnement de AWS IoT flotte à l'adresse suivante :

- <https://d2s8p88vqu9w66.cloudfront.net/releases/aws-greengrass-FleetProvisioningByClaim/fleetprovisioningbyclaim-latest.jar>

Note

Vous pouvez télécharger une version spécifique du plugin de provisionnement de AWS IoT flotte à l'emplacement suivant. Remplacez *version* par la version à télécharger. Pour plus d'informations sur chaque version du plugin de provisionnement de flotte, consultez [AWS IoT journal des modifications du plugin de provisionnement de flotte](#).

```
https://d2s8p88vqu9w66.cloudfront.net/releases/aws-greengrass-  
FleetProvisioningByClaim/fleetprovisioningbyclaim-version.jar
```

Le plugin de provisionnement de flotte est open source. Pour consulter son code source, consultez le [plugin de provisionnement de AWS IoT flotte](#) sur GitHub.

Pour télécharger le plugin de provisionnement de AWS IoT flotte

- Sur votre appareil, téléchargez le plug-in de provisionnement de AWS IoT flotte dans un fichier nommé `aws.greengrass.FleetProvisioningByClaim.jar`. *GreengrassInstaller* Remplacez-le par le dossier que vous souhaitez utiliser.

Linux or Unix

```
curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/aws-greengrass-  
FleetProvisioningByClaim/fleetprovisioningbyclaim-latest.jar  
> GreengrassInstaller/aws.greengrass.FleetProvisioningByClaim.jar
```

Windows Command Prompt (CMD)

```
curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/aws-greengrass-  
FleetProvisioningByClaim/fleetprovisioningbyclaim-latest.jar  
> GreengrassInstaller/aws.greengrass.FleetProvisioningByClaim.jar
```

PowerShell

```
iwr -Uri https://d2s8p88vqu9w66.cloudfront.net/releases/aws-greengrass-  
FleetProvisioningByClaim/fleetprovisioningbyclaim-latest.jar -  
OutFile GreengrassInstaller/aws.greengrass.FleetProvisioningByClaim.jar
```

Si vous téléchargez ce logiciel, vous acceptez le [contrat de licence du logiciel Greengrass Core](#).

Installer le logiciel AWS IoT Greengrass Core

Exécutez le programme d'installation avec des arguments qui spécifient les actions suivantes :

- Effectuez l'installation à partir d'un fichier de configuration partiel qui indique d'utiliser le plug-in de provisionnement de flotte pour provisionner les AWS ressources. Le logiciel AWS IoT Greengrass Core utilise un fichier de configuration qui spécifie la configuration de chaque composant Greengrass de l'appareil. Le programme d'installation crée un fichier de configuration complet à partir du fichier de configuration partiel que vous fournissez et AWS des ressources créées par le plug-in de provisionnement de flotte.
- Spécifiez l'utilisation de l'utilisateur `ggc_user` du système pour exécuter les composants logiciels sur le périphérique principal. Sur les appareils Linux, cette commande indique également d'utiliser le groupe `ggc_group` système, et le programme d'installation crée l'utilisateur et le groupe système pour vous.
- Configurez le logiciel AWS IoT Greengrass Core en tant que service système qui s'exécute au démarrage. Sur les appareils Linux, cela nécessite le [système d'initialisation Systemd](#).

Important

Sur les appareils Windows Core, vous devez configurer le logiciel AWS IoT Greengrass Core en tant que service système.

Pour plus d'informations sur les arguments que vous pouvez spécifier, consultez [Arguments d'installation](#).

 Note

Si vous utilisez AWS IoT Greengrass un appareil dont la mémoire est limitée, vous pouvez contrôler la quantité de mémoire utilisée par le logiciel AWS IoT Greengrass Core. Pour contrôler l'allocation de mémoire, vous pouvez définir les options de taille de segment de mémoire JVM dans le paramètre de `jvmOptions` configuration de votre composant Nucleus. Pour de plus amples informations, veuillez consulter [Contrôlez l'allocation de mémoire grâce aux options JVM](#).

Pour installer le logiciel AWS IoT Greengrass Core

1. Vérifiez la version du logiciel AWS IoT Greengrass Core.
 - *GreengrassInstaller* Remplacez-le par le chemin d'accès au dossier contenant le logiciel.

```
java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
```

2. Utilisez un éditeur de texte pour créer un fichier de configuration nommé `config.yaml` à fournir au programme d'installation.

Par exemple, sur un système basé sur Linux, vous pouvez exécuter la commande suivante pour utiliser GNU nano pour créer le fichier.

```
nano GreengrassInstaller/config.yaml
```

Copiez le contenu YAML suivant dans le fichier. Ce fichier de configuration partiel spécifie les paramètres du plugin de provisionnement de flotte. Pour plus d'informations sur les options que vous pouvez spécifier, consultez [Configuration du plugin de provisionnement de AWS IoT flotte](#).

Linux or Unix

```
---
services:
  aws.greengrass.Nucleus:
```

```

version: "2.16.1"
configuration:
  fipsMode: "true"
  greengrassDataPlaneEndpoint: "iotData"
aws.greengrass.FleetProvisioningByClaim:
  configuration:
    rootPath: "/greengrass/v2"
    awsRegion: "us-west-2"
    iotDataEndpoint: "data.iot-fips.us-west-2.amazonaws.com"
    iotCredEndpoint: "data.credentials.iot-fips.us-west-2.amazonaws.com"
    iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
    provisioningTemplate: "GreengrassFleetProvisioningTemplate"
    claimCertificatePath: "/greengrass/v2/claim-certs/claim.pem.crt"
    claimCertificatePrivateKeyPath: "/greengrass/v2/claim-certs/
claim.private.pem.key"
    rootCaPath: "/greengrass/v2/AmazonRootCA1.pem"
    templateParameters:
      ThingName: "MyGreengrassCore"
      ThingGroupName: "MyGreengrassCoreGroup"

```

Windows

```

---
services:
  aws.greengrass.Nucleus:
    version: "2.16.1"
  aws.greengrass.FleetProvisioningByClaim:
    configuration:
      rootPath: "C:\\greengrass\\v2"
      awsRegion: "us-west-2"
      iotDataEndpoint: "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
      iotCredentialEndpoint: "device-credentials-prefix.credentials.iot.us-
west-2.amazonaws.com"
      iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
      provisioningTemplate: "GreengrassFleetProvisioningTemplate"
      claimCertificatePath: "C:\\greengrass\\v2\\claim-certs\\claim.pem.crt"
      claimCertificatePrivateKeyPath: "C:\\greengrass\\v2\\claim-certs\\
\\claim.private.pem.key"
      rootCaPath: "C:\\greengrass\\v2\\AmazonRootCA1.pem"
      templateParameters:
        ThingName: "MyGreengrassCore"
        ThingGroupName: "MyGreengrassCoreGroup"

```

Ensuite, procédez comme suit :

- **2.16.1** Remplacez-le par la version du logiciel AWS IoT Greengrass Core.
- Remplacez chaque instance de `/greengrass/v2` ou `C:\greengrass\v2` par le dossier racine de Greengrass.

 Note

Sur les appareils Windows, vous devez spécifier les séparateurs de chemin sous forme de barres obliques inverses (\\) doubles, tels que. `C:\\greengrass\\v2`

- **us-west-2** Remplacez-le par la AWS région dans laquelle vous avez créé le modèle de provisionnement et les autres ressources.
 - Remplacez le `iotDataEndpoint` par votre point de terminaison de AWS IoT données.
 - Remplacez le point de terminaison `iotCredentialEndpoint` par vos AWS IoT informations d'identification.
 - Remplacez ***GreengrassCoreTokenExchangeRoleAlias*** par le nom de l'alias du rôle d'échange de jetons.
 - Remplacez ***GreengrassFleetProvisioningTemplate*** par le nom du modèle de provisionnement de flotte.
 - Remplacez le `claimCertificatePath` par le chemin d'accès au certificat de réclamation sur l'appareil.
 - Remplacez le `claimCertificatePrivateKeyPath` par le chemin d'accès à la clé privée du certificat de réclamation sur l'appareil.
 - Remplacez les paramètres du modèle (`templateParameters`) par les valeurs à utiliser pour approvisionner le périphérique. Cet exemple fait référence à l'[exemple de modèle](#) qui définit `ThingName` et définit `ThingGroupName` les paramètres.
3. Exécutez le programme d'installation. Spécifiez `--trusted-plugin` pour fournir le plug-in de provisionnement de flotte et spécifiez `--init-config` pour fournir le fichier de configuration.
- ***/greengrass/v2*** Remplacez-le par le dossier racine de Greengrass.
 - Remplacez chaque instance de ***GreengrassInstaller*** par le dossier dans lequel vous avez décompressé le programme d'installation.

Linux or Unix

```
sudo -E java -Droot="/greengrass/v2" -Dlog.store=FILE \  
-jar ./GreengrassInstaller/lib/Greengrass.jar \  
--trusted-plugin ./GreengrassInstaller/  
aws.greengrass.FleetProvisioningByClaim.jar \  
--init-config ./GreengrassInstaller/config.yaml \  
--component-default-user ggc_user:ggc_group \  
--setup-system-service true
```

Windows Command Prompt (CMD)

```
java -Droot="C:\greengrass\v2" "-Dlog.store=FILE" ^  
-jar ./GreengrassInstaller/lib/Greengrass.jar ^  
--trusted-plugin ./GreengrassInstaller/  
aws.greengrass.FleetProvisioningByClaim.jar ^  
--init-config ./GreengrassInstaller/config.yaml ^  
--component-default-user ggc_user ^  
--setup-system-service true
```

PowerShell

```
java -Droot="C:\greengrass\v2" "-Dlog.store=FILE" `  
-jar ./GreengrassInstaller/lib/Greengrass.jar `  
--trusted-plugin ./GreengrassInstaller/  
aws.greengrass.FleetProvisioningByClaim.jar `  
--init-config ./GreengrassInstaller/config.yaml `  
--component-default-user ggc_user `  
--setup-system-service true
```

Important

Sur les appareils Windows Core, vous devez spécifier de configurer le logiciel AWS IoT Greengrass Core en tant que service système.

Si vous le spécifiez `--setup-system-service true`, le programme d'installation affiche `Successfully set up Nucleus as a system service` s'il a configuré et exécuté le

logiciel en tant que service système. Dans le cas contraire, le programme d'installation n'affiche aucun message s'il installe le logiciel avec succès.

 Note

Vous ne pouvez pas utiliser l'`deploy-dev-tool` argument pour déployer des outils de développement locaux lorsque vous exécutez le programme d'installation sans l'`--provision true` argument. Pour plus d'informations sur le déploiement de la CLI Greengrass directement sur votre appareil, consultez. [Interface de ligne de commande Greengrass](#)

4. Vérifiez l'installation en consultant les fichiers du dossier racine.

Linux or Unix

```
ls /greengrass/v2
```

Windows Command Prompt (CMD)

```
dir C:\greengrass\v2
```

PowerShell

```
ls C:\greengrass\v2
```

Si l'installation a réussi, le dossier racine contient plusieurs dossiers, tels que `configpackages`, `etlogs`.

Si vous avez installé le logiciel AWS IoT Greengrass Core en tant que service système, le programme d'installation exécute le logiciel pour vous. Dans le cas contraire, vous devez exécuter le logiciel manuellement. Pour de plus amples informations, veuillez consulter [Exécutez le logiciel AWS IoT Greengrass Core](#).

Pour plus d'informations sur la configuration et l'utilisation du logiciel AWS IoT Greengrass, consultez les rubriques suivantes :

- [Configuration du logiciel AWS IoT Greengrass de base](#)

- [Développer des AWS IoT Greengrass composants](#)
- [Déployer AWS IoT Greengrass des composants sur des appareils](#)
- [Interface de ligne de commande Greengrass](#)

Installation de points de terminaison FIPS avec provisionnement automatique des ressources

Le logiciel AWS IoT Greengrass Core inclut un programme d'installation qui configure votre appareil en tant qu'appareil principal Greengrass. Pour configurer rapidement un appareil, le programme d'installation peut configurer l' AWS IoT AWS IoT objet, le groupe d'objets, le rôle IAM et l'alias de AWS IoT rôle dont le périphérique principal a besoin pour fonctionner. Le programme d'installation peut également déployer les outils de développement locaux sur le périphérique principal, afin que vous puissiez utiliser l'appareil pour développer et tester des composants logiciels personnalisés. Le programme d'installation a besoin AWS d'informations d'identification pour provisionner ces ressources et créer le déploiement.

Si vous ne pouvez pas fournir AWS d'informations d'identification à l'appareil, vous pouvez fournir les AWS ressources dont le périphérique principal a besoin pour fonctionner. Vous pouvez également déployer les outils de développement sur un appareil principal pour l'utiliser comme périphérique de développement. Cela vous permet de fournir moins d'autorisations à l'appareil lorsque vous exécutez le programme d'installation. Pour de plus amples informations, veuillez consulter [Installation AWS IoT Greengrass du logiciel Core avec provisionnement manuel des ressources](#).

Important

Avant de télécharger le logiciel AWS IoT Greengrass Core, vérifiez que votre appareil principal répond à la [configuration requise](#) pour installer et exécuter le logiciel AWS IoT Greengrass Core v2.0.

Rubriques

- [Configuration de l'environnement de l'appareil](#)
- [Fournir des AWS informations d'identification à l'appareil](#)
- [Téléchargez le logiciel AWS IoT Greengrass de base](#)
- [Installation du logiciel AWS IoT Greengrass Core](#)

Configuration de l'environnement de l'appareil

Suivez les étapes décrites dans cette section pour configurer un appareil Linux ou Windows à utiliser comme périphérique AWS IoT Greengrass principal.

Configuration d'un appareil Linux

Pour configurer un appareil Linux pour AWS IoT Greengrass V2

1. Installez le moteur d'exécution Java, dont le logiciel AWS IoT Greengrass Core a besoin pour fonctionner. Nous vous recommandons d'utiliser les versions de support à long terme d'[Amazon Corretto](#) ou d'OpenJDK. La version 8 ou supérieure est requise. Les commandes suivantes vous montrent comment installer OpenJDK sur votre appareil.

- Pour les distributions basées sur Debian ou Ubuntu :

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

- Pour les distributions basées sur Red Hat :

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

- Dans Amazon Linux 2 :

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

- Pour Amazon Linux 2023 :

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

Lorsque l'installation est terminée, exécutez la commande suivante pour vérifier que Java s'exécute sur votre appareil Linux.

```
java -version
```

La commande affiche la version de Java exécutée sur le périphérique. Par exemple, sur une distribution basée sur Debian, le résultat peut ressembler à celui de l'exemple suivant.

```
openjdk version "11.0.9.1" 2020-11-04
```

```
OpenJDK Runtime Environment (build 11.0.9.1+1-post-Debian-1deb10u2)
OpenJDK 64-Bit Server VM (build 11.0.9.1+1-post-Debian-1deb10u2, mixed mode)
```

2. (Facultatif) Créez l'utilisateur système et le groupe par défaut qui exécutent les composants sur le périphérique. Vous pouvez également choisir de laisser le programme d'installation du logiciel AWS IoT Greengrass Core créer cet utilisateur et ce groupe lors de l'installation avec l'argument `--component-default-user` installer. Pour de plus amples informations, veuillez consulter [Arguments d'installation](#).

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

3. Vérifiez que l'utilisateur qui exécute le logiciel AWS IoT Greengrass Core (généralement `root`) est autorisé à exécuter `sudo` avec n'importe quel utilisateur et n'importe quel groupe.

- a. Exécutez la commande suivante pour ouvrir le `/etc/sudoers` fichier.

```
sudo visudo
```

- b. Vérifiez que l'autorisation accordée à l'utilisateur ressemble à l'exemple suivant.

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

4. (Facultatif) Pour [exécuter des fonctions Lambda conteneurisées](#), vous devez activer [cgroups](#) v1, et vous devez activer et monter les cgroups de mémoire et de périphériques. Si vous ne prévoyez pas d'exécuter des fonctions Lambda conteneurisées, vous pouvez ignorer cette étape.

Pour activer ces options cgroups, démarrez le périphérique avec les paramètres du noyau Linux suivants.

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

Pour plus d'informations sur l'affichage et la définition des paramètres du noyau de votre appareil, consultez la documentation de votre système d'exploitation et de votre chargeur de démarrage. Suivez les instructions pour définir définitivement les paramètres du noyau.

5. Installez toutes les autres dépendances requises sur votre appareil, comme indiqué dans la liste des exigences figurant dans [Exigences relatives aux dispositifs](#).

Configuration d'un appareil Windows

Note

Cette fonctionnalité est disponible pour les versions 2.5.0 et ultérieures du composant [Greengrass](#) nucleus.

Pour configurer un appareil Windows pour AWS IoT Greengrass V2

1. Installez le moteur d'exécution Java, dont le logiciel AWS IoT Greengrass Core a besoin pour fonctionner. Nous vous recommandons d'utiliser les versions de support à long terme d'[Amazon Corretto](#) ou d'OpenJDK. La version 8 ou supérieure est requise.
2. Vérifiez si Java est disponible sur la variable système [PATH](#), et ajoutez-le dans le cas contraire. Le LocalSystem compte exécute le logiciel AWS IoT Greengrass Core. Vous devez donc ajouter Java à la variable système PATH au lieu de la variable utilisateur PATH pour votre utilisateur. Procédez comme suit :
 - a. Appuyez sur la touche Windows pour ouvrir le menu de démarrage.
 - b. Tapez **environment variables** pour rechercher les options du système dans le menu Démarrer.
 - c. Dans les résultats de recherche du menu Démarrer, choisissez Modifier les variables d'environnement du système pour ouvrir la fenêtre des propriétés du système.
 - d. Choisissez les variables d'environnement... pour ouvrir la fenêtre des variables d'environnement.
 - e. Sous Variables système, sélectionnez Chemin, puis Modifier. Dans la fenêtre Modifier la variable d'environnement, vous pouvez afficher chaque chemin sur une ligne distincte.
 - f. Vérifiez si le chemin d'accès au bin dossier d'installation de Java est présent. Le chemin peut ressembler à celui de l'exemple suivant.

```
C:\\Program Files\\Amazon Corretto\\jdk11.0.13_8\\bin
```
 - g. Si le bin dossier de l'installation Java est absent de Path, choisissez Nouveau pour l'ajouter, puis OK.
3. Ouvrez l'invite de commande Windows (cmd.exe) en tant qu'administrateur.
4. Créez l'utilisateur par défaut dans le LocalSystem compte sur l'appareil Windows.
password Remplacez-le par un mot de passe sécurisé.

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

Tip

En fonction de votre configuration Windows, le mot de passe de l'utilisateur peut être configuré pour expirer à une date ultérieure. Pour vous assurer que vos applications Greengrass continuent de fonctionner, suivez la date d'expiration du mot de passe et mettez-le à jour avant son expiration. Vous pouvez également définir le mot de passe de l'utilisateur pour qu'il n'expire jamais.

- Pour vérifier la date d'expiration d'un utilisateur et de son mot de passe, exécutez la commande suivante.

```
net user ggc_user | findstr /C:expires
```

- Pour définir le mot de passe d'un utilisateur de manière à ce qu'il n'expire jamais, exécutez la commande suivante.

```
wmic UserAccount where "Name='ggc_user'" set PasswordExpires=False
```

- Si vous utilisez Windows 10 ou une version ultérieure où la [wmiccommande est obsolète](#), exécutez la commande suivante PowerShell .

```
Get-CimInstance -Query "SELECT * from Win32_UserAccount WHERE name = 'ggc_user'" | Set-CimInstance -Property @{PasswordExpires="False"}
```

5. Téléchargez et installez l'[PsExecutilitaire](#) de Microsoft sur l'appareil.
6. Utilisez l' PsExec utilitaire pour stocker le nom d'utilisateur et le mot de passe de l'utilisateur par défaut dans l'instance Credential Manager du LocalSystem compte. *password* Remplacez-le par le mot de passe utilisateur que vous avez défini précédemment.

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

S'il PsExec License Agreements'ouvre, choisissez Acceptd'accepter la licence et exécutez la commande.

 Note

Sur les appareils Windows, le LocalSystem compte exécute le noyau Greengrass, et vous devez utiliser l' PsExec utilitaire pour stocker les informations utilisateur par défaut dans le LocalSystem compte. L'application Credential Manager stocke ces informations dans le compte Windows de l'utilisateur actuellement connecté, plutôt que dans le LocalSystem compte.

Fournir des AWS informations d'identification à l'appareil

Fournissez vos AWS informations d'identification à votre appareil afin que le programme d'installation puisse fournir les AWS ressources nécessaires. Pour plus d'informations sur les autorisations requises, consultez [Politique IAM minimale permettant au programme d'installation de provisionner les ressources](#).

Pour fournir des AWS informations d'identification à l'appareil

- Fournissez vos AWS informations d'identification à l'appareil afin que le programme d'installation puisse fournir les ressources AWS IoT et IAM pour votre appareil principal. Pour renforcer la sécurité, nous vous recommandons d'obtenir des informations d'identification temporaires pour un rôle IAM qui n'accorde que les autorisations minimales nécessaires au provisionnement. Pour de plus amples informations, veuillez consulter [Politique IAM minimale permettant au programme d'installation de provisionner les ressources](#).

 Note

Le programme d'installation n'enregistre ni ne stocke vos informations d'identification.

Sur votre appareil, effectuez l'une des opérations suivantes pour récupérer les informations d'identification et les mettre à la disposition du programme d'installation du logiciel AWS IoT Greengrass Core :

- (Recommandé) Utilisez des informations d'identification temporaires provenant de AWS IAM Identity Center

- a. Fournissez l'ID de clé d'accès, la clé d'accès secrète et le jeton de session provenant du centre d'identité IAM. Pour plus d'informations, voir Actualisation manuelle des informations d'identification dans la section Obtenir et actualiser [des informations d'identification temporaires](#) dans le guide de l'utilisateur d'IAM Identity Center.
- b. Exécutez les commandes suivantes pour fournir les informations d'identification au logiciel AWS IoT Greengrass Core.

Linux or Unix

```
export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
export AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o50ytwEXAMPLE=
```

Windows Command Prompt (CMD)

```
set AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
set AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
set AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o50ytwEXAMPLE=
```

PowerShell

```
$env:AWS_ACCESS_KEY_ID="AKIAIOSFODNN7EXAMPLE"
$env:AWS_SECRET_ACCESS_KEY="wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY"
$env:AWS_SESSION_TOKEN="AQoDYXdzEJr1K...o50ytwEXAMPLE="
```

- Utilisez les informations d'identification de sécurité temporaires issues d'un rôle IAM :
 - a. Fournissez l'ID de clé d'accès, la clé d'accès secrète et le jeton de session correspondant au rôle IAM que vous assumez. Pour plus d'informations sur la façon de récupérer ces informations d'identification, consultez la section [Demande d'informations d'identification de sécurité temporaires](#) dans le guide de l'utilisateur IAM.
 - b. Exécutez les commandes suivantes pour fournir les informations d'identification au logiciel AWS IoT Greengrass Core.

Linux or Unix

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

```
export AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o50ytwEXAMPLE=
```

Windows Command Prompt (CMD)

```
set AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE  
set AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY  
set AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o50ytwEXAMPLE=
```

PowerShell

```
$env:AWS_ACCESS_KEY_ID="AKIAIOSFODNN7EXAMPLE"  
$env:AWS_SECRET_ACCESS_KEY="wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY"  
$env:AWS_SESSION_TOKEN="AQoDYXdzEJr1K...o50ytwEXAMPLE="
```

- Utilisez les informations d'identification à long terme d'un utilisateur IAM :
 - a. Fournissez l'ID de clé d'accès et la clé d'accès secrète pour votre utilisateur IAM. Vous pouvez créer un utilisateur IAM pour le provisionnement, que vous supprimerez ultérieurement. Pour connaître la politique IAM à communiquer à l'utilisateur, consultez [Politique IAM minimale permettant au programme d'installation de provisionner les ressources](#). Pour plus d'informations sur la façon de récupérer des informations d'identification à long terme, consultez [la section Gestion des clés d'accès pour les utilisateurs IAM](#) dans le guide de l'utilisateur IAM.
 - b. Exécutez les commandes suivantes pour fournir les informations d'identification au logiciel AWS IoT Greengrass Core.

Linux or Unix

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

Windows Command Prompt (CMD)

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

PowerShell

```
$env:AWS_ACCESS_KEY_ID="AKIAIOSFODNN7EXAMPLE"
```

```
$env:AWS_SECRET_ACCESS_KEY="wJa1rXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY"
```

- c. (Facultatif) Si vous avez créé un utilisateur IAM pour approvisionner votre appareil Greengrass, supprimez-le.
- d. (Facultatif) Si vous avez utilisé l'ID de clé d'accès et la clé d'accès secrète d'un utilisateur IAM existant, mettez à jour les clés de cet utilisateur afin qu'elles ne soient plus valides. Pour plus d'informations, consultez la section [Mise à jour des clés d'accès](#) dans le guide de Gestion des identités et des accès AWS l'utilisateur.

Téléchargez le logiciel AWS IoT Greengrass de base

Vous pouvez télécharger la dernière version du logiciel AWS IoT Greengrass Core à l'adresse suivante :

- <https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest.zip>

Note

Vous pouvez télécharger une version spécifique du logiciel AWS IoT Greengrass Core à l'emplacement suivant. Remplacez *version* par la version à télécharger.

```
https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-version.zip
```

Pour télécharger le logiciel AWS IoT Greengrass Core

1. Sur votre appareil principal, téléchargez le logiciel AWS IoT Greengrass Core dans un fichier nommé `greengrass-nucleus-latest.zip`.

Linux or Unix

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

Windows Command Prompt (CMD)

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

PowerShell

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

Si vous téléchargez ce logiciel, vous acceptez le [contrat de licence du logiciel Greengrass Core](#).

2. (Facultatif) Pour vérifier la signature du logiciel Greengrass Nucleus

Note

Cette fonctionnalité est disponible avec Greengrass nucleus version 2.9.5 et versions ultérieures.

- a. Utilisez la commande suivante pour vérifier la signature de votre artefact Greengrass nucleus :

Linux or Unix

```
jarsigner -verify -certs -verbose greengrass-nucleus-latest.zip
```

Windows Command Prompt (CMD)

Le nom du fichier peut être différent selon la version du JDK que vous installez.
jdk17.0.6_10 Remplacez-le par la version du JDK que vous avez installée.

```
"C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe" -verify -certs -verbose greengrass-nucleus-latest.zip
```

PowerShell

Le nom du fichier peut être différent selon la version du JDK que vous installez.
`jdk17.0.6_10` Remplacez-le par la version du JDK que vous avez installée.

```
'C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe' -
verify -certs -verbose greengrass-nucleus-latest.zip
```

- b. L'`jarsigner` invocation produit une sortie qui indique les résultats de la vérification.
 - i. Si le fichier zip Greengrass nucleus est signé, le résultat contient l'instruction suivante :

```
jar verified.
```

- ii. Si le fichier zip Greengrass nucleus n'est pas signé, le résultat contient l'instruction suivante :

```
jar is unsigned.
```

- c. Si vous avez fourni l'`-certs` option Jarsigner en même temps que les `-verbose` options `-verify` et, le résultat inclut également des informations détaillées sur le certificat du signataire.
3. Décompressez le logiciel AWS IoT Greengrass Core dans un dossier de votre appareil.
`GreengrassInstaller` Remplacez-le par le dossier que vous souhaitez utiliser.

Linux or Unix

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

Windows Command Prompt (CMD)

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

PowerShell

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



```
rm greengrass-nucleus-latest.zip
```

4. (Facultatif) Exécutez la commande suivante pour voir la version du logiciel AWS IoT Greengrass Core.

```
java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
```

Important

Si vous installez une version du noyau Greengrass antérieure à la version 2.4.0, ne supprimez pas ce dossier après avoir installé le logiciel Core. AWS IoT Greengrass Le logiciel AWS IoT Greengrass Core utilise les fichiers de ce dossier pour s'exécuter.

Si vous avez téléchargé la dernière version du logiciel, vous devez installer la version 2.4.0 ou ultérieure, et vous pouvez supprimer ce dossier après avoir installé le logiciel AWS IoT Greengrass Core.

Installation du logiciel AWS IoT Greengrass Core

Exécutez le programme d'installation avec des arguments spécifiant les actions suivantes :

- Créez les AWS ressources dont le périphérique principal a besoin pour fonctionner.
- Spécifiez l'utilisation de l'utilisateur `ggc_user` du système pour exécuter les composants logiciels sur le périphérique principal. Sur les appareils Linux, cette commande indique également d'utiliser le groupe `ggc_group` système, et le programme d'installation crée l'utilisateur et le groupe système pour vous.
- Configurez le logiciel AWS IoT Greengrass Core en tant que service système qui s'exécute au démarrage. Sur les appareils Linux, cela nécessite le [système d'initialisation Systemd](#).

Important

Sur les appareils Windows Core, vous devez configurer le logiciel AWS IoT Greengrass Core en tant que service système.

Pour configurer un dispositif de développement avec des outils de développement locaux, spécifiez l'argument `--deploy-dev-tools true`. Le déploiement des outils de développement local peut prendre jusqu'à une minute une fois l'installation terminée.

Pour plus d'informations sur les arguments que vous pouvez spécifier, consultez [Arguments d'installation](#).

Note

Si vous utilisez AWS IoT Greengrass sur un appareil dont la mémoire est limitée, vous pouvez contrôler la quantité de mémoire utilisée par le logiciel AWS IoT Greengrass Core. Pour contrôler l'allocation de mémoire, vous pouvez définir les options de taille de segment de mémoire JVM dans le paramètre de `jvmOptions` configuration de votre composant Nucleus. Pour de plus amples informations, veuillez consulter [Contrôlez l'allocation de mémoire grâce aux options JVM](#).

Pour installer le logiciel AWS IoT Greengrass Core

1. Utilisez un éditeur de texte pour créer un fichier de configuration nommé `config.yaml` à fournir au programme d'installation.

Par exemple, sur un système basé sur Linux, vous pouvez exécuter la commande suivante pour utiliser GNU nano pour créer le fichier.

```
nano GreengrassInstaller/config.yaml
```

Copiez le contenu YAML suivant dans le fichier. Ce fichier de configuration partiel spécifie les paramètres du système et les paramètres du noyau Greengrass.

```
---
services:
  aws.greengrass.Nucleus:
    configuration:
      fipsMode: "true"
      iotDataEndpoint: "data.iot-fips.us-west-2.amazonaws.com"
      iotCredEndpoint: "data.credentials.iot-fips.us-west-2.amazonaws.com"
      greengrassDataPlaneEndpoint: "iotData"
```

- Remplacez *us-west-2* par l' Région AWS endroit où vous avez créé les ressources.
 - Remplacez le *iotDataEndpoint* par votre point de terminaison de AWS IoT données.
 - Remplacez le point de terminaison *iotCredEndpoint* par vos AWS IoT informations d'identification.
2. Exécutez le programme d'installation AWS IoT Greengrass Core. Remplacez les valeurs des arguments dans votre commande comme suit.

 Note

Pour Windows, la limitation de la longueur du chemin est de 260 caractères. Si vous utilisez Windows, utilisez un dossier racine tel que `C:\greengrass\v2` ou `D:\greengrass\v2` pour conserver les chemins des composants Greengrass en dessous de la limite de 260 caractères.

- a. */greengrass/v2* ou *C:\greengrass\v2* : chemin d'accès au dossier racine à utiliser pour installer le logiciel AWS IoT Greengrass Core.
- b. *GreengrassInstaller*. Le chemin d'accès au dossier dans lequel vous avez décompressé le programme d'installation du logiciel AWS IoT Greengrass Core.
- c. *region*. L' Région AWS endroit dans lequel trouver ou créer des ressources.
- d. *MyGreengrassCore*. Le nom de l' AWS IoT appareil principal de votre Greengrass. Si l'objet n'existe pas, le programme d'installation le crée. Le programme d'installation télécharge les certificats pour s'authentifier en tant qu' AWS IoT objet. Pour de plus amples informations, veuillez consulter [Authentification et autorisation de l'appareil pour AWS IoT Greengrass](#).

 Note

Le nom de l'objet ne peut pas contenir de caractères deux-points (:).

- e. *MyGreengrassCoreGroup*. Le nom du AWS IoT groupe d'objets de votre appareil Greengrass principal. Si le groupe d'objets n'existe pas, le programme d'installation le crée et y ajoute l'objet. Si le groupe d'objets existe et fait l'objet d'un déploiement actif, le périphérique principal télécharge et exécute le logiciel spécifié par le déploiement.

 Note

Le nom du groupe d'objets ne peut pas contenir de deux-points (:).

- f. *GreengrassV2IoTThingPolicy*. Le nom de la AWS IoT politique qui permet aux appareils principaux de Greengrass de communiquer avec AWS IoT et. AWS IoT Greengrass Si la AWS IoT politique n'existe pas, le programme d'installation crée une AWS IoT politique permissive portant ce nom. Vous pouvez restreindre les autorisations de cette politique pour votre cas d'utilisation. Pour de plus amples informations, veuillez consulter [AWS IoT Politique minimale pour les appareils AWS IoT Greengrass V2 principaux](#).
- g. *GreengrassV2TokenExchangeRole*. Nom du rôle IAM qui permet au périphérique principal de Greengrass d'obtenir AWS des informations d'identification temporaires. Si le rôle n'existe pas, le programme d'installation le crée, puis crée et attache une politique nommée *GreengrassV2TokenExchangeRoleAccess*. Pour de plus amples informations, veuillez consulter [Autoriser les appareils principaux à interagir avec les AWS services](#).
- h. *GreengrassCoreTokenExchangeRoleAlias*. Alias du rôle IAM qui permet au périphérique principal de Greengrass d'obtenir des informations d'identification temporaires ultérieurement. Si l'alias de rôle n'existe pas, le programme d'installation le crée et le pointe vers le rôle IAM que vous spécifiez. Pour de plus amples informations, veuillez consulter [Autoriser les appareils principaux à interagir avec les AWS services](#).

Linux or Unix

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

Windows Command Prompt (CMD)

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

PowerShell

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

Important

Sur les appareils Windows Core, vous `--setup-system-service true` devez spécifier de configurer le logiciel AWS IoT Greengrass Core en tant que service système.

Le programme d'installation affiche les messages suivants en cas de succès :

- Si vous le spécifiez `--provision`, le programme d'installation imprime `Successfully configured Nucleus with provisioned resource details` s'il a correctement configuré les ressources.

- Si vous le spécifiez `--deploy-dev-tools`, le programme d'installation affiche `Configured Nucleus to deploy aws.greengrass.Cli` component s'il a créé le déploiement avec succès.
 - Si vous le spécifiez `--setup-system-service true`, le programme d'installation affiche `Successfully set up Nucleus as a system service` s'il a configuré et exécuté le logiciel en tant que service.
 - Si vous ne le spécifiez pas `--setup-system-service true`, le programme d'installation affiche `Launched Nucleus successfully` s'il a réussi et a exécuté le logiciel.
3. Ignorez cette étape si vous avez installé la [Noyau de Greengrass](#) version 2.0.4 ou une version ultérieure. Si vous avez téléchargé la dernière version du logiciel, vous avez installé la version 2.0.4 ou ultérieure.

Exécutez la commande suivante pour définir les autorisations de fichier requises pour le dossier racine de votre logiciel AWS IoT Greengrass Core. Remplacez-le `/greengrass/v2` par le dossier racine que vous avez spécifié dans votre commande d'installation et remplacez-le `/greengrass` par le dossier parent de votre dossier racine.

```
sudo chmod 755 /greengrass/v2 && sudo chmod 755 /greengrass
```

Si vous avez installé le logiciel AWS IoT Greengrass Core en tant que service système, le programme d'installation exécute le logiciel pour vous. Dans le cas contraire, vous devez exécuter le logiciel manuellement. Pour de plus amples informations, veuillez consulter [Exécutez le logiciel AWS IoT Greengrass Core](#).

Note

Par défaut, le rôle IAM créé par le programme d'installation n'autorise pas l'accès aux artefacts des composants dans les compartiments S3. Pour déployer des composants personnalisés qui définissent des artefacts dans Amazon S3, vous devez ajouter des autorisations au rôle afin de permettre à votre appareil principal de récupérer les artefacts des composants. Pour de plus amples informations, veuillez consulter [Autoriser l'accès aux compartiments S3 pour les artefacts des composants](#).

Si vous ne possédez pas encore de compartiment S3 pour les artefacts des composants, vous pouvez ajouter ces autorisations ultérieurement après avoir créé un compartiment.

Pour plus d'informations sur la configuration et l'utilisation du logiciel AWS IoT Greengrass, consultez les rubriques suivantes :

- [Configuration du logiciel AWS IoT Greengrass de base](#)
- [Développer des AWS IoT Greengrass composants](#)
- [Déployer AWS IoT Greengrass des composants sur des appareils](#)
- [Interface de ligne de commande Greengrass](#)

Composants de première partie conformes à la norme FIPS

<code>aws.greengrass.Nucleus</code>	<code>data.iot-fips. <i>us-east-1</i> .amazonaws.com</code>
	<code>greengrass-fips. <i>us-east-1</i> .amazonaws.com</code>
	<code>data.credentials.iot-fips. <i>us-east-1</i> .amazonaws.com</code>
<code>aws.greengrass.TokenExchangeService</code>	<code>data.credentials.iot-fips. <i>us-east-1</i> .amazonaws.com</code>
<code>aws.greengrass.Cli</code>	
<code>aws.greengrass.StreamManager</code>	• <code>kinesis-fips. <i>us-east-1</i> .amazonaws.com</code> • <code>data.iotsitewise-fips. <i>us-east-1</i> .amazonaws.com</code> • <code>s3-fips.<i>us-east-1</i> .amazonaws.com</code>

	 Note Le gestionnaire de flux ne prend pas en charge le point de AWS IoT Analytics terminaison FIPS
<code>aws.greengrass.LogManager</code>	<code>logs-fips. <i>us-east-1</i> .amazonaws.com</code>
<code>aws.greengrass.crypto.Pkcs11Provider</code>	
<code>aws.greengrass.ShadowManager</code>	
<code>aws.greengrass.DockerApplicationManager</code>	<code>ecr-fips. <i>us-east-1</i> .amazonaws.com</code>
<code>aws.greengrass.SecretManager</code>	<code>secretsmanager-fips. <i>us-east-1</i> .amazonaws.com</code>
<code>aws.greengrass.telemetry.NucleusEmitter</code>	
<code>aws.greengrass.clientdevices.IPDetector</code>	
<code>aws.greengrass.DiskSpooler</code>	

Résilience dans AWS IoT Greengrass

L'infrastructure AWS mondiale est construite autour des régions et des zones de disponibilité d'Amazon Web Services. Chacune Région AWS fournit plusieurs zones de disponibilité physiquement séparées et isolées, connectées par un réseau à faible latence, à haut débit et hautement redondant. 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 davantage disponibles, tolérantes aux pannes et ont

une plus grande capacité de mise à l'échelle que les infrastructures traditionnelles à un ou plusieurs centres de données.

Pour plus d'informations, consultez [Infrastructure mondiale AWS](#).

Outre l'infrastructure AWS mondiale, AWS IoT Greengrass propose plusieurs fonctionnalités pour répondre à vos besoins en matière de résilience et de sauvegarde des données.

- Vous pouvez configurer un périphérique principal Greengrass pour écrire des journaux dans le système de fichiers local et dans Logs. CloudWatch Si le périphérique principal perd sa connectivité, il peut continuer à enregistrer les messages sur le système de fichiers. Lorsqu'il se reconnecte, il écrit les messages du journal dans CloudWatch Logs. Pour de plus amples informations, veuillez consulter [AWS IoT Greengrass Journaux de surveillance](#).
- Si un périphérique principal perd de l'alimentation pendant un déploiement, il reprend le déploiement après le redémarrage du logiciel AWS IoT Greengrass principal.
- Si un appareil principal perd sa connectivité Internet, les appareils clients Greengrass peuvent continuer à communiquer sur le réseau local.
- Vous pouvez créer des composants Greengrass qui lisent les [flux du gestionnaire](#) de flux et envoient les données vers des destinations de stockage locales.

Sécurité de l'infrastructure dans AWS IoT Greengrass

En tant que service géré, AWS IoT Greengrass il est protégé par les procédures de sécurité du réseau AWS mondial décrites dans le livre blanc [Amazon Web Services : présentation des processus de sécurité](#).

Vous utilisez des appels d'API AWS publiés pour accéder AWS IoT Greengrass via le réseau. Les clients doivent prendre en charge le protocole TLS (Transport Layer Security) 1.2 ou version ultérieure. Nous recommandons TLS 1.3 ou version ultérieure. Les clients doivent également prendre en charge les suites de chiffrement PFS (Perfect Forward Secrecy) comme Ephemeral Diffie-Hellman (DHE) ou Elliptic Curve Ephemeral Diffie-Hellman (ECDHE) La plupart des systèmes modernes tels que Java 7 et les versions ultérieures prennent en charge ces modes.

Les demandes doivent être signées à l'aide d'un identifiant de clé d'accès et d'une clé d'accès secrète associée à un mandataire 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 un AWS IoT Greengrass environnement, les appareils utilisent des certificats X.509 et des clés cryptographiques pour se connecter et s'authentifier auprès du. AWS Cloud Pour de plus amples informations, veuillez consulter [the section called “Authentification et autorisation d'appareil”](#).

Analyse de configuration et de vulnérabilité 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 répond à ces défis en fournissant des outils permettant d'identifier les problèmes de sécurité et les écarts par rapport aux meilleures pratiques. Vous pouvez l'utiliser AWS IoT Device Defender pour analyser, auditer et surveiller les appareils connectés afin de détecter les comportements anormaux et d'atténuer les risques de sécurité. AWS IoT Device Defender peut auditer les appareils pour s'assurer qu'ils respectent les meilleures pratiques de sécurité et détecter les comportements anormaux sur les appareils. Cela permet d'appliquer des politiques de sécurité cohérentes sur tous vos appareils et de réagir rapidement lorsque des appareils sont compromis. IF pour plus d'informations, consultez les rubriques suivantes :

- Le [composant Device Defender](#)
- [AWS IoT Device Defender](#) dans le Manuel du développeur AWS IoT Core .

Dans AWS IoT Greengrass les environnements, vous devez prendre en compte les points suivants :

- Il est de votre responsabilité de sécuriser vos appareils physiques, le système de fichiers de vos appareils et le réseau local.
- AWS IoT Greengrass n'impose pas l'isolation du réseau pour les composants Greengrass définis par l'utilisateur, qu'ils soient exécutés ou non dans un conteneur Greengrass. Il est donc possible pour les composants de Greengrass de communiquer avec tout autre processus exécuté dans le système ou à l'extérieur via le réseau.

Intégrité du code dans AWS IoT Greengrass V2

AWS IoT Greengrass déploie les composants logiciels depuis les AWS Cloud appareils qui exécutent le logiciel AWS IoT Greengrass principal. Ces composants logiciels incluent les [composants AWS fournis](#) et les [composants personnalisés](#) que vous téléchargez sur votre Compte AWS. Chaque composant est composé d'une recette. La recette définit les métadonnées du composant et un certain nombre d'artefacts, qui sont des binaires de composants, tels que le code compilé et les ressources statiques. Les artefacts des composants sont stockés dans Amazon S3.

Lorsque vous développez et déployez des composants Greengrass, vous devez suivre ces étapes de base qui fonctionnent avec les artefacts des composants présents sur votre appareil Compte AWS et sur celui-ci :

1. Créez et téléchargez des artefacts dans des compartiments S3.
2. Créez un composant à partir d'une recette et d'artefacts du AWS IoT Greengrass service, qui calcule le [hachage cryptographique](#) de chaque artefact.
3. Déployez un composant sur les appareils principaux de Greengrass, qui téléchargent et vérifient l'intégrité de chaque artefact.

AWS est responsable du maintien de l'intégrité des artefacts une fois que vous les avez chargés dans des compartiments S3, y compris lorsque vous déployez des composants sur les appareils principaux de Greengrass. Vous êtes responsable de la sécurisation des artefacts logiciels avant de les télécharger dans des compartiments S3. Vous êtes également responsable de la sécurisation de l'accès aux ressources qu'il contient, y compris aux compartiments S3 dans lesquels vous téléchargez les artefacts des composants.

Note

Amazon S3 fournit une fonctionnalité appelée S3 Object Lock que vous pouvez utiliser pour vous protéger contre les modifications apportées aux artefacts des composants dans vos Compte AWS compartiments S3. Vous pouvez utiliser S3 Object Lock pour empêcher la suppression ou le remplacement d'artefacts de composants. Pour plus d'informations, consultez la section [Utilisation du verrouillage d'objets S3](#) dans le guide de l'utilisateur d'Amazon Simple Storage Service.

Lorsque vous AWS publiez un composant public et que vous téléchargez un composant personnalisé, AWS IoT Greengrass calcule un condensé cryptographique pour chaque artefact de composant. AWS IoT Greengrass met à jour la recette du composant pour inclure le condensé de chaque artefact et l'algorithme de hachage utilisé pour calculer ce condensé. Ce condensé garantit l'intégrité de l'artefact, car si l'artefact change pendant AWS Cloud ou pendant le téléchargement, son condensé de fichier ne correspondra pas au condensé AWS IoT Greengrass stocké dans la recette du composant. Pour plus d'informations, consultez la section [Artefacts dans la référence de la recette du composant](#).

Lorsque vous déployez un composant sur un périphérique principal, le logiciel AWS IoT Greengrass Core télécharge la recette du composant et chaque artefact de composant défini par la recette. Le logiciel AWS IoT Greengrass Core calcule le résumé de chaque fichier d'artefact téléchargé et le compare au résumé de cet artefact dans la recette. Si les résumés ne correspondent pas, le déploiement échoue et le logiciel AWS IoT Greengrass Core supprime les artefacts téléchargés du système de fichiers de l'appareil. Pour plus d'informations sur la manière dont les connexions entre les appareils principaux AWS IoT Greengrass sont sécurisées, consultez [Chiffrement en transit](#).

Vous êtes responsable de la sécurisation des fichiers d'artefacts des composants sur les systèmes de fichiers de vos appareils principaux. Le logiciel AWS IoT Greengrass Core enregistre les artefacts dans le packages dossier du dossier racine de Greengrass. Vous pouvez l'utiliser AWS IoT Device Defender pour analyser, auditer et surveiller les appareils principaux. Pour de plus amples informations, veuillez consulter [Analyse de configuration et de vulnérabilité dans AWS IoT Greengrass](#).

AWS IoT Greengrass et points de terminaison VPC d'interface ([AWS PrivateLink](#))

Vous pouvez établir une connexion privée entre votre VPC et le plan de AWS IoT Greengrass contrôle en créant un point de terminaison VPC d'interface. Vous pouvez utiliser ce point de terminaison pour gérer les composants, les déploiements et les principaux appareils du AWS IoT Greengrass service. Les points de terminaison de l'interface sont alimentés par [AWS PrivateLink](#) une technologie qui vous permet d'accéder AWS IoT Greengrass APIs en privé sans passerelle Internet, appareil NAT, connexion VPN ou connexion AWS Direct Connect. Les instances de votre VPC n'ont pas besoin d'adresses IP publiques pour communiquer avec elles. AWS IoT Greengrass APIs Le trafic entre votre VPC et celui qui AWS IoT Greengrass ne quitte pas le réseau Amazon.

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 AWS IoT Greengrass terminaison VPC](#)
- [Création d'un point de terminaison VPC d'interface pour les opérations du plan AWS IoT Greengrass de contrôle](#)
- [Création d'une politique de point de terminaison VPC pour AWS IoT Greengrass](#)
- [Faire fonctionner un appareil AWS IoT Greengrass principal dans un VPC](#)

Considérations relatives aux points de AWS IoT Greengrass terminaison VPC

Avant de configurer un point de terminaison VPC d'interface pour AWS IoT Greengrass, consultez les [propriétés et les limites du point de terminaison d'interface](#) dans le guide de l'utilisateur Amazon VPC. Tenez également compte des considérations suivantes :

- AWS IoT Greengrass prend en charge les appels à toutes ses actions d'API du plan de contrôle depuis votre VPC. Le plan de commande inclut des opérations telles que [CreateDeployment](#) et [ListEffectiveDeployments](#). Le plan de contrôle n'inclut pas les opérations telles que [ResolveComponentCandidatesDiscover](#), qui sont des opérations de plan de données.
- Les points de terminaison VPC pour ne AWS IoT Greengrass sont actuellement pas pris en charge dans les régions de Chine AWS .

Création d'un point de terminaison VPC d'interface pour les opérations du plan AWS IoT Greengrass de contrôle

Vous pouvez créer un point de terminaison VPC pour le plan de AWS IoT Greengrass contrôle à l'aide de la console Amazon VPC ou du (). AWS Command Line Interface AWS CLI Pour plus d'informations, consultez [Création d'un point de terminaison d'interface](#) dans le Guide de l'utilisateur Amazon VPC.

Créez un point de terminaison VPC à l' AWS IoT Greengrass aide du nom de service suivant :

- com.amazonaws. *region*. herbe verte

Si vous activez le DNS privé pour le point de terminaison, vous pouvez envoyer des demandes d'API 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 politique de point de terminaison VPC pour AWS IoT Greengrass

Vous pouvez associer une politique de point de terminaison à votre point de terminaison VPC qui contrôle l'accès aux opérations du plan AWS IoT Greengrass de contrôle. La politique spécifie les informations suivantes :

- Le principal qui peut exécuter des actions.
- Les actions que le principal peut effectuer.
- Les ressources sur lesquelles le principal 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.

Exemple Exemple : politique de point de terminaison VPC pour les actions AWS IoT Greengrass

Voici un exemple de politique de point de terminaison pour AWS IoT Greengrass. Lorsqu'elle est attachée à un point de terminaison, cette politique accorde l'accès aux AWS IoT Greengrass actions répertoriées à tous les principaux sur toutes les ressources.

```
{
  "Statement": [
    {
      "Principal": "*",
      "Effect": "Allow",
      "Action": [
        "greengrass:CreateDeployment",
        "greengrass:ListEffectiveDeployments"
      ],
      "Resource": "*"
    }
  ]
}
```

Faire fonctionner un appareil AWS IoT Greengrass principal dans un VPC

Vous pouvez utiliser un appareil principal Greengrass et effectuer des déploiements en VPC sans accès public à Internet. Vous devez au minimum configurer les points de terminaison VPC suivants avec les alias DNS correspondants. Pour plus d'informations sur la création et l'utilisation de points de terminaison VPC, consultez la section Créer [un point de terminaison VPC dans le guide de l'utilisateur Amazon VPC](#).

Note

La fonctionnalité VPC permettant de créer automatiquement un enregistrement DNS est désactivée pour AWS IoT data et AWS IoT Credentials. Pour connecter ces points de terminaison, vous devez créer manuellement un enregistrement DNS privé. Pour plus d'informations, consultez la section [DNS privé pour les points de terminaison de l'interface](#). Pour plus d'informations sur les limites des AWS IoT Core VPC, consultez la section Limitations des points de [terminaison VPC](#).

Prérequis

- Vous devez installer le logiciel AWS IoT Greengrass Core en suivant les étapes de provisionnement manuel. Pour de plus amples informations, veuillez consulter [Installation AWS IoT Greengrass du logiciel Core avec provisionnement manuel des ressources](#).

Limites

- L'utilisation d'un appareil principal Greengrass en VPC n'est pas prise en charge dans les régions de Chine et. AWS GovCloud (US) Regions
- [Pour plus d'informations sur les limites des points de terminaison VPC du fournisseur AWS IoT d'informations d'identification AWS IoT data et les limites, consultez la section Limitations.](#)

Configurez votre appareil principal Greengrass pour qu'il fonctionne en VPC

1. Obtenez les AWS IoT points de terminaison qui vous Compte AWS conviennent et enregistrez-les pour les utiliser ultérieurement. Votre appareil utilise ces points de terminaison pour se connecter à AWS IoT. Procédez comme suit :

- a. Obtenez le point de terminaison de AWS IoT données pour votre Compte AWS.

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

La réponse ressemble à l'exemple suivant, si la demande aboutit.

```
{
  "endpointAddress": "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
}
```

- b. Obtenez le point de terminaison des informations d' AWS IoT identification pour votre Compte AWS.

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

La réponse ressemble à l'exemple suivant, si la demande aboutit.

```
{
  "endpointAddress": "device-credentials-prefix.credentials.iot.us-
west-2.amazonaws.com"
}
```

2. Créez une interface Amazon VPC pour les points de terminaison AWS IoT data et identifiez les points de AWS IoT terminaison :
 - a. Accédez au point de terminaison d'un [VPC](#) de la console , sous Virtual private cloud dans le menu de gauche, choisissez point de terminaison puis Créer un point de terminaison .
 - b. Dans la page Create Endpoint (Créer un point de terminaison, spécifiez les informations suivantes.
 - Choisissez Service AWS s pour la catégorie de service .
 - Pour Nom du service, effectuez une recherche en saisissant le mot-clé `iot`. Dans la liste des services `iot` affichés, choisissez le point de terminaison.

Si vous créez un point de terminaison VPC pour le plan de AWS IoT Core données, choisissez le point de terminaison de l'API du plan de AWS IoT Core données pour votre région. Le format du nom du point de terminaison est `com.amazonaws.region.iot.data`.

Si vous créez un point de terminaison VPC pour le fournisseur AWS IoT Core d'informations d'identification, choisissez le point de terminaison du fournisseur AWS IoT Core d'informations d'identification pour votre région. Le format du nom du point de terminaison est `com.amazonaws.region.iot.credentials`.

 Note

Le nom du service pour le plan de AWS IoT Core données dans la région de Chine sera au format suivant `cn.com.amazonaws.region.iot.data`. La création de points de terminaison VPC pour le fournisseur AWS IoT Core d'informations d'identification n'est pas prise en charge dans la région de la Chine.

- Pour le VPC et les sous-réseaux, choisissez le VPC dans lequel vous souhaitez créer le point de terminaison et les zones de disponibilité (AZs) dans lesquelles vous souhaitez créer le réseau de points de terminaison.
- Pour Activer le nom DNS, assurez-vous que Activer pour ce point de terminaison n'est pas sélectionné. Ni le plan de AWS IoT Core données ni le fournisseur AWS IoT Core d'identifiants ne prennent encore en charge les noms DNS privés.
- Pour (Groupe de sécurité), sélectionnez les groupes de sécurité que vous souhaitez associer aux interfaces réseau des points de terminaison.
- En option, vous pouvez ajouter ou supprimer des balises. Les balises sont des paires nom-valeur que vous utilisez pour associer à votre point de terminaison.

c. Pour créer votre point de terminaison VPC, choisissez Créer un point de terminaison.

3. Après avoir créé le AWS PrivateLink point de terminaison, dans l'onglet Détails de votre point de terminaison, vous verrez une liste de noms DNS. Vous pouvez utiliser l'un de ces noms DNS que vous avez créés dans cette section pour [configurer votre zone hébergée privée](#).
4. Créez un point de terminaison Amazon S3. Pour plus d'informations, consultez [Créer un point de terminaison VPC pour Amazon S3](#).
5. Si vous utilisez des composants [Greengrass AWS fournis](#), des points de terminaison et des configurations supplémentaires peuvent être nécessaires. Pour consulter les exigences relatives aux terminaux, sélectionnez le composant dans la liste des composants AWS fournis et consultez la section Exigences. Par exemple, les [exigences du composant du gestionnaire de journaux](#) indiquent que ce composant doit être capable d'effectuer des demandes sortantes vers le point de terminaison `logs.region.amazonaws.com`.

Si vous utilisez votre propre composant, vous devrez peut-être examiner les dépendances et effectuer des tests supplémentaires pour déterminer si des points de terminaison supplémentaires sont nécessaires.

6. Dans la configuration du noyau de Greengrass, `greengrassDataPlaneEndpoint` il doit être défini sur **iotdata**. Pour plus d'informations, consultez la section Configuration du [noyau de Greengrass](#).
7. Si vous vous trouvez dans la `us-east-1` région, définissez le paramètre de configuration `s3EndpointType` sur **REGIONAL** dans la configuration du noyau Greengrass. Cette fonctionnalité est disponible pour les versions 2.11.3 ou ultérieures de Greengrass nucleus.

Exemple Exemple : configuration des composants

```
{
  "aws.greengrass.Nucleus": {
    "configuration": {
      "awsRegion": "us-east-1",
      "iotCredEndpoint": "xxxxxx.credentials.iot.region.amazonaws.com",
      "iotDataEndpoint": "xxxxxx-ats.iot.region.amazonaws.com",
      "greengrassDataPlaneEndpoint": "iotdata",
      "s3EndpointType": "REGIONAL"
      ...
    }
  }
}
```

Le tableau suivant fournit des informations sur les alias DNS privés personnalisés correspondants.

Service	Nom du service de point de terminaison d'un VPC	Type de point de terminaison VPC	Alias DNS privé personnalisé	Remarques
AWS IoT data	<code>com.amazonaws.<i>region</i>.iot.d</code>	utilisateur	<code><i>prefix</i>-ats.iot.<i>region</i>.s.com</code>	L'enregistrement DNS privé doit

Service	Nom du service de point de terminaison d'un VPC	Type de point de terminaison VPC	Alias DNS privé personnalisé	Remarques
				correspon dre au AWS IoT data point de terminais on de votre compte :aws iot describe- endpoint -- endpoin t-type iot:Data- ATS .

Service	Nom du service de point de terminaison d'un VPC	Type de point de terminaison VPC	Alias DNS privé personnalisé	Remarques
AWS IoT Informations d'identification	com.amazonaws. <i>region</i> .iot.credentials	utilisateur	<i>prefix</i> .credentials.iot.s.com	L'enregistrement DNS privé doit correspondre au point de terminaison des AWS IoT informations d'identification de votre compte :aws-iot-describe-endpoint -- endpoint-type iot:CredentialProvider .

Service	Nom du service de point de terminaison d'un VPC	Type de point de terminaison VPC	Alias DNS privé personnalisé	Remarques
Amazon S3	com.amazonaws. <i>region</i> .s3	utilisateur		L'enregistrement DNS est automatiquement créé.

Bonnes pratiques en matière de sécurité pour AWS IoT Greengrass

Cette rubrique présente les meilleures pratiques de sécurité pour AWS IoT Greengrass.

Accorder le moins d'autorisations possibles

Respectez le principe du moindre privilège pour vos composants en les exécutant en tant qu'utilisateurs non privilégiés. Les composants ne doivent pas être exécutés en tant que root, sauf si cela est absolument nécessaire.

Utilisez 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 noyau Greengrass.

Ne codez pas en dur les informations d'identification dans les composants Greengrass

Ne codez pas en dur les informations d'identification dans vos composants Greengrass définis par l'utilisateur. Pour mieux protéger vos informations d'identification :

- Pour interagir avec les AWS services, définissez des autorisations pour des actions et des ressources spécifiques dans le rôle de [service principal de Greengrass sur les appareils](#).
- Utilisez le [composant secret manager](#) 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 composants Greengrass ou vos fonctions Lambda définis par l'utilisateur.
- N'utilisez pas d'informations sensibles dans les noms et les ressources IDs de Greengrass :
 - Appareils principaux
 - Composants
 - Déploiements
 - Loggers

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 .

Recommandations de Cipher Suite

Greengrass sélectionne par défaut les dernières suites de chiffrement TLS disponibles sur l'appareil. Envisagez de désactiver l'utilisation des anciennes suites de chiffrement sur l'appareil. Par exemple, les suites de chiffrement CBC.

Pour plus d'informations, consultez la section [Configuration de la cryptographie Java](#).

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

Utilisation AWS IoT Device Tester pour la AWS IoT Greengrass V2

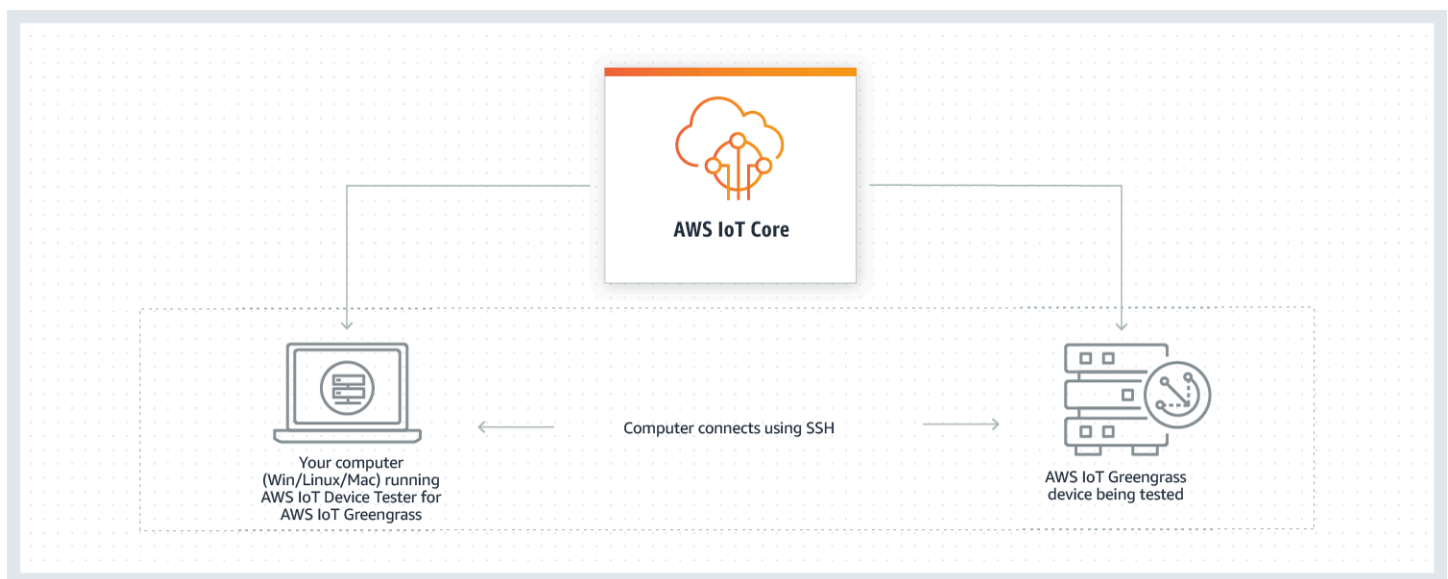
AWS IoT Device Tester (IDT) est un framework de test téléchargeable qui vous permet de valider les appareils IoT. Vous pouvez utiliser IDT AWS IoT Greengrass pour exécuter la suite de AWS IoT Greengrass qualification et créer et exécuter des suites de tests personnalisées pour vos appareils.

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 AWS IoT Device Tester for AWS IoT Greengrass V2 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 déployer des composants et les mettre à niveau.

Si vous souhaitez ajouter votre matériel au catalogue des AWS Partner appareils, exécutez la suite de AWS IoT Greengrass qualification pour générer des rapports de test auxquels vous pouvez les soumettre AWS IoT. Pour de plus amples informations, veuillez consulter [AWS Device Qualification Program](#).



IDT for AWS IoT Greengrass V2 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 test est l'ensemble de tests individuels liés à une fonctionnalité particulière, telle que les déploiements de composants.

Pour de plus amples informations, veuillez consulter [Utiliser IDT pour exécuter la suite de AWS IoT Greengrass qualifications](#).

Suites de tests personnalisées

À partir de IDT v4.0.1, IDT pour AWS IoT Greengrass V2 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 de plus amples informations, veuillez consulter [Utilisez IDT pour développer et exécuter vos propres suites de tests](#).

Versions prises en charge de AWS IoT Device Tester for AWS IoT Greengrass V2

Cette rubrique répertorie les versions prises en charge d'IDT pour AWS IoT Greengrass V2. À titre de bonne pratique, nous vous recommandons d'utiliser la dernière version d'IDT pour AWS IoT Greengrass V2 qui prend en charge votre version cible de la AWS IoT Greengrass V2. Les nouvelles versions de AWS IoT Greengrass peuvent vous obliger à télécharger une nouvelle version d'IDT pour AWS IoT Greengrass V2. Vous recevez une notification lorsque vous lancez un test si IDT pour AWS IoT Greengrass V2 n'est pas compatible avec la version AWS IoT Greengrass que vous utilisez.

En téléchargeant le logiciel, vous acceptez le [contrat AWS IoT Device Tester de licence](#).

 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.

Dernière version IDT pour V2 AWS IoT Greengrass

Vous pouvez utiliser cette version d'IDT pour AWS IoT Greengrass V2 avec la AWS IoT Greengrass version répertoriée ici.

IDT v4.9.4 pour AWS IoT Greengrass

AWS IoT Greengrass Versions prises en charge :

- [Greengrass Nucleus](#) v2.12.0, v2.11.0, v2.10.0 et v2.9.5

Téléchargements du logiciel IDT :

- [IDT v4.9.4 avec suite GGV2 de tests Q_2.5.4 pour Linux](#)
- [IDT v4.9.4 avec suite GGV2 de tests Q_2.5.4 pour macOS](#)
- [IDT v4.9.4 avec suite GGV2 de tests Q_2.5.4 pour Windows](#)

Notes de mise à jour :

- Permet la validation et la qualification des appareils exécutant les versions 2.12.0, 2.11.0, 2.10.0 et 2.9.5 du logiciel AWS IoT Greengrass Core.
- Supprime le gestionnaire de flux et les groupes de tests d'apprentissage automatique.

Remarques supplémentaires :

- Si votre appareil utilise un HSM et que vous utilisez nucleus 2.10.x, migrez vers Greengrass nucleus version 2.11.0 ou ultérieure.

Versions de suite de tests :

GGV2Q_2.5.4

- Publié le 03/05/2020

Versions IDT antérieures pour AWS IoT Greengrass

Les versions antérieures suivantes d'IDT pour AWS IoT Greengrass V2 sont également prises en charge.

IDT v4.9.3 pour AWS IoT Greengrass

AWS IoT Greengrass Versions prises en charge :

- [Greengrass Nucleus](#) v2.12.0, v2.11.0, v2.10.0 et v2.9.5

Téléchargements du logiciel IDT :

- [IDT v4.9.3 avec suite GGV2 de tests Q_2.5.3 pour Linux](#)
- [IDT v4.9.3 avec suite GGV2 de tests Q_2.5.3 pour macOS](#)
- [IDT v4.9.3 avec suite GGV2 de tests Q_2.5.3 pour Windows](#)

Notes de mise à jour :

- Résout un problème lors des tests de composants lors du test d'un périphérique Linux à partir d'un hôte Windows ou vice versa.
- Supprime le `localcomponent` scénario de test du groupe de component test. Ce cas de test n'est plus requis pour la qualification.

Remarques supplémentaires :

- Si votre appareil utilise un HSM et que vous utilisez nucleus 2.10.x, migrez vers Greengrass nucleus version 2.11.0 ou ultérieure.

Versions de suite de tests :

GGV2Q_2.5.3

- Publié le 2024.04.05

Versions non prises en charge de AWS IoT Device Tester for V2 AWS IoT Greengrass

Cette rubrique répertorie les versions non prises en charge d'IDT pour AWS IoT Greengrass V2. Les versions non prises en charge ne reçoivent pas de corrections de bogues ou de mises à jour. Pour de plus amples informations, veuillez consulter [the section called “Politique de support AWS IoT Device Tester pour AWS IoT Greengrass”](#).

IDT v4.9.2 pour AWS IoT Greengrass

Notes de mise à jour :

- Résout un problème d'échec de la suite de tests Lambda en raison de l'obsolescence de Java 8.

Versions de suite de tests :

GGV2Q_2.5.2

- Publié le 2024.03.18

IDT v4.9.1 pour AWS IoT Greengrass

Notes de mise à jour :

- Vous permet de valider et de qualifier les appareils exécutant les versions 2.12.0, 2.11.0, 2.10.0 et 2.9.5 du logiciel AWS IoT Greengrass Core.
- Correctifs de bogues mineurs.

Versions de suite de tests :

GGV2Q_2.5.1

- Publié le 2023.10.05

IDT v4.7.0 pour AWS IoT Greengrass

AWS IoT Greengrass Versions prises en charge :

- [Greengrass Nucleus](#) v2.11.0, v2.10.0 et v2.9.5

Notes de mise à jour :

- Vous permet de valider et de qualifier les appareils exécutant les versions 2.11.0, 2.10.0 et 2.9.5 du logiciel AWS IoT Greengrass Core.
- Permet de stocker les valeurs des données utilisateur IDT dans le AWS Systems Manager Parameter Store et de les récupérer dans la configuration à l'aide de la syntaxe des espaces réservés.
- Correctifs de bogues mineurs.

Versions de suite de tests :

GGV2Q_2.5.0

- Publié le 13 décembre 2020

IDT v4.5.11 pour AWS IoT Greengrass

Notes de mise à jour :

- Vous permet de valider et de qualifier les appareils exécutant les versions 2.9.1, 2.9.0, 2.8.1, 2.8.0, 2.7.0 et 2.6.0 du logiciel AWS IoT Greengrass Core.
- Permet de tester PreInstalled Greengrass sur un appareil principal.
- Correctifs de bogues mineurs.

Versions de suite de tests :

GGV2Q_2.4.1

- Publié le 2022.10.13

IDT v4.5.8 pour AWS IoT Greengrass

Notes de mise à jour :

- Vous permet de valider et de qualifier les appareils exécutant les versions 2.7.0, 2.6.0 et 2.5.6 du logiciel AWS IoT Greengrass Core.
- Vous permet de tester avec PreInstalled Greengrass sur un appareil principal.
- Correctifs de bogues mineurs.

Versions de suite de tests :

GGV2Q_2.4.0

- Publié le 12/08/2022

IDT v4.5.3 pour AWS IoT Greengrass

Notes de mise à jour :

- Vous permet de valider et de qualifier les appareils exécutant les versions 2.7.0, 2.6.0, 2.5.6, 2.5.5, 2.5.4 et 2.5.3 du logiciel AWS IoT Greengrass Core.
- Les mises à jour DockerApplicationManager testent pour utiliser une image docker basée sur l'ECR.
- Correctifs de bogues mineurs.

Versions de suite de tests :

GGV2Q_2.3.1

- Publié le 15 avril 2020

IDT v4.5.1 pour AWS IoT Greengrass

Notes de mise à jour :

- Vous permet de valider et de qualifier les appareils exécutant le logiciel AWS IoT Greengrass Core v2.5.3.
- Ajoute la prise en charge de la validation et de la qualification des appareils basés sur Linux qui utilisent un module de sécurité matériel (HSM) pour stocker la clé privée et le certificat utilisés par le logiciel Core. AWS IoT Greengrass
- Implémente le nouvel orchestrateur de tests IDT pour configurer des suites de tests personnalisées. Pour de plus amples informations, veuillez consulter [Configuration de l'orchestrateur de test IDT](#).
- Corrections de bogues mineurs supplémentaires.

Versions de suite de tests :

GGV2Q_2.3.0

- Publié le 2022.01.11

IDT v4.4.1 pour AWS IoT Greengrass

Notes de mise à jour :

- Vous permet de valider et de qualifier les appareils exécutant le logiciel AWS IoT Greengrass Core v2.5.2.
- Prend en charge l'utilisation d'un rôle IAM défini par l'utilisateur en tant que rôle d'échange de jetons que le périphérique testé suppose d'interagir avec AWS les ressources.

Vous pouvez spécifier le rôle IAM dans le [userdata.jsonfichier](#). Si vous spécifiez un rôle personnalisé, IDT utilise ce rôle au lieu de créer le rôle d'échange de jetons par défaut lors du test.

- Corrections de bogues mineurs supplémentaires.

Versions de suite de tests :

GGV2Q_2.2.1

- Publié le 12 décembre 2021

IDT v4.4.0 pour AWS IoT Greengrass

Notes de mise à jour :

- Vous permet de valider et de qualifier les appareils exécutant le logiciel AWS IoT Greengrass Core v2.5.0.

- Ajoute la prise en charge de la validation et de la qualification des appareils exécutant le logiciel AWS IoT Greengrass Core sous Windows.
- Prend en charge l'utilisation de la validation par clé publique pour les connexions de périphériques Secure Shell (SSH).
- Améliore la politique IAM relative aux autorisations IDT grâce aux meilleures pratiques de sécurité.
- Corrections de bogues mineurs supplémentaires.

Versions de suite de tests :

GGV2Q_2.1.0

- Publié le 2021.11.19

IDT v4.2.0 pour AWS IoT Greengrass

Notes de mise à jour :

- Inclut la prise en charge de la qualification des fonctionnalités suivantes sur les appareils exécutant le logiciel AWS IoT Greengrass Core v2.2.0 et versions ultérieures :
 - Docker : confirme que les appareils peuvent télécharger une image de conteneur Docker depuis Amazon Elastic Container Registry (Amazon ECR).
 - [Apprentissage automatique : vérifie que les appareils peuvent effectuer des inférences d'apprentissage automatique \(ML\) à l'aide des frameworks Deep Learning Runtime ou Lite ML. TensorFlow](#)
 - Gestionnaire de flux : vérifie que les appareils peuvent télécharger, installer et exécuter le gestionnaire de flux. AWS IoT Greengrass
- Vous permet de valider et de qualifier les appareils exécutant le logiciel AWS IoT Greengrass Core v2.4.0, v2.3.0, v2.2.0 et v2.1.0.
- Regroupe les journaux de test pour chaque cas de test dans un `<test-case-id>` dossier distinct au sein du `<device-tester-extract-location>/results/<execution-id>/logs/<test-group-id>` répertoire.
- Corrections de bogues mineurs supplémentaires.

Versions de suite de tests :

GGV2Q_2.0.1

- Publié le 31 août 2021

IDT v4.1.0 pour AWS IoT Greengrass

Notes de mise à jour :

- Vous permet de valider et de qualifier les appareils exécutant le logiciel AWS IoT Greengrass Core v2.3.0, v2.2.0, v2.1.0 et v2.0.5.
- Améliore la `userdata.json` configuration en supprimant l'obligation de spécifier les `GreengrassCLIVersion` propriétés `GreengrassNucleusVersion` et.
- Inclut la prise en charge de la qualification des fonctionnalités Lambda et MQTT pour le logiciel AWS IoT Greengrass Core v2.1.0 et versions ultérieures. Vous pouvez désormais utiliser IDT for AWS IoT Greengrass V2 pour valider que votre périphérique principal peut exécuter des fonctions Lambda et qu'il peut publier des sujets MQTT et s'y AWS IoT Core abonner.
- Améliore les capacités de journalisation.
- Corrections de bogues mineurs supplémentaires.

Versions de suite de tests :

GGV2Q_1.1.1

- Publié le 2021.06.18

IDT v4.0.2 pour AWS IoT Greengrass

Notes de mise à jour :

- Vous permet de valider et de qualifier les appareils exécutant le logiciel AWS IoT Greengrass Core v2.1.0.
- Ajoute la prise en charge de la qualification des fonctionnalités Lambda et MQTT pour le logiciel AWS IoT Greengrass Core v2.1.0 et les versions ultérieures. Vous pouvez désormais utiliser IDT for AWS IoT Greengrass V2 pour valider que votre périphérique principal peut exécuter des fonctions Lambda et qu'il peut publier des sujets MQTT et s'y AWS IoT Core abonner.
- Améliore les capacités de journalisation.
- Corrections de bogues mineurs supplémentaires.

Versions de suite de tests :

GGV2Q_1.1.1

- Publié le 2021.05.05

IDT v4.0.1 pour AWS IoT Greengrass

Notes de mise à jour :

- Vous permet de valider et de qualifier les appareils exécutant le logiciel AWS IoT Greengrass version 2.
- Vous permet de développer et d'exécuter vos suites de tests personnalisées à l'aide AWS IoT Device Tester de for AWS IoT Greengrass. Pour de plus amples informations, veuillez consulter [Utilisez IDT pour développer et exécuter vos propres suites de tests](#).
- Fournit des applications IDT signées par code pour macOS et Windows. Sur macOS, vous devrez peut-être accorder une exception de sécurité pour IDT. Pour de plus amples informations, veuillez consulter [Exception de sécurité sur macOS](#).

Versions de suite de tests :

GGV2Q_1.0.0

- Publié le 22 décembre 2020
- La suite de tests exécute uniquement les tests requis pour la qualification, sauf si vous définissez la valeur correspondante value dans le features tableau suryes.

Télécharger IDT pour V2 AWS IoT Greengrass

Cette rubrique décrit les options de téléchargement AWS IoT Device Tester pour la AWS IoT Greengrass version 2. Vous pouvez utiliser l'un des liens de téléchargement de logiciels suivants ou suivre les instructions pour télécharger IDT par programmation.

Rubriques

- [Télécharger IDT manuellement](#)
- [Téléchargez IDT par programmation](#)

En téléchargeant le logiciel, vous acceptez le [contrat AWS IoT Device Tester de licence](#).

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.

Télécharger IDT manuellement

Cette rubrique répertorie les versions prises en charge d'IDT pour AWS IoT Greengrass V2. À titre de bonne pratique, nous vous recommandons d'utiliser la dernière version d'IDT pour AWS IoT Greengrass V2 qui prend en charge votre version cible de la AWS IoT Greengrass V2. Les nouvelles versions de AWS IoT Greengrass peuvent vous obliger à télécharger une nouvelle version d'IDT pour AWS IoT Greengrass V2. Vous recevez une notification lorsque vous lancez un test si IDT pour AWS IoT Greengrass V2 n'est pas compatible avec la version AWS IoT Greengrass que vous utilisez.

IDT v4.9.4 pour AWS IoT Greengrass

AWS IoT Greengrass Versions prises en charge :

- [Greengrass Nucleus](#) v2.12.0, v2.11.0, v2.10.0 et v2.9.5

Téléchargements du logiciel IDT :

- [IDT v4.9.4 avec suite GGV2 de tests Q_2.5.4 pour Linux](#)
- [IDT v4.9.4 avec suite GGV2 de tests Q_2.5.4 pour macOS](#)
- [IDT v4.9.4 avec suite GGV2 de tests Q_2.5.4 pour Windows](#)

Notes de mise à jour :

- Permet la validation et la qualification des appareils exécutant les versions 2.12.0, 2.11.0, 2.10.0 et 2.9.5 du logiciel AWS IoT Greengrass Core.
- Supprime le gestionnaire de flux et les groupes de tests d'apprentissage automatique.

Remarques supplémentaires :

- Si votre appareil utilise un HSM et que vous utilisez nucleus 2.10.x, migrez vers Greengrass nucleus version 2.11.0 ou ultérieure.

Versions de suite de tests :

GGV2Q_2.5.4

- Publié le 03/05/2020

Téléchargez IDT par programmation

IDT fournit une opération d'API que vous pouvez utiliser pour récupérer une URL où vous pouvez télécharger IDT par programmation. Vous pouvez également utiliser cette opération d'API pour vérifier si vous disposez de la dernière version d'IDT. Cette opération d'API possède le point de terminaison suivant.

```
https://download.devicetester.iotdevicesecosystem.amazonaws.com/latestidt
```

Pour appeler cette opération d'API, vous devez être autorisé à effectuer l'**iot-device-tester:LatestIdt**action. Incluez votre AWS signature et `iot-device-tester` utilisez-la comme nom de service.

Demande d'API

HostOs — Le système d'exploitation de la machine hôte. Sélectionnez parmi les options suivantes :

- `mac`
- `linux`
- `windows`

TestSuiteType — Le type de suite de tests. Choisissez l'option suivante :

GGV2— IDT pour V2 AWS IoT Greengrass

ProductVersion

(Facultatif) La version du noyau Greengrass. Le service renvoie la dernière version compatible d'IDT pour cette version du noyau Greengrass. Si vous ne spécifiez pas cette option, le service renvoie la dernière version d'IDT.

Réponse de l'API

La réponse de l'API est au format suivant. `DownloadURL` inclut un fichier zip.

```
{
  "Success": True or False,
  "Message": Message,
  "LatestBk": {
    "Version": The version of the IDT binary,
    "TestSuiteVersion": The version of the test suite,
    "DownloadURL": The URL to download the IDT Bundle, valid for one hour
  }
}
```

Exemples

Vous pouvez vous référer aux exemples suivants pour télécharger IDT par programmation. Ces exemples utilisent des informations d'identification que vous stockez dans les variables

d'AWS_SECRET_ACCESS_KEYenvironnement AWS_ACCESS_KEY_ID et. Pour suivre les meilleures pratiques de sécurité, ne stockez pas vos informations d'identification dans votre code.

Exemple Exemple : téléchargement à l'aide de la version 7.75.0 ou ultérieure de cURL (Mac et Linux)

Si vous utilisez la version 7.75.0 ou ultérieure de cURL, vous pouvez utiliser le `aws-sigv4` drapeau pour signer la demande d'API. Cet exemple utilise [jq](#) pour analyser l'URL de téléchargement à partir de la réponse.

Warning

L'`aws-sigv4`indicateur nécessite que les paramètres de requête de la requête CURL GET soient dans l'ordre de HostOs/ProductVersion/TestSuiteType ouHostOs/TestSuiteType. Les commandes non conformes entraîneront une erreur lors de l'obtention de signatures non concordantes pour la chaîne canonique par l'API Gateway.

Si le paramètre facultatif ProductVersion est inclus, vous devez utiliser une version de produit prise en charge, comme indiqué dans [Versions prises en charge de AWS IoT Device Tester for AWS IoT Greengrass V2](#).

- `us-west-2`Remplacez-le par votre Région AWS. Pour la liste des codes de région, voir [Points de terminaison régionaux](#).
- `linux`Remplacez-le par le système d'exploitation de votre machine hôte.
- `2.5.3`Remplacez-le par votre version de AWS IoT Greengrass Nucleus.

```
url=$(curl --request GET "https://
download.devicetester.iotdevicesecosystem.amazonaws.com/latestidt?
HostOs=linux&ProductVersion=2.5.3&TestSuiteType=GGV2" \
--user $AWS_ACCESS_KEY_ID:$AWS_SECRET_ACCESS_KEY \
--aws-sigv4 "aws:amz:us-west-2:iot-device-tester" \
| jq -r '.LatestBk["DownloadURL"]')

curl $url --output devicetester.zip
```

Exemple Exemple : téléchargement à l'aide d'une version antérieure de cURL (Mac et Linux)

Vous pouvez utiliser la commande cURL suivante avec une AWS signature que vous signez et calculez. Pour plus d'informations sur la façon de signer et de calculer une AWS signature, consultez [Signature des demandes AWS d'API](#).

- **linux** Remplacez-le par le système d'exploitation de votre machine hôte.
- Remplacez **Timestamp** par la date et l'heure, par exemple **20220210T004606Z**.
- Remplacez **Date** par la date, telle que **20220210**.
- **AWSRegion** Remplacez-le par votre Région AWS. Pour la liste des codes de région, voir [Points de terminaison régionaux](#).
- **AWSSignature** Remplacez-le par la [AWS signature](#) que vous générez.

```
curl --location --request GET 'https://
download.devicetester.iotdevicesecosystem.amazonaws.com/latestidt?
HostOs=linux&TestSuiteType=GGV2' \
--header 'X-Amz-Date: Timestamp \
--header 'Authorization: AWS4-HMAC-SHA256 Credential=$AWS_ACCESS_KEY_ID/Date/AWSRegion/
iot-device-tester/aws4_request, SignedHeaders=host;x-amz-date, Signature=AWSSignature'
```

Exemple Exemple : téléchargement à l'aide d'un script Python

Cet exemple utilise la bibliothèque de [requêtes](#) Python. Cet exemple est adapté de l'exemple Python pour [signer une demande d' AWS API](#) dans la référence AWS générale.

- Remplacez **us-west-2** par votre région. Pour la liste des codes de région, voir [Points de terminaison régionaux](#).
- **linux** Remplacez-le par le système d'exploitation de votre machine hôte.

```
# Copyright 2010-2022 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#
# This file is licensed under the Apache License, Version 2.0 (the "License").
# You may not use this file except in compliance with the License. A copy of the
# License is located at
#
# http://aws.amazon.com/apache2.0/
#
# This file is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS
# OF ANY KIND, either express or implied. See the License for the specific
# language governing permissions and limitations under the License.
#
# See: http://docs.aws.amazon.com/general/latest/gr/sigv4_signing.html
# This version makes a GET request and passes the signature
# in the Authorization header.
```

```

import sys, os, base64, datetime, hashlib, hmac
import requests # pip install requests
# ***** REQUEST VALUES *****
method = 'GET'
service = 'iot-device-tester'
host = 'download.devicetester.iotdevicesecosystem.amazonaws.com'
region = 'us-west-2'
endpoint = 'https://download.devicetester.iotdevicesecosystem.amazonaws.com/latestidt'
request_parameters = 'HostOs=linux&TestSuiteType=GGV2'

# Key derivation functions. See:
# http://docs.aws.amazon.com/general/latest/gr/signature-v4-examples.html#signature-v4-examples-python
def sign(key, msg):
    return hmac.new(key, msg.encode('utf-8'), hashlib.sha256).digest()

def getSignatureKey(key, dateStamp, regionName, serviceName):
    kDate = sign(('AWS4' + key).encode('utf-8'), dateStamp)
    kRegion = sign(kDate, regionName)
    kService = sign(kRegion, serviceName)
    kSigning = sign(kService, 'aws4_request')
    return kSigning

# Read AWS access key from env. variables or configuration file. Best practice is NOT
# to embed credentials in code.
access_key = os.environ.get('AWS_ACCESS_KEY_ID')
secret_key = os.environ.get('AWS_SECRET_ACCESS_KEY')
if access_key is None or secret_key is None:
    print('No access key is available.')
    sys.exit()

# Create a date for headers and the credential string
t = datetime.datetime.utcnow()
amzdate = t.strftime('%Y%m%dT%H%M%SZ')
datestamp = t.strftime('%Y%m%d') # Date w/o time, used in credential scope

# ***** TASK 1: CREATE A CANONICAL REQUEST *****
# http://docs.aws.amazon.com/general/latest/gr/sigv4-create-canonical-request.html
# Step 1 is to define the verb (GET, POST, etc.)--already done.
# Step 2: Create canonical URI--the part of the URI from domain to query
# string (use '/' if no path)
canonical_uri = '/latestidt'
# Step 3: Create the canonical query string. In this example (a GET request),
# request parameters are in the query string. Query string values must

```

```

# be URL-encoded (space=%20). The parameters must be sorted by name.
# For this example, the query string is pre-formatted in the request_parameters
# variable.
canonical_querystring = request_parameters
# Step 4: Create the canonical headers and signed headers. Header names
# must be trimmed and lowercase, and sorted in code point order from
# low to high. Note that there is a trailing \n.
canonical_headers = 'host:' + host + '\n' + 'x-amz-date:' + amzdate + '\n'
# Step 5: Create the list of signed headers. This lists the headers
# in the canonical_headers list, delimited with ";" and in alpha order.
# Note: The request can include any headers; canonical_headers and
# signed_headers lists those that you want to be included in the
# hash of the request. "Host" and "x-amz-date" are always required.
signed_headers = 'host;x-amz-date'
# Step 6: Create payload hash (hash of the request body content). For GET
# requests, the payload is an empty string ("").
payload_hash = hashlib.sha256('').encode('utf-8')).hexdigest()
# Step 7: Combine elements to create canonical request
canonical_request = method + '\n' + canonical_uri + '\n' + canonical_querystring + '\n'
+ canonical_headers + '\n' + signed_headers + '\n' + payload_hash

# ***** TASK 2: CREATE THE STRING TO SIGN*****
# Match the algorithm to the hashing algorithm you use, either SHA-1 or
# SHA-256 (recommended)
algorithm = 'AWS4-HMAC-SHA256'
credential_scope = datestamp + '/' + region + '/' + service + '/' + 'aws4_request'
string_to_sign = algorithm + '\n' + amzdate + '\n' + credential_scope + '\n' +
    hashlib.sha256(canonical_request.encode('utf-8')).hexdigest()
# ***** TASK 3: CALCULATE THE SIGNATURE *****
# Create the signing key using the function defined above.
signing_key = getSignatureKey(secret_key, datestamp, region, service)
# Sign the string_to_sign using the signing_key
signature = hmac.new(signing_key, (string_to_sign).encode('utf-8'),
    hashlib.sha256).hexdigest()

# ***** TASK 4: ADD SIGNING INFORMATION TO THE REQUEST *****
# The signing information can be either in a query string value or in
# a header named Authorization. This code shows how to use a header.
# Create authorization header and add to request headers
authorization_header = algorithm + ' ' + 'Credential=' + access_key + '/' +
    credential_scope + ', ' + 'SignedHeaders=' + signed_headers + ', ' + 'Signature=' +
    signature
# The request can include any headers, but MUST include "host", "x-amz-date",
# and (for this scenario) "Authorization". "host" and "x-amz-date" must

```

```
# be included in the canonical_headers and signed_headers, as noted
# earlier. Order here is not significant.
# Python note: The 'host' header is added automatically by the Python 'requests'
# library.
headers = {'x-amz-date':amzdate, 'Authorization':authorization_header}

# ***** SEND THE REQUEST *****
request_url = endpoint + '?' + canonical_querystring
print('\nBEGIN REQUEST+++++')
print('Request URL = ' + request_url)
response = requests.get(request_url, headers=headers)
print('\nRESPONSE+++++')
print('Response code: %d\n' % response.status_code)
print(response.text)

download_url = response.json()["LatestBk"]["DownloadURL"]
r = requests.get(download_url)
open('devicetester.zip', 'wb').write(r.content)
```

Utiliser IDT pour exécuter la suite de AWS IoT Greengrass qualifications

Vous pouvez utiliser AWS IoT Device Tester for AWS IoT Greengrass V2 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 déployer des composants et les mettre à niveau.

Outre les appareils de test, IDT for AWS IoT Greengrass V2 crée des ressources (par exemple, des AWS IoT objets, des groupes, etc.) dans votre Compte AWS corps afin de faciliter le processus de qualification.

Pour créer ces ressources, IDT for AWS IoT Greengrass V2 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.

Lorsque vous utilisez IDT for AWS IoT Greengrass V2 pour exécuter la suite de AWS IoT Greengrass qualification, celle-ci exécute les étapes suivantes :

1. Chargement et validation des informations d'identification et de la configuration 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 carte a réussi les tests obligatoires pour la qualification.

Versions de la suite de tests

IDT for AWS IoT Greengrass V2 organise les tests en suites de tests et en 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 test est l'ensemble de tests individuels liés à une fonctionnalité particulière, telle que les déploiements de composants.

Les suites de tests sont versionnées à l'aide d'un *major.minor.patch* format, par exemple `GGV2Q_1.0.0`. Lorsque vous téléchargez IDT, le package inclut la dernière version de la suite de qualification Greengrass.

Important

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. Pour de plus amples informations, veuillez consulter [the section called “Politique de support AWS IoT Device Tester pour AWS IoT Greengrass”](#).

Vous pouvez exécuter `list-supported-products` pour répertorier les versions AWS IoT Greengrass et les suites de tests prises en charge par votre version actuelle d'IDT.

Descriptions des groupes de tests

Groupes de test requis pour la qualification du noyau

Ces groupes de test sont nécessaires pour qualifier votre appareil AWS IoT Greengrass V2 pour le catalogue AWS Partner d'appareils.

Dépendances fondamentales

Vérifie que l'appareil répond à toutes les exigences logicielles et matérielles du logiciel AWS IoT Greengrass Core. Ce groupe de test inclut le cas de test suivant :

Version de Java

Vérifie que la version Java requise est installée sur l'appareil testé. AWS IoT Greengrass nécessite Java 8 ou une version ultérieure.

PreTest Validation

Vérifie que l'appareil répond à la configuration logicielle requise pour exécuter des tests.

- Pour les appareils basés sur Linux, ce test vérifie si le périphérique peut exécuter les commandes Linux suivantes :

`chmod, cp, echo, grep, kill, ln, mkinfo, ps, rm, sh, uname`

- Pour les appareils Windows, ce test vérifie si les logiciels Microsoft suivants sont installés sur l'appareil :

[Powershell v5.1 ou version ultérieure, .NET v4.6.1 ou version ultérieure, Visual C++ 2017 ou version ultérieure, utilitaire PsExec](#)

Vérificateur de version

Vérifie que la version AWS IoT Greengrass fournie est compatible avec la version de AWS IoT Device Tester que vous utilisez.

Composant

Vérifie que l'appareil peut déployer des composants et les mettre à niveau. Ce groupe de test comprend les tests suivants :

Composant cloud

Valide les capacités de l'appareil pour les composants du cloud.

Composant local

Valide la capacité de l'appareil pour les composants locaux.

Lambda

Ce test ne s'applique pas aux appareils Windows.

Vérifie que le périphérique peut déployer des composants de fonctions Lambda qui utilisent l'environnement d'exécution Java et que les fonctions Lambda peuvent AWS IoT Core utiliser des sujets MQTT comme sources d'événements pour les messages professionnels.

MQTT

Valide que l'appareil peut s'abonner et publier sur des sujets AWS IoT Core MQTT.

Groupes de test facultatifs

Note

Ces groupes de test sont facultatifs et utilisés uniquement pour les appareils principaux Greengrass basés sur Linux éligibles. Si vous choisissez de vous qualifier pour les tests facultatifs, votre appareil est répertorié avec des fonctionnalités supplémentaires dans le catalogue des AWS Partner appareils.

Dépendances de Docker

Vérifie que l'appareil répond à toutes les dépendances techniques requises pour utiliser le composant Docker application manager (`AWSaws.greengrass.DockerApplicationManager` fourni.

Qualification du gestionnaire d'applications Docker

Vérifie que l'appareil peut télécharger une image de conteneur Docker depuis Amazon ECR.

Dépendances du Machine Learning

Note

Le groupe de test optionnel d'apprentissage automatique n'est pris en charge que dans IDT v4.9.3.

Valide que l'appareil répond à toutes les dépendances techniques requises pour utiliser les composants d'apprentissage automatique (ML) AWS fournis.

Tests d'inférence pour le Machine Learning

Note

Le groupe de test optionnel d'apprentissage automatique n'est pris en charge que dans IDT v4.9.3.

Valide que le périphérique peut effectuer une inférence ML à l'aide des frameworks [Deep Learning Runtime](#) et [TensorFlow Lite](#) ML.

Dépendances du Stream Manager

Note

Le groupe de test optionnel du gestionnaire de flux n'est pris en charge que dans IDT v4.9.3.

Vérifie que l'appareil peut télécharger, installer et exécuter le [gestionnaire de AWS IoT Greengrass flux](#).

Intégration de sécurité matérielle (HSI)

Note

Ce test est disponible dans IDT v4.9.3 et versions ultérieures pour les appareils basés sur Linux uniquement. AWS IoT Greengrass ne prend actuellement pas en charge l'intégration de la sécurité matérielle pour les appareils Windows.

Vérifie que le périphérique peut authentifier les connexions aux AWS IoT Greengrass services AWS IoT et à l'aide d'une clé privée et d'un certificat stockés dans un module de sécurité matériel (HSM). Ce test vérifie également que le [composant fournisseur PKCS #11 AWS](#) fourni peut s'interfacer avec le HSM à l'aide d'une bibliothèque PKCS #11 fournie par le fournisseur. Pour de plus amples informations, veuillez consulter [Intégration de sécurité matérielle](#).

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) pour AWS IoT Greengrass

Téléchargez la dernière version de AWS IoT Device Tester for AWS IoT Greengrass

Téléchargez la [dernière version](#) d'IDT et extrayez le logiciel dans un emplacement (`<device-tester-extract-location>`) de votre système de fichiers où vous disposez d'autorisations de lecture/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.

Téléchargez le AWS IoT Greengrass logiciel

IDT for AWS IoT Greengrass V2 teste la compatibilité de votre appareil avec une version spécifique de AWS IoT Greengrass. Exécutez la commande suivante pour télécharger le logiciel AWS IoT Greengrass Core dans un fichier nommé `aws.greengrass.nucleus.zip`. *version* Remplacez-le par une [version de composant Nucleus prise en charge](#) pour votre version IDT.

Linux or Unix

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

Windows Command Prompt (CMD)

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

PowerShell

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

Placez le `aws.greengrass.nucleus.zip` fichier téléchargé dans le *<device-tester-extract-location>*/products/ dossier.

 Note

Ne placez pas plusieurs fichiers dans ce répertoire pour le même système d'exploitation et la même architecture.

Créez et configurez un Compte AWS

Avant de pouvoir utiliser AWS IoT Device Tester la AWS IoT Greengrass version V2, vous devez effectuer les étapes suivantes :

1. [Configurez 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 AWS des ressources, telles que AWS IoT des objets et AWS IoT Greengrass des composants, en votre nom.

Pour créer ces ressources, IDT for AWS IoT Greengrass V2 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 AWS gratuit](#), 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 : Configurez un Compte AWS

Dans cette étape, créez et configurez un Compte AWS. Si vous disposez déjà d'un Compte AWS, passez directement à l'étape [the section called “Étape 2 : Configurer les autorisations pour IDT”](#).

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

Pour vous inscrire à un Compte AWS

1. Ouvrez l'<https://portal.aws.amazon.com/billing/inscription>.

2. Suivez les instructions en ligne.

Dans le cadre de la procédure d'inscription, vous recevrez un appel téléphonique ou un SMS et vous saisissez 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 AWS est créé. Par défaut, seul l'utilisateur racine a accès à l'ensemble des Services AWS et des ressources de ce compte. La meilleure pratique de sécurité consiste à attribuer un accès administratif à un utilisateur, et à utiliser uniquement l'utilisateur racine pour effectuer les [tâches nécessitant un accès utilisateur racine](#).

Afin de créer un utilisateur administrateur, choisissez l'une des options suivantes :

Choisissez un moyen de gérer votre administrateur	Pour	En	Vous pouvez également
Dans IAM Identity Center (Recommandé)	Utiliser des identifiants à court terme pour accéder à AWS. C'est conforme aux bonnes pratiques en matière de sécurité. Pour plus d'informations sur les bonnes pratiques, consultez Bonnes pratiques de sécurité dans IAM dans le Guide de l'utilisateur IAM.	Suivant les instructions fournies dans Mise en route dans le Guide de l'utilisateur AWS IAM Identity Center .	Configurez l'accès par programmation en configurant le AWS CLI à utiliser AWS IAM Identity Center dans le guide de l'AWS Command Line Interface utilisateur.
Dans IAM	Utiliser les informations d'identification	Suivant les instructions fournies dans Création d'un	Configurer l'accès par programmation en suivant

Choisissez un moyen de gérer votre administrateur	Pour	En	Vous pouvez également
(Non recommandé)	à long terme pour accéder à AWS.	utilisateur IAM pour l'accès d'urgence dans le Guide de l'utilisateur IAM.	les instructions fournies dans Gestion des clés d'accès pour les utilisateurs IAM dans le Guide de l'utilisateur IAM.

Étape 2 : Configurer les autorisations pour IDT

Dans cette étape, configurez les autorisations utilisées par IDT pour AWS IoT Greengrass V2 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 de test pour IDT, passez directement à [Configurez votre appareil pour exécuter des tests IDT](#).

Pour configurer des autorisations pour IDT (console)

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. Si vous ne l'utilisez pas PreInstalled, dans l'onglet JSON, remplacez le contenu de l'espace réservé par la politique suivante. Si vous utilisez PreInstalled, passez à l'étape suivante.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "passRoleForResources",
      "Effect": "Allow",
```



```
"Action": "iam:PassRole",
"Resource": "arn:aws:iam::*:role/idt-*",
"Condition": {
  "StringEquals": {
    "iam:PassedToService": [
      "iot.amazonaws.com",
      "lambda.amazonaws.com",
      "greengrass.amazonaws.com"
    ]
  }
},
{
  "Sid": "lambdaResources",
  "Effect": "Allow",
  "Action": [
    "lambda:CreateFunction",
    "lambda:PublishVersion",
    "lambda:DeleteFunction",
    "lambda:GetFunction"
  ],
  "Resource": [
    "arn:aws:lambda::*:function:idt-*"
  ]
},
{
  "Sid": "iotResources",
  "Effect": "Allow",
  "Action": [
    "iot:CreateThing",
    "iot:DeleteThing",
    "iot:DescribeThing",
    "iot:CreateThingGroup",
    "iot:DeleteThingGroup",
    "iot:DescribeThingGroup",
    "iot:AddThingToThingGroup",
    "iot:RemoveThingFromThingGroup",
    "iot:AttachThingPrincipal",
    "iot:DetachThingPrincipal",
    "iot:UpdateCertificate",
    "iot:DeleteCertificate",
    "iot:CreatePolicy",
    "iot:AttachPolicy",
    "iot:DetachPolicy",
```

```

        "iot:DeletePolicy",
        "iot:GetPolicy",
        "iot:Publish",
        "iot:TagResource",
        "iot:ListThingPrincipals",
        "iot:ListAttachedPolicies",
        "iot:ListTargetsForPolicy",
        "iot:ListThingGroupsForThing",
        "iot:ListThingsInThingGroup",
        "iot:CreateJob",
        "iot:DescribeJob",
        "iot:DescribeJobExecution",
        "iot:CancelJob"
    ],
    "Resource": [
        "arn:aws:iot:*:*:thing/idt-*",
        "arn:aws:iot:*:*:thinggroup/idt-*",
        "arn:aws:iot:*:*:policy/idt-*",
        "arn:aws:iot:*:*:cert/*",
        "arn:aws:iot:*:*:topic/idt-*",
        "arn:aws:iot:*:*:job/*"
    ]
},
{
    "Sid": "s3Resources",
    "Effect": "Allow",
    "Action": [
        "s3:GetObject",
        "s3:PutObject",
        "s3:DeleteObjectVersion",
        "s3:DeleteObject",
        "s3:CreateBucket",
        "s3:ListBucket",
        "s3:ListBucketVersions",
        "s3:DeleteBucket",
        "s3:PutObjectTagging",
        "s3:PutBucketTagging"
    ],
    "Resource": "arn:aws:s3:*:*:idt-*"
},
{
    "Sid": "roleAliasResources",
    "Effect": "Allow",
    "Action": [

```

```
        "iot:CreateRoleAlias",
        "iot:DescribeRoleAlias",
        "iot>DeleteRoleAlias",
        "iot:TagResource",
        "iam:GetRole"
    ],
    "Resource": [
        "arn:aws:iot:*:*:rolealias/idt-*",
        "arn:aws:iam:*:*:role/idt-*"
    ]
},
{
    "Sid": "idtExecuteAndCollectMetrics",
    "Effect": "Allow",
    "Action": [
        "iot-device-tester:SendMetrics",
        "iot-device-tester:SupportedVersion",
        "iot-device-tester:LatestIdt",
        "iot-device-tester:CheckVersion",
        "iot-device-tester:DownloadTestSuite"
    ],
    "Resource": "*"
},
{
    "Sid": "genericResources",
    "Effect": "Allow",
    "Action": [
        "greengrass:*",
        "iot:GetThingShadow",
        "iot:UpdateThingShadow",
        "iot:ListThings",
        "iot:DescribeEndpoint",
        "iot:CreateKeysAndCertificate"
    ],
    "Resource": "*"
},
{
    "Sid": "iamResourcesUpdate",
    "Effect": "Allow",
    "Action": [
        "iam:CreateRole",
        "iam>DeleteRole",
        "iam:CreatePolicy",
        "iam>DeletePolicy",
```

```

        "iam:AttachRolePolicy",
        "iam:DetachRolePolicy",
        "iam:TagRole",
        "iam:TagPolicy",
        "iam:GetPolicy",
        "iam:ListAttachedRolePolicies",
        "iam:ListEntitiesForPolicy"
    ],
    "Resource": [
        "arn:aws:iam::*:role/idt-*",
        "arn:aws:iam::*:policy/idt-*"
    ]
}
]
}

```

- c. Si vous utilisez PreInstalled, dans l'onglet JSON, remplacez le contenu de l'espace réservé par la politique suivante. Assurez-vous de :
- Remplacez *thingName* et *thingGroup* dans l'iotResourcesinstruction par le nom de l'objet et le groupe d'objets créés lors de l'installation de Greengrass sur votre appareil testé (DUT) pour ajouter des autorisations.
 - Remplacez le *passRole* et *roleAlias* dans l'roleAliasResourcesinstruction et l'passRoleForResourcesinstruction par les rôles créés lors de l'installation de Greengrass sur votre DUT.

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "passRoleForResources",
      "Effect": "Allow",
      "Action": "iam:PassRole",
      "Resource": "arn:aws:iam::*:role/passRole",
      "Condition": {
        "StringEquals": {
          "iam:PassedToService": [
            "iot.amazonaws.com",
            "lambda.amazonaws.com",
            "greengrass.amazonaws.com"
          ]
        }
      }
    }
  ]
}

```

```
    }  
  }  
},  
{  
  "Sid": "lambdaResources",  
  "Effect": "Allow",  
  "Action": [  
    "lambda:CreateFunction",  
    "lambda:PublishVersion",  
    "lambda:DeleteFunction",  
    "lambda:GetFunction"  
  ],  
  "Resource": [  
    "arn:aws:lambda:*:*:function:idt-*"  
  ]  
},  
{  
  "Sid": "iotResources",  
  "Effect": "Allow",  
  "Action": [  
    "iot:CreateThing",  
    "iot:DeleteThing",  
    "iot:DescribeThing",  
    "iot:CreateThingGroup",  
    "iot:DeleteThingGroup",  
    "iot:DescribeThingGroup",  
    "iot:AddThingToThingGroup",  
    "iot:RemoveThingFromThingGroup",  
    "iot:AttachThingPrincipal",  
    "iot:DetachThingPrincipal",  
    "iot:UpdateCertificate",  
    "iot:DeleteCertificate",  
    "iot:CreatePolicy",  
    "iot:AttachPolicy",  
    "iot:DetachPolicy",  
    "iot:DeletePolicy",  
    "iot:GetPolicy",  
    "iot:Publish",  
    "iot:TagResource",  
    "iot:ListThingPrincipals",  
    "iot:ListAttachedPolicies",  
    "iot:ListTargetsForPolicy",  
    "iot:ListThingGroupsForThing",  
    "iot:ListThingsInThingGroup",  
  ]  
}
```

```

        "iot:CreateJob",
        "iot:DescribeJob",
        "iot:DescribeJobExecution",
        "iot:CancelJob"
    ],
    "Resource": [
        "arn:aws:iot:*:*:thing/thingName",
        "arn:aws:iot:*:*:thinggroup/thingGroup",
        "arn:aws:iot:*:*:policy/idx-*",
        "arn:aws:iot:*:*:cert/*",
        "arn:aws:iot:*:*:topic/idx-*",
        "arn:aws:iot:*:*:job/*"
    ]
},
{
    "Sid": "s3Resources",
    "Effect": "Allow",
    "Action": [
        "s3:GetObject",
        "s3:PutObject",
        "s3:DeleteObjectVersion",
        "s3:DeleteObject",
        "s3:CreateBucket",
        "s3:ListBucket",
        "s3:ListBucketVersions",
        "s3:DeleteBucket",
        "s3:PutObjectTagging",
        "s3:PutBucketTagging"
    ],
    "Resource": "arn:aws:s3:*:*:idx-*"
},
{
    "Sid": "roleAliasResources",
    "Effect": "Allow",
    "Action": [
        "iot:CreateRoleAlias",
        "iot:DescribeRoleAlias",
        "iot:DeleteRoleAlias",
        "iot:TagResource",
        "iam:GetRole"
    ],
    "Resource": [
        "arn:aws:iot:*:*:rolealias/roleAlias",
        "arn:aws:iam:*:*:role/idx-*"
    ]
}

```

```
]
},
{
  "Sid": "idtExecuteAndCollectMetrics",
  "Effect": "Allow",
  "Action": [
    "iot-device-tester:SendMetrics",
    "iot-device-tester:SupportedVersion",
    "iot-device-tester:LatestIdt",
    "iot-device-tester:CheckVersion",
    "iot-device-tester:DownloadTestSuite"
  ],
  "Resource": "*"
},
{
  "Sid": "genericResources",
  "Effect": "Allow",
  "Action": [
    "greengrass:*",
    "iot:GetThingShadow",
    "iot:UpdateThingShadow",
    "iot:ListThings",
    "iot:DescribeEndpoint",
    "iot:CreateKeysAndCertificate"
  ],
  "Resource": "*"
},
{
  "Sid": "iamResourcesUpdate",
  "Effect": "Allow",
  "Action": [
    "iam:CreateRole",
    "iam>DeleteRole",
    "iam:CreatePolicy",
    "iam>DeletePolicy",
    "iam:AttachRolePolicy",
    "iam:DetachRolePolicy",
    "iam:TagRole",
    "iam:TagPolicy",
    "iam:GetPolicy",
    "iam:ListAttachedRolePolicies",
    "iam:ListEntitiesForPolicy"
  ],
  "Resource": [
```

```
        "arn:aws:iam::*:role/idt-*",
        "arn:aws:iam::*:policy/idt-*"
    ]
}
]
```

 Note

Si vous souhaitez utiliser un rôle [IAM personnalisé comme rôle d'échange de jetons](#) pour votre appareil testé, assurez-vous de mettre à jour la `roleAliasResources` déclaration et la `passRoleForResources` déclaration de votre politique afin d'autoriser votre ressource de rôle IAM personnalisée.

- d. Choisissez Examiner une politique.
 - e. Pour Nom, saisissez **IDTGreengrassIAMPermissions**. Sous Résumé, vérifiez les autorisations accordées par votre stratégie.
 - f. Choisissez Create Policy (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 Attacher directement les stratégies existantes à l'utilisateur.
 - ii. Recherchez la IDTGreengrassIAMPermissionspolitique 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 (clé d'accès IDs et clés d'accès secrètes), choisissez Afficher à côté 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)

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 .
 - a. Si vous ne l'utilisez pas PreInstalled, ouvrez un éditeur de texte et enregistrez le contenu de la politique ci-dessous dans un fichier JSON. Si vous utilisez PreInstalled, passez à l'étape suivante.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "passRoleForResources",
      "Effect": "Allow",
      "Action": "iam:PassRole",
      "Resource": "arn:aws:iam::*:role/idt-*",
      "Condition": {
        "StringEquals": {
          "iam:PassedToService": [
            "iot.amazonaws.com",
            "lambda.amazonaws.com",
            "greengrass.amazonaws.com"
          ]
        }
      }
    }
  ]
},
```

```
{
  "Sid": "lambdaResources",
  "Effect": "Allow",
  "Action": [
    "lambda:CreateFunction",
    "lambda:PublishVersion",
    "lambda:DeleteFunction",
    "lambda:GetFunction"
  ],
  "Resource": [
    "arn:aws:lambda:*:*:function:idt-*"
  ]
},
{
  "Sid": "iotResources",
  "Effect": "Allow",
  "Action": [
    "iot:CreateThing",
    "iot:DeleteThing",
    "iot:DescribeThing",
    "iot:CreateThingGroup",
    "iot:DeleteThingGroup",
    "iot:DescribeThingGroup",
    "iot:AddThingToThingGroup",
    "iot:RemoveThingFromThingGroup",
    "iot:AttachThingPrincipal",
    "iot:DetachThingPrincipal",
    "iot:UpdateCertificate",
    "iot:DeleteCertificate",
    "iot:CreatePolicy",
    "iot:AttachPolicy",
    "iot:DetachPolicy",
    "iot:DeletePolicy",
    "iot:GetPolicy",
    "iot:Publish",
    "iot:TagResource",
    "iot:ListThingPrincipals",
    "iot:ListAttachedPolicies",
    "iot:ListTargetsForPolicy",
    "iot:ListThingGroupsForThing",
    "iot:ListThingsInThingGroup",
    "iot:CreateJob",
    "iot:DescribeJob",
    "iot:DescribeJobExecution",
```

```
    "iot:CancelJob"
  ],
  "Resource": [
    "arn:aws:iot:*:*:thing/idt-*",
    "arn:aws:iot:*:*:thinggroup/idt-*",
    "arn:aws:iot:*:*:policy/idt-*",
    "arn:aws:iot:*:*:cert/*",
    "arn:aws:iot:*:*:topic/idt-*",
    "arn:aws:iot:*:*:job/*"
  ]
},
{
  "Sid": "s3Resources",
  "Effect": "Allow",
  "Action": [
    "s3:GetObject",
    "s3:PutObject",
    "s3:DeleteObjectVersion",
    "s3:DeleteObject",
    "s3:CreateBucket",
    "s3:ListBucket",
    "s3:ListBucketVersions",
    "s3:DeleteBucket",
    "s3:PutObjectTagging",
    "s3:PutBucketTagging"
  ],
  "Resource": "arn:aws:s3:*:*:idt-*"
},
{
  "Sid": "roleAliasResources",
  "Effect": "Allow",
  "Action": [
    "iot:CreateRoleAlias",
    "iot:DescribeRoleAlias",
    "iot>DeleteRoleAlias",
    "iot:TagResource",
    "iam:GetRole"
  ],
  "Resource": [
    "arn:aws:iot:*:*:rolealias/idt-*",
    "arn:aws:iam:*:*:role/idt-*"
  ]
},
{
```

```
    "Sid": "idtExecuteAndCollectMetrics",
    "Effect": "Allow",
    "Action": [
        "iot-device-tester:SendMetrics",
        "iot-device-tester:SupportedVersion",
        "iot-device-tester:LatestIdt",
        "iot-device-tester:CheckVersion",
        "iot-device-tester:DownloadTestSuite"
    ],
    "Resource": "*"
},
{
    "Sid": "genericResources",
    "Effect": "Allow",
    "Action": [
        "greengrass:*",
        "iot:GetThingShadow",
        "iot:UpdateThingShadow",
        "iot:ListThings",
        "iot:DescribeEndpoint",
        "iot:CreateKeysAndCertificate"
    ],
    "Resource": "*"
},
{
    "Sid": "iamResourcesUpdate",
    "Effect": "Allow",
    "Action": [
        "iam:CreateRole",
        "iam:DeleteRole",
        "iam:CreatePolicy",
        "iam:DeletePolicy",
        "iam:AttachRolePolicy",
        "iam:DetachRolePolicy",
        "iam:TagRole",
        "iam:TagPolicy",
        "iam:GetPolicy",
        "iam:ListAttachedRolePolicies",
        "iam:ListEntitiesForPolicy"
    ],
    "Resource": [
        "arn:aws:iam::*:role/idt-*",
        "arn:aws:iam::*:policy/idt-*"
    ]
}
```

```

    }
  ]
}

```

- b. Si vous en utilisez un PreInstalled, ouvrez un éditeur de texte et enregistrez le contenu de la politique ci-dessous dans un fichier JSON. Assurez-vous de :
- Remplacez *thingName* et *thingGroup* dans la *iotResources* déclaration créée lors de l'installation de Greengrass sur votre appareil testé (DUT) pour ajouter des autorisations.
 - Remplacez le *passRole* et *roleAlias* dans l'*roleAliasResourcesinstruction* et l'*passRoleForResourceinstruction* par les rôles créés lors de l'installation de Greengrass sur votre DUT.

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "passRoleForResource",
      "Effect": "Allow",
      "Action": "iam:PassRole",
      "Resource": "arn:aws:iam::*:role/passRole",
      "Condition": {
        "StringEquals": {
          "iam:PassedToService": [
            "iot.amazonaws.com",
            "lambda.amazonaws.com",
            "greengrass.amazonaws.com"
          ]
        }
      }
    },
    {
      "Sid": "lambdaResources",
      "Effect": "Allow",
      "Action": [
        "lambda:CreateFunction",
        "lambda:PublishVersion",
        "lambda:DeleteFunction",
        "lambda:GetFunction"
      ]
    }
  ]
}

```

```
"Resource":[
  "arn:aws:lambda:*:*:function:idt-*"
],
{
  "Sid":"iotResources",
  "Effect":"Allow",
  "Action":[
    "iot:CreateThing",
    "iot>DeleteThing",
    "iot:DescribeThing",
    "iot:CreateThingGroup",
    "iot>DeleteThingGroup",
    "iot:DescribeThingGroup",
    "iot:AddThingToThingGroup",
    "iot:RemoveThingFromThingGroup",
    "iot:AttachThingPrincipal",
    "iot:DetachThingPrincipal",
    "iot:UpdateCertificate",
    "iot>DeleteCertificate",
    "iot:CreatePolicy",
    "iot:AttachPolicy",
    "iot:DetachPolicy",
    "iot>DeletePolicy",
    "iot:GetPolicy",
    "iot:Publish",
    "iot:TagResource",
    "iot:ListThingPrincipals",
    "iot:ListAttachedPolicies",
    "iot:ListTargetsForPolicy",
    "iot:ListThingGroupsForThing",
    "iot:ListThingsInThingGroup",
    "iot:CreateJob",
    "iot:DescribeJob",
    "iot:DescribeJobExecution",
    "iot:CancelJob"
  ],
  "Resource":[
    "arn:aws:iot:*:*:thing/thingName",
    "arn:aws:iot:*:*:thinggroup/thingGroup",
    "arn:aws:iot:*:*:policy/idt-*",
    "arn:aws:iot:*:*:cert/*",
    "arn:aws:iot:*:*:topic/idt-*",
    "arn:aws:iot:*:*:job/*"
```

```

    ]
  },
  {
    "Sid": "s3Resources",
    "Effect": "Allow",
    "Action": [
      "s3:GetObject",
      "s3:PutObject",
      "s3:DeleteObjectVersion",
      "s3:DeleteObject",
      "s3:CreateBucket",
      "s3:ListBucket",
      "s3:ListBucketVersions",
      "s3:DeleteBucket",
      "s3:PutObjectTagging",
      "s3:PutBucketTagging"
    ],
    "Resource": "arn:aws:s3::*:idt-*"
  },
  {
    "Sid": "roleAliasResources",
    "Effect": "Allow",
    "Action": [
      "iot:CreateRoleAlias",
      "iot:DescribeRoleAlias",
      "iot:DeleteRoleAlias",
      "iot:TagResource",
      "iam:GetRole"
    ],
    "Resource": [
      "arn:aws:iot::*:rolealias/roleAlias",
      "arn:aws:iam::*:role/idt-*"
    ]
  },
  {
    "Sid": "idtExecuteAndCollectMetrics",
    "Effect": "Allow",
    "Action": [
      "iot-device-tester:SendMetrics",
      "iot-device-tester:SupportedVersion",
      "iot-device-tester:LatestIdt",
      "iot-device-tester:CheckVersion",
      "iot-device-tester:DownloadTestSuite"
    ]
  },

```

```
    "Resource": "*"
  },
  {
    "Sid": "genericResources",
    "Effect": "Allow",
    "Action": [
      "greengrass:*",
      "iot:GetThingShadow",
      "iot:UpdateThingShadow",
      "iot:ListThings",
      "iot:DescribeEndpoint",
      "iot:CreateKeysAndCertificate"
    ],
    "Resource": "*"
  },
  {
    "Sid": "iamResourcesUpdate",
    "Effect": "Allow",
    "Action": [
      "iam:CreateRole",
      "iam:DeleteRole",
      "iam:CreatePolicy",
      "iam:DeletePolicy",
      "iam:AttachRolePolicy",
      "iam:DetachRolePolicy",
      "iam:TagRole",
      "iam:TagPolicy",
      "iam:GetPolicy",
      "iam:ListAttachedRolePolicies",
      "iam:ListEntitiesForPolicy"
    ],
    "Resource": [
      "arn:aws:iam::*:role/idt-*",
      "arn:aws:iam::*:policy/idt-*"
    ]
  }
]
```

 Note

Si vous souhaitez utiliser un rôle [IAM personnalisé comme rôle d'échange de jetons](#) pour votre appareil testé, assurez-vous de mettre à jour la `roleAliasResources`

déclaration et la `passRoleForResources` déclaration de votre politique afin d'autoriser votre ressource de rôle IAM personnalisée.

- c. Exécutez la commande suivante pour créer une politique gérée par le client nommée `IDTGreengrassIAMPermissions`. *policy.json* Remplacez-le par le chemin complet du fichier JSON que vous avez créé à l'étape précédente.

```
aws iam create-policy --policy-name IDTGreengrassIAMPermissions --policy-document file://policy.json
```

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
```

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 IoT Device Tester autorisations

Les politiques suivantes décrivent AWS IoT Device Tester les autorisations.

AWS IoT Device Tester nécessite ces autorisations pour les fonctionnalités de vérification des versions et de mise à jour automatique.

- `iot-device-tester:SupportedVersion`

Accorde AWS IoT Device Tester l'autorisation de récupérer la liste des produits pris en charge, des suites de tests et des versions IDT.

- `iot-device-tester:LatestIdt`

Accorde AWS IoT Device Tester l'autorisation de récupérer la dernière version d'IDT disponible au téléchargement.

- `iot-device-tester:CheckVersion`

Accorde AWS IoT Device Tester l'autorisation de vérifier la compatibilité des versions pour IDT, les suites de tests et les produits.

- `iot-device-tester:DownloadTestSuite`

AWS IoT Device Tester Autorise le téléchargement des mises à jour des suites de tests.

AWS IoT Device Tester utilise également l'autorisation suivante pour les rapports de mesures facultatifs :

- `iot-device-tester:SendMetrics`

Accorde l'autorisation AWS de collecter des statistiques sur l'utilisation AWS IoT Device Tester interne. Si cette autorisation est omise, ces statistiques ne seront pas collectées.

Configurez votre appareil pour exécuter des tests IDT

Pour permettre à IDT d'exécuter des tests de qualification des appareils, vous devez configurer votre ordinateur hôte pour accéder à votre appareil et configurer les autorisations utilisateur sur votre appareil.

Installation de Java sur l'ordinateur hôte

À partir de la version 4.2.0 d'IDT, les tests de qualification facultatifs AWS IoT Greengrass nécessitent l'exécution de Java.

Vous pouvez utiliser la version 8 ou supérieure de Java. Nous vous recommandons d'utiliser les versions de support à long terme d'[Amazon Corretto](#) ou d'OpenJDK. La version 8 ou supérieure est requise.

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 de plus amples informations, veuillez consulter [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* sont le nom d'utilisateur utilisé pour vous connecter à votre appareil testé et *remote-device-ip* l'adresse IP de l'appareil testé sur lequel effectuer les tests ? 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 en cours de test.
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 depuis la TTYgen fenêtre Pu.
 - 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 de plus amples informations, veuillez consulter [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.

Configuration des informations d'identification utilisateur pour les appareils Windows

Pour qualifier un appareil Windows, vous devez configurer les informations d'identification utilisateur dans le LocalSystem compte de l'appareil testé pour les utilisateurs suivants :

- L'utilisateur Greengrass par défaut ()ggc_user.
- L'utilisateur que vous utilisez pour vous connecter à l'appareil testé. Vous configurez cet utilisateur dans le [device.jsonfichier](#).

Vous devez créer chaque utilisateur du LocalSystem compte sur l'appareil testé, puis enregistrer le nom d'utilisateur et le mot de passe de l'utilisateur dans l'instance Credential Manager du LocalSystem compte.

Pour configurer les utilisateurs sur les appareils Windows

1. Ouvrez l'invite de commande Windows (cmd . exe) en tant qu'administrateur.
2. Créez les utilisateurs dans le LocalSystem compte sur l'appareil Windows. Exécutez la commande suivante pour chaque utilisateur que vous souhaitez créer. Pour l'utilisateur Greengrass par défaut, remplacez par *user-name*. ggc_user *password* Remplacez-le par un mot de passe sécurisé.

```
net user /add user-name password
```

3. Téléchargez et installez l'[PsExec utilitaire](#) de Microsoft sur l'appareil.
4. Utilisez l' PsExec utilitaire pour stocker le nom d'utilisateur et le mot de passe de l'utilisateur par défaut dans l'instance Credential Manager du LocalSystem compte.

Exécutez la commande suivante pour chaque utilisateur que vous souhaitez configurer dans Credential Manager. Pour l'utilisateur Greengrass par défaut, remplacez par *user-name*. ggc_user *password* Remplacez-le par le mot de passe utilisateur que vous avez défini précédemment.

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

S'il PsExec License Agreements'ouvre, choisissez Acceptd'accepter la licence et exécutez la commande.

Note

Sur les appareils Windows, le LocalSystem compte exécute le noyau Greengrass, et vous devez utiliser l' PsExec utilitaire pour stocker les informations utilisateur dans le LocalSystem compte. L'application Credential Manager stocke ces informations dans le compte Windows de l'utilisateur actuellement connecté, plutôt que dans le LocalSystem compte.

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 V2 doit être capable d'exécuter des commandes avec sudo sans qu'un mot de passe ne soit demandé.

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 un rôle d'échange de jetons personnalisé

Vous pouvez choisir d'utiliser un rôle IAM personnalisé comme rôle d'échange de jetons que le périphérique testé suppose d'interagir avec les AWS ressources. Pour plus d'informations sur la création d'un rôle IAM, consultez la section [Création de rôles IAM](#) dans le Guide de l'utilisateur IAM.

Vous devez satisfaire aux exigences suivantes pour permettre à IDT d'utiliser votre rôle IAM personnalisé. Nous vous recommandons vivement de n'ajouter que le minimum d'actions de stratégie requises à ce rôle.

- Le fichier de configuration [userdata.json](#) doit être mis à jour pour que le `GreengrassV2TokenExchangeRole` paramètre soit défini sur `true`
- Le rôle IAM personnalisé doit être configuré avec la politique de confiance minimale suivante :
JSON

```
{
```

```
"Version":"2012-10-17",
"Statement":[
  {
    "Effect":"Allow",
    "Principal":{"
      "Service":[
        "credentials.iot.amazonaws.com",
        "lambda.amazonaws.com",
        "sagemaker.amazonaws.com"
      ]
    },
    "Action":"sts:AssumeRole"
  }
]
```

- Le rôle IAM personnalisé doit être configuré selon la politique d'autorisation minimale suivante :

JSON

```
{
  "Version":"2012-10-17",
  "Statement":[
    {
      "Effect":"Allow",
      "Action":[
        "iot:DescribeCertificate",
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams",
        "iot:Connect",
        "iot:Publish",
        "iot:Subscribe",
        "iot:Receive",
        "iot:ListThingPrincipals",
        "iot:GetThingShadow",
        "iot:UpdateThingShadow",
        "s3:GetBucketLocation",
        "s3:GetObject",
        "s3:PutObject",
        "s3:AbortMultipartUpload",
        "s3:ListMultipartUploadParts"
      ]
    }
  ],
```



```

        "Resource": "*"
    }
]
}

```

- Le nom du rôle IAM personnalisé doit correspondre à la ressource de rôle IAM que vous spécifiez dans les autorisations IAM pour l'utilisateur de test. Par défaut, la [politique des utilisateurs de test](#) autorise l'accès aux rôles IAM dont le nom de rôle contient le `idt-` préfixe. Si le nom de votre rôle IAM n'utilise pas ce préfixe, ajoutez la `arn:aws:iam::*:role/custom-iam-role-name` ressource à l'`roleAliasResources` instruction et à l'`passRoleForResources` instruction dans votre politique utilisateur de test, comme indiqué dans les exemples suivants :

Exemple **passRoleForResources** déclaration

```

{
  "Sid": "passRoleForResources",
  "Effect": "Allow",
  "Action": "iam:PassRole",
  "Resource": "arn:aws:iam::*:role/custom-iam-role-name",
  "Condition": {
    "StringEquals": {
      "iam:PassedToService": [
        "iot.amazonaws.com",
        "lambda.amazonaws.com",
        "greengrass.amazonaws.com"
      ]
    }
  }
}

```

Exemple **roleAliasResources** déclaration

```

{
  "Sid": "roleAliasResources",
  "Effect": "Allow",
  "Action": [
    "iot:CreateRoleAlias",
    "iot:DescribeRoleAlias",
    "iot>DeleteRoleAlias",
    "iot:TagResource",
    "iam:GetRole"
  ]
}

```

```
],
"Resource":[
  "arn:aws:iot:*:*:rolealias/idt-*",
  "arn:aws:iam:*:*:role/custom-iam-role-name"
]
}
```

Configurer votre appareil pour tester des fonctions facultatives

Cette section décrit la configuration requise pour exécuter des tests IDT pour les fonctionnalités optionnelles de Docker et d'apprentissage automatique (ML). Les fonctionnalités ML ne sont prises en charge que dans IDT v4.9.3. Vous devez vous assurer que votre appareil répond à ces exigences uniquement si vous souhaitez tester ces fonctionnalités. Dans le cas contraire, passez à [the section called “Configurer les paramètres IDT”](#).

Rubriques

- [Exigences de qualification pour Docker](#)
- [Exigences de qualification ML](#)
- [Exigences de qualification HSM](#)

Exigences de qualification pour Docker

IDT for AWS IoT Greengrass V2 propose des tests de qualification Docker pour valider que vos appareils peuvent utiliser le composant du [gestionnaire d'applications Docker AWS](#) fourni pour télécharger des images de conteneur Docker que vous pouvez exécuter à l'aide de composants de conteneur Docker personnalisés. Pour plus d'informations sur la création de composants Docker personnalisés, consultez [Exécuter un conteneur Docker](#).

Pour exécuter les tests de qualification Docker, vos appareils testés doivent répondre aux exigences suivantes pour déployer le composant du gestionnaire d'applications Docker.

- [Docker Engine](#) 1.9.1 ou version ultérieure installé sur le périphérique principal de Greengrass. La version 20.10 est la dernière version vérifiée pour fonctionner avec le logiciel AWS IoT Greengrass Core. Vous devez installer Docker directement sur le périphérique principal avant de déployer des composants qui exécutent des conteneurs Docker.
- Le daemon Docker a démarré et s'est exécuté sur le périphérique principal avant que vous ne déployiez ce composant.

- L'utilisateur du système qui exécute un composant de conteneur Docker doit disposer des autorisations root ou administrateur, ou vous devez configurer Docker pour l'exécuter en tant qu'utilisateur non root ou non administrateur.
- Sur les appareils Linux, vous pouvez ajouter un utilisateur au `docker` groupe sans lequel vous pouvez appeler `docker` des commandes `sudo`.
- Sur les appareils Windows, vous pouvez ajouter un utilisateur au `docker-users` groupe pour appeler des `docker` commandes sans privilèges d'administrateur.

Linux or Unix

Pour ajouter `ggc_user` au `docker` groupe l'utilisateur non root que vous utilisez pour exécuter les composants du conteneur Docker, exécutez la commande suivante.

```
sudo usermod -aG docker ggc_user
```

Pour plus d'informations, consultez [Gérer Docker en tant qu'utilisateur non root](#).

Windows Command Prompt (CMD)

Pour ajouter `ggc_user` au `docker-users` groupe l'utilisateur que vous utilisez pour exécuter les composants du conteneur Docker, exécutez la commande suivante en tant qu'administrateur.

```
net localgroup docker-users ggc_user /add
```

Windows PowerShell

Pour ajouter `ggc_user` au `docker-users` groupe l'utilisateur que vous utilisez pour exécuter les composants du conteneur Docker, exécutez la commande suivante en tant qu'administrateur.

```
Add-LocalGroupMember -Group docker-users -Member ggc_user
```

Exigences de qualification ML

Note

La fonctionnalité d'apprentissage automatique n'est prise en charge que dans IDT v4.9.3.

IDT for AWS IoT Greengrass V2 propose des tests de qualification ML pour valider que vos appareils peuvent utiliser les [composants d'apprentissage automatique AWS](#) fournis pour effectuer une inférence ML localement à l'aide des frameworks [Deep Learning Runtime](#) ou [TensorFlow Lite](#) ML. Pour plus d'informations sur l'exécution de l'inférence ML sur les appareils Greengrass, consultez [Exécuter l'inférence de Machine Learning](#).

Pour exécuter des tests de qualification ML, vos appareils testés doivent répondre aux exigences suivantes pour déployer les composants d'apprentissage automatique.

- Sur les appareils principaux de Greengrass exécutant Amazon Linux 2 ou Ubuntu 18.04, la version 2.27 ou ultérieure de la [bibliothèque GNU C](#) (glibc) est installée sur l'appareil.
- Sur les appareils ARMv7L, tels que le Raspberry Pi, les dépendances pour OpenCV-Python sont installées sur l'appareil. Exécutez la commande suivante pour installer les dépendances.

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev  
libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

- Les appareils Raspberry Pi qui exécutent le système d'exploitation Raspberry Pi Bullseye doivent répondre aux exigences suivantes :
 - NumPy 1.22.4 ou version ultérieure installée sur l'appareil. Raspberry Pi OS Bullseye inclut une version antérieure de NumPy. Vous pouvez donc exécuter la commande suivante pour effectuer la mise à niveau NumPy sur l'appareil.

```
pip3 install --upgrade numpy
```

- L'ancienne pile de caméras activée sur l'appareil. Raspberry Pi OS Bullseye inclut une nouvelle pile de caméras activée par défaut et non compatible. Vous devez donc activer la pile de caméras existante.

Pour activer l'ancienne pile de caméras

1. Exécutez la commande suivante pour ouvrir l'outil de configuration du Raspberry Pi.

```
sudo raspi-config
```

2. Sélectionnez Options d'interface.
3. Sélectionnez Legacy camera pour activer l'ancienne pile de caméras.
4. Redémarrez l'appareil Raspberry Pi.

Exigences de qualification HSM

AWS IoT Greengrass fournit un [composant fournisseur PKCS #11](#) à intégrer au module de sécurité matérielle (HSM) PKCS du périphérique. La configuration du HSM dépend de votre appareil et du module HSM que vous avez choisi. Tant que la configuration HSM attendue, telle que décrite dans les [paramètres de configuration IDT](#), est fournie, IDT disposera des informations nécessaires pour exécuter ce test de qualification des fonctionnalités optionnel.

Configurer les paramètres IDT pour exécuter la suite de AWS IoT Greengrass qualification

Avant d'exécuter les tests, vous devez configurer les paramètres des AWS informations d'identification et des périphériques sur votre ordinateur hôte.

Configurer les AWS informations d'identification dans config.json

Vous devez configurer vos informations d'identification d'utilisateur IAM dans le `<device_tester_extract_location>/configs/config.json` fichier. Utilisez les informations d'identification de l'utilisateur IDT pour AWS IoT Greengrass V2 créé dans [the section called "Créez et configurez un Compte AWS"](#). Vous pouvez spécifier vos informations d'identification de deux manières :

- Dans un fichier d'informations d'identification
- En tant que variables d'environnement

Configuration des AWS informations d'identification à l'aide d'un fichier 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`

Ajoutez vos AWS informations d'identification au `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 pour AWS IoT Greengrass V2 afin AWS d'utiliser les informations d'identification de votre `credentials` fichier, modifiez votre `config.json` fichier comme suit :

```
{
  "awsRegion": "region",
  "auth": {
    "method": "file",
    "credentials": {
      "profile": "default"
    }
  }
}
```

Note

Si vous n'utilisez pas le default AWS profil, veuillez à modifier le nom du profil dans votre `config.json` fichier. Pour de plus amples informations, veuillez consulter [Profils nommés](#).

Configurer les AWS informations 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 pour AWS IoT Greengrass V2 peut utiliser les variables d'AWS_SECRET_ACCESS_KEY environnement AWS_ACCESS_KEY_ID et pour stocker vos AWS informations 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": "region",
  "auth": {
    "method": "environment"
  }
}
```

Configurer `device.json`

Note

IDT v4.9.3 permet de tester les fonctionnalités `mldocker`, et `streamManagement` IDT v4.9.4 et versions ultérieures prennent en charge les tests `docker`. Si vous ne souhaitez pas tester ces fonctionnalités, définissez la valeur correspondante `urno`.

Outre les AWS informations d'identification, IDT pour AWS IoT Greengrass V2 a besoin d'informations sur les appareils sur lesquels les tests sont exécutés. Des exemples d'informations pourraient être l'adresse IP, les informations de connexion, le système d'exploitation et l'architecture du processeur.

Vous pouvez fournir ces informations à l'aide du modèle `device.json` situé dans `<device_tester_extract_location>/configs/device.json` :

IDT v4.9.3

```
[
  {
    "id": "<pool-id>",
    "sku": "<sku>",
    "features": [
      {
        "name": "arch",
        "value": "x86_64 | armv6l | armv7l | aarch64"
      },
      {
        "name": "ml",
```

```

    "value": "dlr | tensorflowlite | dlr,tensorflowlite | no"
  },
  {
    "name": "docker",
    "value": "yes | no"
  },
  {
    "name": "streamManagement",
    "value": "yes | no"
  },
  {
    "name": "hsi",
    "value": "hsm | no"
  }
],
"devices": [
  {
    "id": "<device-id>",
    "operatingSystem": "Linux | Windows",
    "connectivity": {
      "protocol": "ssh",
      "ip": "<ip-address>",
      "port": 22,
      "publicKeyPath": "<public-key-path>",
      "auth": {
        "method": "pki | password",
        "credentials": {
          "user": "<user-name>",
          "privKeyPath": "/path/to/private/key",
          "password": "<password>"
        }
      }
    }
  }
]
}
]

```

Note

Spécifiez `privKeyPath` uniquement si `method` est défini sur `pki`.
 Spécifiez `password` uniquement si `method` est défini sur `password`.

Toutes les propriétés contenant des valeurs sont obligatoires, comme décrit 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 répertorier votre appareil dans le catalogue des AWS Partner appareils, le SKU que vous spécifiez ici doit correspondre au SKU que vous utilisez dans le processus de mise en vente.

features

Un tableau contenant les fonctions prises en charge de l'appareil. Toutes les fonctionnalités sont requises.

arch

Les architectures de système d'exploitation prises en charge validées par le test. Les valeurs valides sont :

- x86_64
- armv6l
- armv7l
- aarch64

ml

Valide que l'appareil répond à toutes les dépendances techniques requises pour utiliser les composants d'apprentissage automatique (ML) AWS fournis.

L'activation de cette fonctionnalité confirme également que l'appareil peut effectuer une inférence ML à l'aide des frameworks [Deep Learning Runtime](#) et [TensorFlow Lite](#) ML.

Les valeurs valides sont n'importe quelle combinaison `tensorflowlite` de `dlr` et `ouno`.
`docker`

Vérifie que l'appareil répond à toutes les dépendances techniques requises pour utiliser le composant Docker application manager ()
`AWSaws.greengrass.DockerApplicationManager` fourni.

L'activation de cette fonctionnalité confirme également que l'appareil peut télécharger une image de conteneur Docker depuis Amazon ECR.

Les valeurs valides sont n'importe quelle combinaison de `yes` ou `no`.

`streamManagement`

Vérifie que l'appareil peut télécharger, installer et exécuter le [gestionnaire de AWS IoT Greengrass flux](#).

Les valeurs valides sont n'importe quelle combinaison de `yes` ou `no`.

`hsi`

Valide que le périphérique peut authentifier les connexions aux AWS IoT Greengrass services AWS IoT et à l'aide d'une clé privée et d'un certificat stockés dans un module de sécurité matériel (HSM). Ce test vérifie également que le [composant fournisseur PKCS #11 AWS](#) fourni peut s'interfacer avec le HSM à l'aide d'une bibliothèque PKCS #11 fournie par le fournisseur. Pour de plus amples informations, veuillez consulter [Intégration de sécurité matérielle](#).

Les valeurs valides sont `hsm` ou `no`.

Note

Le test `n'hsi` est disponible qu'avec IDT v4.9.3 et versions ultérieures.

`devices.id`

Un identificateur unique défini par l'utilisateur pour l'appareil testé.

`devices.operatingSystem`

Le système d'exploitation de l'appareil. Les valeurs prises en charge sont Linux et Windows.

`connectivity.protocol`

Le protocole de communication utilisé pour communiquer avec cet appareil. Actuellement, la seule valeur prise en charge `ssh` concerne les appareils physiques.

`connectivity.ip`

L'adresse IP de l'appareil testé.

Cette propriété s'applique uniquement si `connectivity.protocol` est défini sur `ssh`.

`connectivity.port`

Facultatif. Le 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.publicKeyPath`

Facultatif. Le chemin complet vers la clé publique utilisée pour authentifier les connexions à l'appareil testé.

Lorsque vous spécifiez `lepublicKeyPath`, IDT valide la clé publique de l'appareil lorsqu'il établit une connexion SSH avec le périphérique testé. Si cette valeur n'est pas spécifiée, IDT crée une connexion SSH, mais ne valide pas la clé publique de l'appareil.

Nous vous recommandons vivement de spécifier le chemin d'accès à la clé publique et d'utiliser une méthode sécurisée pour récupérer cette clé publique. Pour les clients SSH standard basés sur une ligne de commande, la clé publique est fournie dans le `known_hosts` fichier. Si vous spécifiez un fichier de clé publique distinct, ce fichier doit utiliser le même format que le `known_hosts` fichier, c'est-à-dire *ip-address key-type public-key*. S'il existe plusieurs entrées avec la même adresse IP, l'entrée correspondant au type de clé utilisé par IDT doit se trouver avant les autres entrées du fichier.

`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.

IDT v4.9.4

```
[
  {
    "id": "<pool-id>",
    "sku": "<sku>",
    "features": [
      {
        "name": "arch",
        "value": "x86_64 | armv61 | armv71 | aarch64"
      },
      {
        "name": "docker",
```

```

    "value": "yes | no"
  },
  {
    "name": "hsi",
    "value": "hsm | no"
  }
],
"devices": [
  {
    "id": "<device-id>",
    "operatingSystem": "Linux | Windows",
    "connectivity": {
      "protocol": "ssh",
      "ip": "<ip-address>",
      "port": 22,
      "publicKeyPath": "<public-key-path>",
      "auth": {
        "method": "pki | password",
        "credentials": {
          "user": "<user-name>",
          "privKeyPath": "/path/to/private/key",
          "password": "<password>"
        }
      }
    }
  }
]
}
]

```

Note

Spécifiez `privKeyPath` uniquement si `method` est défini sur `pki`.
 Spécifiez `password` uniquement si `method` est défini sur `password`.

Toutes les propriétés contenant des valeurs sont obligatoires, comme décrit 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 répertorier votre appareil dans le catalogue des AWS Partner appareils, le SKU que vous spécifiez ici doit correspondre au SKU que vous utilisez dans le processus de mise en vente.

`features`

Un tableau contenant les fonctions prises en charge de l'appareil. Toutes les fonctionnalités sont requises.

`arch`

Les architectures de système d'exploitation prises en charge validées par le test. Les valeurs valides sont :

- `x86_64`
- `armv6l`
- `armv7l`
- `aarch64`

`docker`

Vérifie que l'appareil répond à toutes les dépendances techniques requises pour utiliser le composant Docker application manager (`AWSaws.greengrass.DockerApplicationManager` fourni.

L'activation de cette fonctionnalité confirme également que l'appareil peut télécharger une image de conteneur Docker depuis Amazon ECR.

Les valeurs valides sont n'importe quelle combinaison de `yes` ou `no`.

hsi

Valide que le périphérique peut authentifier les connexions aux AWS IoT Greengrass services AWS IoT et à l'aide d'une clé privée et d'un certificat stockés dans un module de sécurité matériel (HSM). Ce test vérifie également que le [composant fournisseur PKCS #11 AWS](#) fourni peut s'interfacer avec le HSM à l'aide d'une bibliothèque PKCS #11 fournie par le fournisseur. Pour de plus amples informations, veuillez consulter [Intégration de sécurité matérielle](#).

Les valeurs valides sont hsm ou no.

Note

Le test n'hsi est disponible qu'avec IDT v4.9.3 et versions ultérieures.

devices.id

Un identificateur unique défini par l'utilisateur pour l'appareil testé.

devices.operatingSystem

Le système d'exploitation de l'appareil. Les valeurs prises en charge sont Linux et Windows.

connectivity.protocol

Le protocole de communication utilisé pour communiquer avec cet appareil. Actuellement, la seule valeur prise en charge ssh concerne les appareils physiques.

connectivity.ip

L'adresse IP de l'appareil testé.

Cette propriété s'applique uniquement si `connectivity.protocol` est défini sur `ssh`.

connectivity.port

Facultatif. Le 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.publicKeyPath`

Facultatif. Le chemin complet vers la clé publique utilisée pour authentifier les connexions à l'appareil testé.

Lorsque vous spécifiez `lepublicKeyPath`, IDT valide la clé publique de l'appareil lorsqu'il établit une connexion SSH avec le périphérique testé. Si cette valeur n'est pas spécifiée, IDT crée une connexion SSH, mais ne valide pas la clé publique de l'appareil.

Nous vous recommandons vivement de spécifier le chemin d'accès à la clé publique et d'utiliser une méthode sécurisée pour récupérer cette clé publique. Pour les clients SSH standard basés sur une ligne de commande, la clé publique est fournie dans le `known_hosts` fichier. Si vous spécifiez un fichier de clé publique distinct, ce fichier doit utiliser le même format que le `known_hosts` fichier, c'est-à-dire *ip-address key-type public-key*. S'il existe plusieurs entrées avec la même adresse IP, l'entrée correspondant au type de clé utilisé par IDT doit se trouver avant les autres entrées du fichier.

`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.

Configurer userdata.json

IDT for AWS IoT Greengrass V2 a également besoin d'informations supplémentaires sur l'emplacement des artefacts de test et des AWS IoT Greengrass logiciels.

Vous pouvez fournir ces informations à l'aide du modèle `userdata.json` situé dans `<device_tester_extract_location>/configs/userdata.json` :

```
{
  "TempResourcesDirOnDevice": "/path/to/temp/folder",
  "InstallationDirRootOnDevice": "/path/to/installation/folder",
  "GreengrassNucleusZip": "/path/to/aws.greengrass.nucleus.zip",
  "PreInstalled": "yes/no",
  "GreengrassV2TokenExchangeRole": "custom-iam-role-name",
  "hsm": {
    "greengrassPkcsPluginJar": "/path/to/aws.greengrass.crypto.Pkcs11Provider-
latest.jar",
    "pkcs11ProviderLibrary": "/path/to/pkcs11-vendor-library",
    "slotId": "slot-id",
    "slotLabel": "slot-label",
    "slotUserPin": "slot-pin",
    "keyLabel": "key-label",
    "preloadedCertificateArn": "certificate-arn"
    "rootCA": "path/to/root-ca"
  }
}
```

Toutes les propriétés contenant des valeurs sont obligatoires, comme décrit ici :

TempResourcesDirOnDevice

Le chemin complet vers un dossier temporaire sur le périphérique testé dans lequel stocker les artefacts de test. Assurez-vous que les autorisations sudo ne sont pas requises pour écrire dans ce répertoire.

Note

IDT supprime le contenu de ce dossier une fois le test terminé.

InstallationDirRootOnDevice

Le chemin complet vers un dossier de l'appareil dans lequel effectuer l'installation AWS IoT Greengrass. Pour PreInstalled Greengrass, il s'agit du chemin d'accès au répertoire d'installation de Greengrass.

Vous devez définir les autorisations de fichier requises pour ce dossier. Exécutez la commande suivante pour chaque dossier du chemin d'installation.

```
sudo chmod 755 folder-name
```

GreengrassNucleusZip

Le chemin complet vers le fichier ZIP (`greengrass-nucleus-latest.zip`) du noyau de Greengrass sur votre ordinateur hôte. Ce champ n'est pas obligatoire pour les tests avec PreInstalled Greengrass.

Note

Pour plus d'informations sur les versions prises en charge du noyau Greengrass pour IDT, consultez [AWS IoT Greengrass Dernière version IDT pour V2 AWS IoT Greengrass](#). Pour télécharger la dernière version du logiciel Greengrass, voir [Télécharger le AWS IoT Greengrass](#) logiciel.

PreInstalled

Cette fonctionnalité est disponible pour IDT v4.5.8 et versions ultérieures uniquement sur les appareils Linux.

(Facultatif) Lorsque la valeur est **yes**, IDT supposera que le chemin indiqué est le répertoire dans `InstallationDirRootOnDevice` lequel Greengrass est installé.

Pour plus d'informations sur l'installation de Greengrass sur votre appareil, consultez.

[Installation AWS IoT Greengrass du logiciel Core avec provisionnement automatique des ressources](#) En cas d'[installation avec provisionnement manuel](#), incluez l'étape « Ajouter l'

AWS IoT objet à un nouveau groupe d'objets ou à un groupe d'objets existant » lors de la création manuelle d'un [AWS IoT objet](#). IDT part du principe que l'objet et le groupe d'objets sont créés lors de la configuration de l'installation. Assurez-vous que ces valeurs sont reflétées dans le `effectiveConfig.yaml` fichier. IDT vérifie la présence du fichier ci-dessous `effectiveConfig.yaml`. `<InstallationDirRootOnDevice>/config/effectiveConfig.yaml`

Pour exécuter des tests avec HSM, assurez-vous que le `aws.greengrass.crypto.Pkcs11Provider` champ est mis à jour dans `effectiveConfig.yaml`.

`GreengrassV2TokenExchangeRole`

(Facultatif) Le rôle IAM personnalisé que vous souhaitez utiliser comme rôle d'échange de jetons que le périphérique testé suppose d'interagir avec les AWS ressources.

 Note

IDT utilise ce rôle IAM personnalisé au lieu de créer le rôle d'échange de jetons par défaut lors du test. Si vous utilisez un rôle personnalisé, vous pouvez mettre à jour les [autorisations IAM pour l'utilisateur de test afin d'exclure](#) l'`iamResourcesUpdate` instruction qui permet à l'utilisateur de créer et de supprimer des rôles et des politiques IAM.

Pour plus d'informations sur la création d'un rôle IAM personnalisé en tant que rôle d'échange de jetons, consultez [Configurer un rôle d'échange de jetons personnalisé](#).

`hsm`

Cette fonctionnalité est disponible pour IDT v4.5.1 et versions ultérieures.

(Facultatif) Les informations de configuration pour les tests avec un module de sécurité AWS IoT Greengrass matériel (HSM). Sinon, la propriété `hsm` doit être omise. Pour de plus amples informations, veuillez consulter [Intégration de sécurité matérielle](#).

Cette propriété s'applique uniquement si `connectivity.protocol` est défini sur `ssh`.

 Warning

La configuration HSM peut être considérée comme une donnée sensible si le module de sécurité matériel est partagé entre IDT et un autre système. Dans ce cas, vous pouvez éviter de sécuriser ces valeurs de configuration en texte brut en les stockant dans un AWS paramètre Parameter Store SecureString et en configurant IDT pour qu'il les récupère pendant l'exécution du test. Pour plus d'informations, consultez [???](#).

`hsm.greengrassPkcsPluginJar`

Le chemin complet vers le [composant fournisseur PKCS #11](#) que vous téléchargez sur la machine hôte IDT. AWS IoT Greengrass fournit ce composant sous forme de fichier JAR que vous pouvez télécharger pour le spécifier en tant que plugin de provisionnement lors de l'installation. Vous pouvez télécharger la dernière version du fichier JAR du composant à l'adresse suivante : <https://d2s8p88vqu9w66.cloudfront.net/releases/Pkcs11Provider/aws.greengrass.crypto.pkcs11Provider-Latest.jar>.

`hsm.pkcs11ProviderLibrary`

Le chemin complet vers la bibliothèque PKCS #11 fournie par le fournisseur du module de sécurité matérielle (HSM) pour interagir avec le HSM.

`hsm.slotId`

ID de slot utilisé pour identifier le slot HSM dans lequel vous chargez la clé et le certificat.

`hsm.slotLabel`

L'étiquette d'emplacement utilisée pour identifier l'emplacement HSM dans lequel vous chargez la clé et le certificat.

`hsm.slotUserPin`

Le code PIN utilisateur utilisé par IDT pour authentifier le logiciel AWS IoT Greengrass Core auprès du HSM.

 Note

Pour des raisons de sécurité, n'utilisez pas le même code PIN utilisateur sur les appareils de production.

`hsm.keyLabel`

Étiquette utilisée pour identifier la clé dans le module matériel. La clé et le certificat doivent utiliser la même étiquette de clé.

`hsm.preloadedCertificateArn`

Le nom de ressource Amazon (ARN) du certificat d'appareil téléchargé dans le AWS IoT cloud.

Vous devez avoir préalablement généré ce certificat à l'aide de la clé du HSM, l'avoir importé dans votre HSM et l'avoir chargé dans le AWS IoT cloud. Pour plus d'informations sur la génération et l'importation du certificat, consultez la documentation de votre HSM.

Vous devez télécharger le certificat sur le même compte et dans la même région que ceux que vous avez fournis dans [config.json](#). Pour plus d'informations sur le téléchargement de votre certificat vers AWS IoT, voir [Enregistrer un certificat client manuellement](#) dans le Guide du AWS IoT développeur.

`hsm.rootCAPath`

(Facultatif) Le chemin complet sur la machine hôte IDT vers l'autorité de certification racine (CA) qui a signé votre certificat. Cela est nécessaire si le certificat de votre HSM créé n'est pas signé par l'autorité de certification racine Amazon.

Extraire la configuration depuis le magasin de AWS paramètres

AWS IoT Device Tester (IDT) inclut une fonctionnalité optionnelle permettant de récupérer les valeurs de configuration depuis le magasin de [paramètres de AWS Systems Manager](#). AWS Parameter Store permet le stockage sécurisé et crypté des configurations. Une fois configuré, IDT peut récupérer les paramètres depuis le AWS Parameter Store au lieu de les stocker en texte brut dans le fichier `userdata.json`. Cela est utile pour toutes les données sensibles qui doivent être stockées cryptées, telles que les mots de passe, les codes PIN et autres secrets.

1. Pour utiliser cette fonctionnalité, vous devez mettre à jour les autorisations utilisées lors de la création de votre [utilisateur IDT afin d'autoriser l' GetParameter action](#) sur les paramètres pour lesquels IDT est configuré pour utiliser. Vous trouverez ci-dessous un exemple de déclaration d'autorisation qui peut être ajoutée à l'utilisateur IDT. Pour plus d'informations, consultez le [AWS Systems Manager guide de l'utilisateur](#).

```
{
  "Sid": "parameterStoreResources",
  "Effect": "Allow",
  "Action": [
    "ssm:GetParameter"
  ],
  "Resource": "arn:aws:ssm:*:*:parameter/IDT*"
}
```

L'autorisation ci-dessus est configurée pour permettre de récupérer tous les paramètres dont le nom commence par IDT, en utilisant le caractère générique. * Vous devez le personnaliser en fonction de vos besoins afin qu'IDT ait accès à tous les paramètres configurés en fonction du nom des paramètres que vous utilisez.

2. Vous devez stocker vos valeurs de configuration dans AWS Parameter Store. Cela peut être fait à partir de la AWS console ou de la AWS CLI. AWS Parameter Store vous permet de choisir un stockage crypté ou non chiffré. Pour le stockage de valeurs sensibles telles que les secrets, les mots de passe et les codes PIN, vous devez utiliser l'option cryptée qui est un type de paramètre de SecureString. Pour télécharger un paramètre à l'aide de la AWS CLI, vous pouvez utiliser la commande suivante :

```
aws ssm put-parameter --name IDT-example-name --value IDT-example-value --type
SecureString
```

Vous pouvez vérifier qu'un paramètre est enregistré à l'aide de la commande suivante.

(Facultatif) Utilisez l'option `--with-decryption` pour récupérer un paramètre SecureString déchiffré.

```
aws ssm get-parameter --name IDT-example-name
```

L'utilisation de la AWS CLI téléchargera le paramètre dans la AWS région de l'utilisateur CLI actuel et IDT récupérera les paramètres de la région configurée dans `config.json`. Pour vérifier votre région à partir de la AWS CLI, utilisez ce qui suit :

```
aws configure get region
```

- Une fois que vous avez une valeur de configuration dans le AWS cloud, vous pouvez mettre à jour n'importe quelle valeur dans la configuration IDT pour la récupérer depuis le AWS cloud. Pour ce faire, vous utilisez un espace réservé dans votre configuration IDT du formulaire `{{AWS.Parameter.parameter_name}}` pour récupérer le paramètre portant ce nom dans le Parameter Store. AWS

Supposons, par exemple, que vous souhaitiez utiliser le IDT-example-name paramètre de l'étape 2 en tant que KeyLabel HSM dans votre configuration HSM. Pour ce faire, vous pouvez mettre à jour votre `userdata.json` compte comme suit :

```
"hsm": {  
    "keyLabel": "{{AWS.Parameter.IDT-example-name}}",  
    [...]  
}
```

IDT récupérera la valeur de ce paramètre au moment de l'exécution qui a été définie IDT-example-value à l'étape 2. Cette configuration est similaire au paramètre, "keyLabel": "IDT-example-value" mais cette valeur est stockée sous forme cryptée dans le AWS Cloud.

Exécutez la suite AWS IoT Greengrass de qualifications

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.

Utilisez la `run-suite` commande suivante pour exécuter une série de tests.

```
devicetester_[linux | mac | win]_x86-64 run-suite \  
  --suite-id suite-id \  
  --group-id group-id \  
  --pool-id your-device-pool \  
  --test-id test-id \  
  --update-idt y/n \  
  --userdata userdata.json
```

Toutes les options sont facultatives. Par exemple, vous pouvez omettre `pool-id` si un seul pool de périphériques, qui est un ensemble d'appareils identiques, est défini dans votre `device.json` fichier. 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 de plus amples informations, veuillez consulter [the section called “Versions de la suite de tests”](#).

Exemples de commandes pour exécuter la suite de qualifications

Les exemples de ligne de commande suivants vous montrent comment exécuter les tests de qualification pour un pool de périphériques. Pour plus d'informations sur `run-suite` et d'autres commandes IDT, veuillez consulter [the section called “Commandes IDT”](#).

Utilisez la commande suivante pour exécuter tous les groupes de tests dans une suite de tests spécifiée. La `list-suites` commande répertorie les suites de tests qui se trouvent dans le `tests` dossier.

```
devicetester_[linux | mac | win]_x86-64 run-suite \  
  --suite-id GGV2Q_1.0.0 \  
  --pool-id <pool-id> \  
  --userdata userdata.json
```

Utilisez la commande suivante pour exécuter un groupe de tests spécifique dans une suite de tests. La `list-groups` commande répertorie les groupes de tests d'une suite de tests.

```
devicetester_[linux | mac | win]_x86-64 run-suite \  
  --suite-id GGV2Q_1.0.0 \  
  --group-id <group-id> \  
  --pool-id <pool-id> \  
  --userdata userdata.json
```

Utilisez la commande suivante pour exécuter un scénario de test spécifique dans un groupe de test.

```
devicetester_[linux | mac | win]_x86-64 run-suite \  
  --group-id <group-id> \  
  --test-id <test-id>
```



```
--test-id <test-id> \  
--userdata userdata.json
```

Utilisez la commande suivante pour exécuter plusieurs scénarios de test dans un groupe de test.

```
devicetester_[linux | mac | win]_x86-64 run-suite \  
--group-id <group-id> \  
--test-id <test-id1>,<test-id2>  
--userdata userdata.json
```

Utilisez la commande suivante pour répertorier tous les cas de test d'un groupe de test.

```
devicetester_[linux | mac | win]_x86-64 list-test-cases --group-id <group-id>
```

Nous vous recommandons d'exécuter la suite complète de tests de qualification, qui exécute les dépendances des groupes de test dans le bon ordre. Si vous choisissez d'exécuter des groupes de test spécifiques, nous vous recommandons d'exécuter d'abord le groupe de test du vérificateur de dépendances pour vous assurer que toutes les dépendances de Greengrass sont installées avant d'exécuter les groupes de test associés. Par exemple :

- Exécutez `coredependencies` avant les groupes de tests de qualification du noyau.

IDT pour les commandes AWS IoT Greengrass V2

Les commandes IDT se trouvent dans le répertoire `<device-tester-extract-location>/bin`. Pour exécuter une suite de tests, vous devez fournir la commande au format suivant :

`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 AWS IoT Greengrass les versions, 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`. Le groupe de test 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 test à exécuter, sous forme de liste séparée par des virgules. Si ce n'est pas spécifié, IDT exécute tous les groupes de tests appropriés dans la suite de tests en fonction des paramètres configurés dans `device.json`. IDT n'exécute aucun groupe de test que l'appareil ne prend en charge en fonction de vos paramètres configurés, même si ces groupes de test sont spécifiés dans la `group-id` liste.
- `test-id`. Les cas de test à exécuter, sous forme de liste séparée par des virgules. Lorsqu'il est spécifié, `group-id` doit spécifier un seul groupe.
- `pool-id`. Le pool d'appareils à tester. Vous devez spécifier un groupe si plusieurs groupes de périphériques sont définis dans votre fichier `device.json`.
- `stop-on-first-failure`. Configure IDT pour qu'il cesse de fonctionner lors du premier échec. Utilisez cette option `group-id` lorsque vous souhaitez déboguer les groupes de test 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 à l'invite de mise à jour de l'IDT. La Y réponse arrête l'exécution du test si IDT détecte qu'il existe une version plus récente. La N réponse poursuit l'exécution du test.
- `userdata`. Le chemin complet du `userdata.json` fichier contenant des informations sur les chemins des artefacts de test. Cette option est obligatoire pour la `run-suite` commande. Le `userdata.json` fichier doit se trouver dans le répertoire `devicetester_extract_location/devicetester_ggv2_ [win|mac|linux] /configs/`.

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.

Pour résoudre les erreurs, voir [Résolution des problèmes liés à IDT pour V2 AWS IoT Greengrass](#).

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 dans `<device-tester-extract-location>/results/<execution-id>/`. Les deux rapports capturent les résultats de l'exécution de la suite de tests de qualification.

`awsiotdevicetester_report.xml` s'agit du rapport de test de qualification que vous soumettez AWS pour répertorier votre appareil dans le catalogue des AWS Partner appareils. Ce rapport contient les éléments suivants :

- La version IDT.
- AWS IoT Greengrass Version 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.
- Répartition des résultats des tests par bibliothèques testées en fonction des fonctionnalités de l'appareil, telles que l'accès aux ressources locales, le shadow et le MQTT.

Le `GGV2Q_Result.xml` rapport 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.
- Répartition des résultats des tests en AWS IoT Greengrass fonction de la fonctionnalité testée.

Interprétation AWS IoT Device Tester des résultats

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 du test. 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`

Le temps, en secondes, nécessaire à l'exécution de la suite de qualification.

`tests`

Le nombre de tests exécutés.

`failures`

Nombre de tests exécutés mais dont le résultat n'est pas probant.

`errors`

Le nombre de tests qu'IDT n'a pas pu exécuter.

`disabled`

Ignorez cet attribut. Elle n'est pas utilisée.

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`.

```
<name="aws-iot-greengrass-v2-core" value="supported" type="required"></feature>
```

En l'absence d'échec des tests ou d'erreurs pour les fonctionnalités requises, votre appareil répond aux exigences techniques requises pour fonctionner AWS IoT Greengrass et peut interagir avec les AWS IoT services. Si vous souhaitez répertorier votre appareil dans le catalogue des AWS Partner appareils, 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 `<testsuite>` XML contient des `<testcase>` balises pour chaque test effectué pour un groupe de test. 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

Le nombre de fois où IDT a exécuté le scénario de 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 débogage. 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 des tests effectués. `<devicetester-extract-location>/results/<execution-id>/logs` Deux ensembles de journaux sont générés :

`test_manager.log`

Journaux générés à partir du composant Test Manager de AWS IoT Device Tester (par exemple, journaux relatifs à la configuration, au séquençage des tests et à la génération de rapports).

`<test-case-id>.log` (for example, `lambdaDeploymentTest.log`)

Journaux du scénario de test au sein du groupe de test, y compris les journaux du périphérique testé. À partir de la version 4.2.0 d'IDT, IDT regroupe les journaux de test pour chaque cas de test dans un `<test-case-id>` dossier distinct au sein du répertoire. `<devicetester-extract-location>/results/<execution-id>/logs/<test-group-id>/`

Utilisez IDT pour développer et exécuter vos propres suites de tests

À partir de IDT v4.0.1, IDT pour AWS IoT Greengrass V2 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.

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 testeurs. 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.
- Consultez les résultats des tests et les journaux d'exécution des tests exécutés par IDT.

Téléchargez la dernière version de AWS IoT Device Tester for AWS IoT Greengrass

Téléchargez la [dernière version](#) d'IDT et extrayez le logiciel dans un emplacement (*<device-tester-extract-location>*) de votre système de fichiers où vous disposez d'autorisations de lecture/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.

Flux de travail de création de suites de tests

Les suites de tests sont composées de trois types de fichiers :

- Fichiers de configuration fournissant à IDT des informations sur la façon d'exécuter la suite de tests.
- Testez les fichiers exécutables utilisés par IDT pour exécuter des scénarios de test.
- Des fichiers supplémentaires sont nécessaires pour exécuter les tests.

Suivez les étapes de base suivantes pour créer des tests IDT personnalisés :

1. [Créez des fichiers de configuration](#) pour votre suite de tests.
2. [Créez des exécutables de cas de test](#) contenant la logique de test de votre suite de tests.
3. Vérifiez et documentez les [informations de configuration requises pour que les testeurs](#) exécutent la suite de tests.
4. Vérifiez qu'IDT peut exécuter votre suite de tests et produire les [résultats des tests](#) comme prévu.

Pour créer rapidement un exemple de suite personnalisée et l'exécuter, suivez les instructions figurant dans [Tutoriel : création et exécution de l'exemple de suite de tests IDT](#).

Pour commencer à créer une suite de tests personnalisée en Python, consultez [Tutoriel : Développement d'une suite de tests IDT simple](#).

Tutoriel : création et exécution de l'exemple de suite de tests IDT

Le AWS IoT Device Tester téléchargement 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 vous pouvez utiliser IDT AWS IoT Greengrass pour exécuter des suites de tests personnalisées.

Dans ce didacticiel, vous allez effectuer les étapes suivantes :

1. [Créez la suite d'exemples de tests](#)
2. [Utiliser IDT pour exécuter l'exemple de suite de tests](#)

Prérequis

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

- Exigences relatives à l'ordinateur hôte
 - Dernière version de AWS IoT Device Tester
 - [Python](#) 3.7 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-la à la python --version place. Si le numéro de version renvoyé est 3.7 ou supérieur, exécutez la commande suivante dans un terminal Powershell pour la définir python3 comme alias pour votre 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 de la section [Télécharger Python](#) pour installer Python 3.7+. Pour plus d'informations, consultez la [documentation Python](#).

- [urllib3](#)

Pour vérifier qu'urllib3 est correctement installé, exécutez la commande suivante :

```
python3 -c 'import urllib3'
```

S'il n'urllib3 est pas installé, exécutez la commande suivante pour l'installer :

```
python3 -m pip install urllib3
```

- Exigences relatives aux dispositifs
- 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 un [Raspberry Pi](#) avec le système d'exploitation Raspberry Pi. Assurez-vous de configurer [SSH](#) sur votre Raspberry Pi pour vous y connecter à distance.

Configurer les informations de l'appareil pour IDT

Configurez les informations de votre appareil pour qu'IDT exécute le test. Vous devez mettre à jour le device.json modèle situé dans le *<device-tester-extract-location>*/configs dossier avec 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 l'`devices` objet, fournissez les informations suivantes :

`id`

Identifiant unique défini par l'utilisateur pour votre appareil.

`connectivity.ip`

L'adresse IP de votre appareil.

`connectivity.port`

Facultatif. Le 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 vous connecter à votre appareil.

`connectivity.auth.credentials.privKeyPath`

Le chemin complet vers la clé privée utilisée pour vous connecter à votre appareil.

Cette valeur s'applique uniquement si `connectivity.auth.method` est défini sur `pki`.

`devices.connectivity.auth.credentials.password`

Le mot de passe utilisé pour vous 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`.

Créez la suite d'exemples de tests

Le `<device-tester-extract-location>/samples/python` dossier contient des exemples de fichiers de configuration, du code source et le SDK du 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 dossier situé à l'intérieur du *<device-tester-extract-location>/tests* dossier. Passez en revue les fichiers du IDTSampleSuitePython_1.0.0 dossier pour comprendre comment l'exemple de suite de tests est structuré et pour voir divers exemples d'exécutables de scénarios de test et de fichiers JSON de configuration de test.

Note

L'exemple de suite de tests inclut le code source python. N'incluez pas d'informations sensibles dans le code de votre suite de tests.

Étape suivante : utilisez IDT pour [exécuter l'exemple de suite de tests](#) que vous avez créée.

Utiliser IDT pour exécuter l'exemple de suite de tests

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 d'exemples de tests et diffuse les résultats sur 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
```

Résolution des problèmes

Utilisez les informations suivantes pour résoudre les problèmes rencontrés lors de l'exécution du didacticiel.

Le scénario de test ne s'exécute pas correctement

Si le test échoue, IDT diffuse les journaux d'erreurs sur la console afin de vous aider à résoudre les problèmes liés à l'exécution du test. Assurez-vous de remplir toutes les [conditions requises](#) pour ce didacticiel.

Impossible de se connecter à l'appareil testé

Vérifiez les paramètres suivants :

- Votre `device.json` fichier contient l'adresse IP, le port et les informations d'authentification corrects.
- Vous pouvez vous connecter à votre appareil via SSH depuis votre ordinateur hôte.

Tutoriel : Développement d'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 décrivant la suite de tests

Ce didacticiel vous montre comment utiliser IDT AWS IoT Greengrass pour développer une suite de tests Python contenant un seul cas de test. Dans ce didacticiel, vous allez effectuer les étapes suivantes :

1. [Création d'un répertoire de suites de tests](#)
2. [Création de fichiers de configuration](#)
3. [Création de l'exécutable du scénario de test](#)
4. [Exécutez la suite de tests](#)

Prérequis

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

- Exigences relatives à l'ordinateur hôte
 - Dernière version de AWS IoT Device Tester
 - [Python](#) 3.7 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-la à la `python --version` place. Si le numéro de version renvoyé est 3.7 ou supérieur, exécutez la commande suivante dans un terminal Powershell pour la définir `python3` comme alias pour votre `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 de la section [Télécharger Python](#) pour installer Python 3.7+. Pour plus d'informations, consultez la [documentation Python](#).

- [urllib3](#)

Pour vérifier qu'`urllib3` est correctement installé, exécutez la commande suivante :

```
python3 -c 'import urllib3'
```

S'il n'y a pas d'installation, exécutez la commande suivante pour l'installer :

```
python3 -m pip install urllib3
```

- Exigences relatives aux dispositifs
- 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 un [Raspberry Pi](#) avec le système d'exploitation Raspberry Pi. Assurez-vous de configurer [SSH](#) sur votre Raspberry Pi pour vous y connecter à distance.

Création d'un répertoire de suites de tests

IDT sépare logiquement les cas de test en groupes de tests au sein de chaque suite de tests. Chaque cas de test doit faire partie d'un groupe de test. 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
```

Création de fichiers de configuration

Votre suite de tests doit contenir les [fichiers de configuration](#) requis suivants :

Fichiers de configuration requis

suite.json

Contient des informations sur la suite de tests. Consultez [Configurer suite.json](#).

group.json

Contient des informations sur un groupe de test. Vous devez créer un `group.json` fichier pour chaque groupe de test de votre suite de tests. Consultez [Configurer group.json](#).

test.json

Contient des informations sur un scénario de test. Vous devez créer un `test.json` fichier pour chaque scénario de test de votre suite de tests. Consultez [Configurer test.json](#).

1. Dans le `MyTestSuite_1.0.0/suite` dossier, créez un `suite.json` fichier avec la structure suivante :

```
{
  "id": "MyTestSuite_1.0.0",
  "title": "My Test Suite",
  "details": "This is my test suite.",
  "userDataRequired": false
}
```

2. Dans le `MyTestSuite_1.0.0/myTestGroup` dossier, créez un `group.json` fichier avec la structure suivante :

```
{
  "id": "MyTestGroup",
  "title": "My Test Group",
  "details": "This is my test group.",
  "optional": false
}
```

3. Dans le `MyTestSuite_1.0.0/myTestGroup/myTestCase` dossier, créez un `test.json` fichier 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 de votre MyTestSuite_1.0.0 dossier doit désormais ressembler à ce qui suit :

```
MyTestSuite_1.0.0
### suite
### suite.json
### myTestGroup
### group.json
### myTestCase
### test.json
```

Obtenez le SDK du client IDT

Vous utilisez le [SDK du client IDT](#) pour permettre à IDT d'interagir avec l'appareil testé et de communiquer les résultats des tests. Pour ce didacticiel, vous allez utiliser la version Python du SDK.

Depuis le *<device-tester-extract-location>*/sdks/python/ dossier, idt_client copiez-le dans votre MyTestSuite_1.0.0/suite/myTestGroup/myTestCase dossier.

Pour vérifier que le SDK a bien été copié, exécutez la commande suivante.

```
cd MyTestSuite_1.0.0/suite/myTestGroup/myTestCase
python3 -c 'import idt_client'
```

Création de l'exécutable du scénario de test

Les exécutables du scénario de test contiennent la logique de test que vous souhaitez exécuter. Une suite de tests peut contenir plusieurs exécutables de scénarios de test. Pour ce didacticiel, vous ne créez qu'un seul exécutable de scénario de test.

1. Créez le fichier de suite de tests.

Dans le `MyTestSuite_1.0.0/suite/myTestGroup/myTestCase` dossier, créez un `myTestCase.py` fichier avec 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 du SDK client pour ajouter la logique de test suivante à votre `myTestCase.py` fichier :

a. Exécutez une commande SSH sur le périphérique 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()
```

Configurer les informations de l'appareil pour IDT

Configurez les informations de votre appareil pour qu'IDT exécute le test. Vous devez mettre à jour le `device.json` modèle situé dans le *<device-tester-extract-location>*/configs dossier avec 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 l'`devices` objet, fournissez les informations suivantes :

`id`

Identifiant unique défini par l'utilisateur pour votre appareil.

`connectivity.ip`

L'adresse IP de votre appareil.

`connectivity.port`

Facultatif. Le 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 vous connecter à votre appareil.

`connectivity.auth.credentials.privKeyPath`

Le chemin complet vers la clé privée utilisée pour vous connecter à votre appareil.

Cette valeur s'applique uniquement si `connectivity.auth.method` est défini sur `pki`.

`devices.connectivity.auth.credentials.password`

Le mot de passe utilisé pour vous 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

Après avoir créé votre suite de tests, vous devez vous assurer qu'elle fonctionne comme prévu. Pour ce faire, procédez comme suit pour exécuter la suite de tests avec votre pool d'appareils existant.

1. Copiez votre `MyTestSuite_1.0.0` dossier dans `<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 sur 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
```

Résolution des problèmes

Utilisez les informations suivantes pour résoudre tout problème lié à l'exécution du didacticiel.

Le scénario de test ne s'exécute pas correctement

Si le test échoue, IDT diffuse les journaux d'erreurs sur la console afin de vous aider à résoudre les problèmes liés à l'exécution du test. Avant de consulter les journaux d'erreurs, vérifiez les points suivants :

- Le SDK du client IDT se trouve dans le bon dossier, comme décrit dans [cette](#) étape.
- Vous répondez à tous les [prérequis](#) pour ce didacticiel.

Impossible de se connecter à l'appareil testé

Vérifiez les paramètres suivants :

- Votre `device.json` fichier contient l'adresse IP, le port et les informations d'authentification corrects.
- Vous pouvez vous connecter à votre appareil via SSH depuis votre ordinateur hôte.

Création de fichiers de configuration de la suite de tests IDT

Cette section décrit les formats dans lesquels vous créez des fichiers de configuration que vous incluez lorsque vous écrivez une suite de tests personnalisée.

Fichiers de configuration requis

`suite.json`

Contient des informations sur la suite de tests. Consultez [Configurer suite.json](#).

`group.json`

Contient des informations sur un groupe de test. Vous devez créer un `group.json` fichier pour chaque groupe de test de votre suite de tests. Consultez [Configurer group.json](#).

`test.json`

Contient des informations sur un scénario de test. Vous devez créer un `test.json` fichier pour chaque scénario de test de votre suite de tests. Consultez [Configurer test.json](#).

Fichiers de configuration facultatifs

`test_orchestrator.yaml` ou `state_machine.json`

Définit la manière dont les tests sont exécutés lorsque IDT exécute la suite de tests. SSe [Configurer test_orchestrator.yaml](#).

Note

À partir de IDT v4.5.1, vous utilisez le `test_orchestrator.yaml` fichier pour définir le flux de travail de test. Dans les versions précédentes d'IDT, vous utilisiez

le `state_machine.json` fichier. Pour plus d'informations sur la machine à états, consultez [Configuration de la machine d'état IDT](#).

`userdata_schema.json`

Définit le schéma du [userdata.json](#) fichier que les testeurs peuvent inclure dans leur configuration de configuration. Le `userdata.json` fichier est utilisé pour toute information de configuration supplémentaire requise pour exécuter le test mais absente du `device.json` fichier. Consultez [Configurer userdata_schema.json](#).

Les fichiers de configuration sont placés dans votre dossier *<custom-test-suite-folder>*, comme indiqué ici.

```
<custom-test-suite-folder>
### suite
  ### suite.json
  ### test_orchestrator.yaml
  ### userdata_schema.json
  ### <test-group-folder>
    ### group.json
    ### <test-case-folder>
      ### test.json
```

Configurer suite.json

Le `suite.json` fichier définit les variables d'environnement et détermine si les données utilisateur sont nécessaires pour exécuter la suite de tests. Utilisez le modèle suivant pour configurer votre *<custom-test-suite-folder>/suite/suite.json* 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

Un identifiant unique défini par l'utilisateur pour la suite de tests. La valeur de `id` doit correspondre au nom du dossier de la suite de tests dans lequel se trouve le `suite.json` fichier. Le nom et la version de la suite doivent également répondre aux exigences suivantes :

- `<suite-name>` ne peut pas contenir de traits de soulignement.
- `<suite-version>` est noté `x.x.x`, où `x` est un nombre.

L'ID est indiqué 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 la CLI IDT pour les testeurs.

details

Brève description de l'objectif de la suite de tests.

userDataRequired

Définit si les testeurs doivent inclure des informations personnalisées dans un `userdata.json` fichier. Si vous définissez cette valeur sur `true`, vous devez également inclure le [userdata_schema.json](#) fichier dans le dossier de votre suite de tests.

environmentVariables

Facultatif. Un tableau de variables d'environnement à définir pour cette suite de tests.

environmentVariables.key

Le nom de la variable d'environnement.

environmentVariables.value

Valeur de la variable d'environnement.

Configurer group.json

Le `group.json` fichier définit si un groupe de test est obligatoire ou facultatif. Utilisez le modèle suivant pour configurer votre `<custom-test-suite-folder>/suite/<test-group>/group.json` 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

Un identifiant unique défini par l'utilisateur pour le groupe de test. La valeur de `id` doit correspondre au nom du dossier du groupe de test dans lequel se trouve le `group.json` fichier et ne peut pas contenir 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 la CLI IDT pour les testeurs.

details

Brève description de l'objectif du groupe de test.

optional

Facultatif. Définissez sur `true` pour afficher ce groupe de test en tant que groupe facultatif une fois qu'IDT a terminé d'exécuter les tests requis. La valeur par défaut est `false`.

Configurer test.json

Le `test.json` fichier détermine les exécutables du scénario de test et les variables d'environnement utilisées par un scénario de test. Pour plus d'informations sur la création d'exécutables de scénarios de test, consultez [Création d'exécutables de scénarios de test IDT](#)

Utilisez le modèle suivant pour configurer votre `<custom-test-suite-folder>/suite/<test-group>/<test-case>/test.json` 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 scénario de test. La valeur de `id` doit correspondre au nom du dossier de scénario de test dans lequel se trouve le `test.json` fichier et ne peut pas contenir de traits de soulignement (`_`). L'ID est utilisé dans les rapports de test générés par IDT.

`title`

Nom descriptif du scénario de test. Le nom est affiché dans la CLI IDT pour les testeurs.

`details`

Brève description de l'objectif du scénario de test.

`requiredDUT`

Facultatif. Réglez sur `true` si un appareil est requis pour exécuter ce test, sinon sur `false`. La valeur par défaut est `true`. Les testeurs configureront les appareils qu'ils utiliseront pour exécuter le test dans leur `device.json` fichier.

`requiredResources`

Facultatif. Un tableau qui fournit des informations sur les périphériques de ressources nécessaires pour exécuter ce test.

`requiredResources.name`

Le nom unique à attribuer au périphérique de ressource lors de l'exécution de ce test.

`requiredResources.features`

Un ensemble de fonctionnalités de périphérique de ressources définies par l'utilisateur.

`requiredResources.features.name`

Nom de la fonctionnalité. Fonctionnalité de l'appareil pour laquelle vous souhaitez utiliser cet appareil. Ce nom est comparé au nom de la fonctionnalité fourni par le testeur dans le `resource.json` fichier.

`requiredResources.features.version`

Facultatif. Version de la fonctionnalité. Cette valeur est comparée à la version de fonctionnalité fournie par le lanceur de test dans le `resource.json` fichier. Si aucune version n'est fournie, la fonctionnalité n'est pas cochée. Si aucun numéro de version n'est requis pour la fonctionnalité, laissez ce champ vide.

`requiredResources.features.jobSlots`

Facultatif. Le 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 fonctionnalités individuelles, nous vous recommandons de définir cette valeur sur 1.

`execution.timeout`

Durée (en millisecondes) pendant laquelle IDT attend la fin du test. Pour plus d'informations sur la définition de cette valeur, consultez [Création d'exécutables de scénarios de test IDT](#).

`execution.os`

Les exécutables du scénario de test à exécuter sont basés sur le système d'exploitation de l'ordinateur hôte qui exécute IDT. Les valeurs prises en charge sont `linux`, `mac` et `win`.

`execution.os.cmd`

Le chemin d'accès au fichier exécutable du scénario de test que vous souhaitez exécuter pour le système d'exploitation spécifié. Cet emplacement doit se trouver dans le chemin du système.

`execution.os.args`

Facultatif. Les arguments à fournir pour exécuter l'exécutable du scénario de test.

`environmentVariables`

Facultatif. Un tableau de variables d'environnement défini pour ce cas de test.

`environmentVariables.key`

Le 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` fichier et dans le `suite.json` fichier, la valeur du `test.json` fichier est prioritaire.

Configurer `test_orchestrator.yaml`

Un orchestrateur de tests 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 d'orchestrateur de test défini par l'utilisateur, IDT générera un orchestrateur de tests pour vous.

L'orchestrateur de test par défaut exécute les fonctions suivantes :

- Permet aux testeurs de sélectionner et d'exécuter des groupes de tests spécifiques, au lieu de recourir à la suite de tests complète.
- Si aucun groupe de test spécifique n'est sélectionné, exécute chaque groupe de test de la suite de tests dans un ordre aléatoire.
- Génère des rapports et imprime un résumé de console présentant les résultats des tests pour chaque groupe de test et chaque cas de test.

Pour plus d'informations sur le fonctionnement de l'orchestrateur de test IDT, consultez. [Configuration de l'orchestrateur de test IDT](#)

Configurer `userdata_schema.json`

Le `userdata_schema.json` fichier détermine le schéma dans lequel les testeurs fournissent les données utilisateur. Les données utilisateur sont requises si votre suite de tests nécessite des informations qui ne figurent pas dans le `device.json` fichier. Par exemple, vos tests peuvent nécessiter des informations d'identification du 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 appelé `userdata`, dont la valeur est un `userdata.json` fichier, que les utilisateurs créent dans leur `<device-tester-extract-location>/config` dossier. Le format du `userdata.json` fichier est basé sur le `userdata_schema.json` fichier que vous incluez dans la suite de tests.

Pour indiquer que les testeurs doivent fournir un `userdata.json` fichier :

1. Dans le `suite.json` fichier, définissez `userDataRequired` sur `true`.
2. Dans votre `<custom-test-suite-folder>`, créez un `userdata_schema.json` fichier.
3. Modifiez le `userdata_schema.json` fichier pour créer un schéma [JSON IETF Draft v4](#) valide.

Lorsque IDT exécute votre suite de tests, il lit automatiquement le schéma et l'utilise pour valider le `userdata.json` fichier fourni par le testeur. S'il est valide, le contenu du `userdata.json` fichier est disponible à la fois dans le contexte [IDT et dans le contexte](#) de l'[orchestrateur de test](#).

Configuration de l'orchestrateur de test IDT

À partir de IDT v4.5.1, IDT inclut un nouveau composant d'orchestrateur de test. L'orchestrateur de tests est un composant IDT qui contrôle le flux d'exécution de la suite de tests et génère le rapport de test une fois que IDT a terminé d'exécuter tous les tests. L'orchestrateur de tests détermine la sélection des tests et l'ordre dans lequel les tests sont exécutés en fonction des règles définies par l'utilisateur.

Si votre suite de tests n'inclut pas d'orchestrateur de test défini par l'utilisateur, IDT générera un orchestrateur de tests pour vous.

L'orchestrateur de test par défaut exécute les fonctions suivantes :

- Permet aux testeurs de sélectionner et d'exécuter des groupes de tests spécifiques, au lieu de recourir à la suite de tests complète.
- Si aucun groupe de test spécifique n'est sélectionné, exécute chaque groupe de test de la suite de tests dans un ordre aléatoire.
- Génère des rapports et imprime un résumé de console présentant les résultats des tests pour chaque groupe de test et chaque cas de test.

L'orchestrateur de test remplace l'orchestrateur de test IDT. Nous vous recommandons vivement d'utiliser l'orchestrateur de tests pour développer vos suites de tests au lieu de l'orchestrateur de tests IDT. L'orchestrateur de test fournit les fonctionnalités améliorées suivantes :

- Utilise un format déclaratif par rapport au format impératif utilisé par la machine d'état IDT. Cela vous permet de spécifier les tests que vous souhaitez exécuter et le moment où vous souhaitez les exécuter.

- Gère la gestion de groupes spécifiques, la génération de rapports, la gestion des erreurs et le suivi des résultats afin que vous n'ayez pas à gérer manuellement ces actions.
- Utilise le format YAML, qui prend en charge les commentaires par défaut.
- Nécessite 80 % d'espace disque en moins que l'orchestrateur de test pour définir le même flux de travail.
- Ajoute une validation préalable pour vérifier que la définition de votre flux de travail ne contient pas de tests IDs ou de dépendances circulaires incorrects.

Tester le format de l'orchestrateur

Vous pouvez utiliser le modèle suivant pour configurer votre propre *<custom-test-suite-folder>/suite/test_orchestrator.yaml* fichier :

```
Aliases:
  string: context-expression

ConditionalTests:
  - Condition: context-expression
    Tests:
      - test-descriptor

Order:
  - - group-descriptor
    - group-descriptor

Features:
  - Name: feature-name
    Value: support-description
    Condition: context-expression
    Tests:
      - test-descriptor
    OneOfTests:
      - test-descriptor
    IsRequired: boolean
```

Tous les champs qui contiennent des valeurs sont requis, comme indiqué ici :

Aliases

Facultatif. Chaînes définies par l'utilisateur qui correspondent à des expressions de contexte. Les alias vous permettent de générer des noms conviviaux pour identifier les expressions contextuelles dans la configuration de votre orchestrateur de test. Cela est particulièrement utile si vous créez des expressions contextuelles complexes ou des expressions que vous utilisez à plusieurs endroits.

Vous pouvez utiliser des expressions contextuelles pour stocker des requêtes contextuelles qui vous permettent d'accéder aux données d'autres configurations IDT. Pour de plus amples informations, veuillez consulter [Accédez aux données dans le contexte](#).

Exemple exemple

Aliases:

```
FizzChosen: "'{{$pool.features[?(@.name == 'Fizz')].value[0]}}' == 'yes'"
BuzzChosen: "'{{$pool.features[?(@.name == 'Buzz')].value[0]}}' == 'yes'"
FizzBuzzChosen: "'{{$aliases.FizzChosen}}' && '{{$aliases.BuzzChosen}}'"
```

ConditionalTests

Facultatif. Une liste de conditions et les cas de test correspondants qui sont exécutés lorsque chaque condition est satisfaite. Chaque condition peut comporter plusieurs scénarios de test ; toutefois, vous ne pouvez attribuer un scénario de test donné qu'à une seule condition.

Par défaut, IDT exécute tout scénario de test qui n'est pas affecté à une condition de cette liste. Si vous ne spécifiez pas cette section, IDT exécute tous les groupes de tests de la suite de tests.

Chaque élément de la `ConditionalTests` liste inclut les paramètres suivants :

Condition

Expression de contexte qui correspond à une valeur booléenne. Si la valeur évaluée est vraie, IDT exécute les scénarios de test spécifiés dans le `Tests` paramètre.

Tests

La liste des descripteurs de test.

Chaque descripteur de test utilise l'identifiant du groupe de test et un ou plusieurs cas de test IDs pour identifier les tests individuels à exécuter à partir d'un groupe de test spécifique. Le descripteur de test utilise le format suivant :

```
GroupId: group-id  
CaseIds: [test-id, test-id] # optional
```

Exemple exemple

L'exemple suivant utilise des expressions contextuelles génériques que vous pouvez définir comme telles `Aliases`.

```
ConditionalTests:  
  - Condition: "{{$aliases.Condition1}}"  
    Tests:  
      - GroupId: A  
      - GroupId: B  
  - Condition: "{{$aliases.Condition2}}"  
    Tests:  
      - GroupId: D  
  - Condition: "{{$aliases.Condition1}} || {{$aliases.Condition2}}"  
    Tests:  
      - GroupId: C
```

Sur la base des conditions définies, IDT sélectionne les groupes de test comme suit :

- Si `Condition1` c'est vrai, IDT exécute les tests dans les groupes de tests A, B et C.
- Si `Condition2` c'est vrai, IDT exécute les tests dans les groupes de test C et D.

Order

Facultatif. Ordre dans lequel les tests doivent être exécutés. Vous spécifiez l'ordre des tests au niveau du groupe de test. Si vous ne spécifiez pas cette section, IDT exécute tous les groupes de test applicables dans un ordre aléatoire. La valeur de `Order` est une liste de listes de descripteurs de groupes. Tout groupe de test non répertorié peut être exécuté en `Order` parallèle avec n'importe quel autre groupe de test répertorié.

Chaque liste de descripteurs de groupe contient un ou plusieurs descripteurs de groupe et identifie l'ordre dans lequel exécuter les groupes spécifiés dans chaque descripteur. Vous pouvez utiliser les formats suivants pour définir des descripteurs de groupes individuels :

- *group-id*: l'ID de groupe d'un groupe de test existant.
- [*group-id*, *group-id*]—Liste des groupes de tests qui peuvent être exécutés dans n'importe quel ordre les uns par rapport aux autres.

- "*" —Wildcard. Cela équivaut à la liste de tous les groupes de test qui ne sont pas déjà spécifiés dans la liste actuelle des descripteurs de groupes.

La valeur pour `Order` doit également répondre aux exigences suivantes :

- Le groupe de test IDs que vous spécifiez dans un descripteur de groupe doit exister dans votre suite de tests.
- Chaque liste de descripteurs de groupes doit inclure au moins un groupe de test.
- Chaque liste de descripteurs de groupes doit contenir un groupe IDs unique. Vous ne pouvez pas répéter un identifiant de groupe de test dans des descripteurs de groupe individuels.
- Une liste de descripteurs de groupes peut comporter au maximum un descripteur de groupe générique. Le descripteur de groupe générique doit être le premier ou le dernier élément de la liste.

Exemple Exemples

Pour une suite de tests contenant les groupes de tests A, B, C, D et E, la liste d'exemples suivante montre différentes manières de spécifier qu'IDT doit d'abord exécuter le groupe de test A, puis exécuter le groupe de test B, puis exécuter les groupes de test C, D et E dans n'importe quel ordre.

- `Order:`
 - A
 - B
 - [C, D, E]

- `Order:`
 - A
 - B
 - "*"

- `Order:`
 - A
 - B
 - B
 - C
 - B
 - D

- - B
- E

Features

Facultatif. Liste des fonctionnalités du produit que vous souhaitez qu'IDT ajoute au `awsiotdevicetester_report.xml` fichier. Si vous ne spécifiez pas cette section, IDT n'ajoutera aucune fonctionnalité du produit au rapport.

Les fonctionnalités d'un produit sont des informations définies par l'utilisateur concernant des critères spécifiques auxquels un appareil peut répondre. Par exemple, la fonctionnalité du produit MQTT peut indiquer que l'appareil publie correctement les messages MQTT. Dans `awsiotdevicetester_report.xml`, les fonctionnalités du produit sont définies sous la forme de `supported`/`not-supported`, ou d'une valeur personnalisée définie par l'utilisateur, en fonction de la réussite des tests spécifiés.

Chaque élément de la Features liste comprend les paramètres suivants :

Name

Le nom de la fonctionnalité.

Value

Facultatif. La valeur personnalisée que vous souhaitez utiliser dans le rapport à la place de `supported`. Si cette valeur n'est pas spécifiée, l'IDT basé sur le paramètre définit la valeur de la fonction sur `supported` ou en `not-supported` fonction des résultats des tests. Si vous testez la même fonctionnalité dans des conditions différentes, vous pouvez utiliser une valeur personnalisée pour chaque instance de cette fonctionnalité dans la Features liste, et IDT concatène les valeurs des entités pour les conditions prises en charge. Pour plus d'informations, veuillez consulter la rubrique

Condition

Expression de contexte qui correspond à une valeur booléenne. Si la valeur évaluée est vraie, IDT ajoute la fonctionnalité au rapport de test une fois l'exécution de la suite de tests terminée. Si la valeur évaluée est fausse, le test n'est pas inclus dans le rapport.

Tests

Facultatif. La liste des descripteurs de test. Tous les tests spécifiés dans cette liste doivent réussir pour que la fonctionnalité soit prise en charge.

Chaque descripteur de test de cette liste utilise l'ID du groupe de test et un ou plusieurs cas de test IDs pour identifier les tests individuels à exécuter à partir d'un groupe de test spécifique. Le descripteur de test utilise le format suivant :

```
GroupId: group-id  
CaseIds: [test-id, test-id] # optional
```

Vous devez spécifier l'une Tests ou l'autre des fonctionnalités de la Features liste ou OneOfTests pour chacune d'entre elles.

OneOfTests

Facultatif. La liste des descripteurs de test. Au moins l'un des tests spécifiés dans cette liste doit réussir pour que la fonctionnalité soit prise en charge.

Chaque descripteur de test de cette liste utilise l'ID du groupe de test et un ou plusieurs cas de test IDs pour identifier les tests individuels à exécuter à partir d'un groupe de test spécifique. Le descripteur de test utilise le format suivant :

```
GroupId: group-id  
CaseIds: [test-id, test-id] # optional
```

Vous devez spécifier l'une Tests ou l'autre des fonctionnalités de la Features liste ou OneOfTests pour chacune d'entre elles.

IsRequired

La valeur booléenne qui définit si la fonctionnalité est requise dans le rapport de test. La valeur par défaut est false.

Exemple

Contexte de l'orchestrateur de tests

Le contexte de l'orchestrateur de test est un document JSON en lecture seule qui contient les données mises à la disposition de l'orchestrateur de test pendant l'exécution. Le contexte de l'orchestrateur de test n'est accessible que depuis l'orchestrateur de test et contient des informations qui déterminent le flux de test. Par exemple, vous pouvez utiliser les informations configurées par les testeurs dans le `userdata.json` fichier pour déterminer si un test spécifique est requis pour être exécuté.

Le contexte de l'orchestrateur de test utilise le format suivant :

```
{
  "pool": {
    <device-json-pool-element>
  },
  "userData": {
    <userdata-json-content>
  },
  "config": {
    <config-json-content>
  }
}
```

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` fichier.

userData

Informations contenues dans le `userdata.json` fichier.

config

Informations contenues dans le `config.json` fichier.

Vous pouvez interroger le contexte à l'aide de la JSONPath notation. La syntaxe des JSONPath requêtes dans les définitions d'état est `{{query}}`. Lorsque vous accédez aux données depuis le contexte de l'orchestrateur de test, assurez-vous que chaque valeur correspond à une chaîne, un nombre ou un booléen.

Pour plus d'informations sur l'utilisation de la JSONPath notation pour accéder aux données depuis le contexte, consultez [Utiliser le contexte IDT](#).

Configuration de la machine d'état IDT

Important

À partir de IDT v4.5.1, cette machine à états est obsolète. Nous vous recommandons vivement d'utiliser le nouvel orchestrateur de test. Pour de plus amples informations, veuillez consulter [Configuration de l'orchestrateur de test 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 à états pour vous. La machine à états par défaut exécute les fonctions suivantes :

- Permet aux testeurs de sélectionner et d'exécuter des groupes de tests spécifiques, plutôt que la suite de tests complète.
- Si aucun groupe de test spécifique n'est sélectionné, exécute chaque groupe de test de la suite de tests dans un ordre aléatoire.
- Génère des rapports et imprime un résumé de console présentant 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 doit effectuer IDT, par exemple exécuter un groupe de test ou produire un fichier de rapport.
- Le passage à 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 l'un Succeed ou l'autreFail.

Format de la machine à états

Vous pouvez utiliser le modèle suivant pour configurer votre propre *<custom-test-suite-folder>*/suite/state_machine.json 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 à états.

StartAt

Nom de l'état dans lequel IDT commence à exécuter la suite de tests. La valeur de `StartAt` doit être définie sur l'un des états répertoriés dans l'`States` objet.

States

Objet qui associe les noms d'état définis par l'utilisateur à des états IDT valides. Chaque État. *state-name* l'objet contient la définition d'un état valide mappé à. *state-name*

L'`States` objet doit inclure les `Fail` états `Succeed` et. Pour plus d'informations sur les états valides, consultez [États valides et définitions d'états](#).

États valides et définitions d'états

Cette section décrit les définitions d'état de tous les états valides qui peuvent être utilisés dans la machine d'état IDT. Certains des états suivants prennent en charge les configurations au niveau du scénario de test. Toutefois, nous vous recommandons de configurer les règles de transition d'état au niveau du groupe de test plutôt qu'au niveau du cas de test, sauf si cela est absolument nécessaire.

Définitions des États

- [RunTask](#)
- [Choice](#)
- [Parallèle](#)
- [AddProductFeatures](#)
- [Rapport](#)
- [LogMessage](#)
- [SelectGroup](#)
- [Fail](#)
- [Succeed](#)

RunTask

L'RunTaskÉtat exécute des scénarios de test à partir 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 vers lequel passer après l'exécution des actions dans 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 testeur.

TestCases

Facultatif. Un tableau de scénarios de test IDs provenant du groupe spécifié dans `TestGroup`. Sur la base des valeurs de `TestGroup` et `TestCases`, IDT détermine le comportement d'exécution du test comme suit :

- Lorsque `TestGroup` les deux `TestCases` sont spécifiés, IDT exécute les cas de test spécifiés à partir du groupe de test.
- Lorsqu'ils `TestCases` sont spécifiés mais `TestGroup` non spécifiés, IDT exécute les cas de test spécifiés.
- Lorsqu'il `TestGroup` est spécifié, mais `TestCases` non spécifié, IDT exécute tous les cas de test au sein du groupe de test spécifié.
- Lorsque ni l'`TestGroup` ni l'autre n'`TestCases` est spécifié, IDT exécute tous les cas de test à partir du groupe de test sélectionné par le lanceur de tests dans la CLI IDT. Pour permettre la sélection de groupes pour les testeurs, vous devez inclure `RunTask` les deux `Choice` états dans votre `state_machine.json` fichier. Pour un exemple de la façon dont cela fonctionne, voir [Exemple de machine à états : exécuter des groupes de test sélectionnés par l'utilisateur](#).

Pour plus d'informations sur l'activation des commandes IDT CLI pour les testeurs, consultez [the section called "Activer les commandes IDT CLI"](#).

ResultVar

Nom de la variable de contexte à définir avec les résultats du test. Ne spécifiez pas cette valeur si vous n'avez pas indiqué de valeur pour `TestGroup`. IDT définit la valeur de la variable que vous définissez dans `true` ou `ResultVar` en `false` fonction des éléments suivants :

- Si le nom de la variable est au format `text_text_passed`, la valeur est définie de manière à indiquer si tous les tests du premier groupe de tests ont été réussis ou ignorés.
- Dans tous les autres cas, la valeur est définie selon que tous les tests de tous les groupes de tests ont été réussis ou ignorés.

Généralement, vous utiliserez `RunTask state` pour spécifier un identifiant de groupe de test sans spécifier de cas de test individuel IDs, 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 parallèle, dans un ordre aléatoire. Toutefois, si tous les scénarios de test nécessitent l'exécution d'un périphérique et qu'un seul périphérique est disponible, les scénarios de test seront exécutés de manière séquentielle.

Gestion des erreurs

Si l'un des groupes de tests ou des scénarios de test IDs spécifiés n'est pas valide, cet état génère l'erreur d'`RunTaskError`exécution. Si l'état rencontre une erreur d'exécution, il définit également la `hasExecutionError` variable dans le contexte de la machine à états sur `true`.

Choice

L'`Choice`état vous permet de définir dynamiquement l'état suivant vers lequel passer 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 par défaut vers lequel passer si aucune des expressions définies dans `Choices` peut être évaluée `true`.

FallthroughOnError

Facultatif. Spécifie le comportement lorsque l'état rencontre une erreur lors de l'évaluation des expressions. Définissez cette `true` valeur si vous souhaitez ignorer une expression si l'évaluation entraîne une erreur. Si aucune expression ne correspond, la machine à états passe à l'`Default`état. Si la `FallthroughOnError` valeur n'est pas spécifiée, elle est définie par défaut sur `false`.

Choices

Tableau d'expressions et d'états permettant de déterminer l'état vers lequel passer après avoir exécuté les actions dans l'état actuel.

Choices.Expression

Chaîne d'expression dont l'évaluation correspond à une valeur booléenne. Si l'expression est évaluée à `true`, la machine à états passe à l'état défini dans `Choices.Next`. Les chaînes d'expression récupèrent les valeurs du contexte de la machine à états, puis exécutent des opérations sur celles-ci pour obtenir une valeur booléenne. Pour plus d'informations sur l'accès au contexte de la machine à états, consultez [Contexte de la machine à états](#).

Choices.Next

Nom de l'état vers lequel passer si l'expression définie dans `Choices.Expression` correspond à `true`.

Gestion des erreurs

L'`Choice` 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 à états.
- 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` bloc pour gérer les erreurs dans cet état. Si vous souhaitez arrêter l'exécution de la machine d'état lorsqu'elle rencontre une erreur, vous devez `FallthroughOnError` définir sur `false`. Toutefois, nous vous recommandons de configurer l'une des `FallthroughOnError` options suivantes et `true`, en fonction de votre cas d'utilisation, d'effectuer l'une des opérations suivantes :

- Si une variable à laquelle vous accédez ne devrait pas exister dans certains cas, utilisez la valeur de `Default` et des `Choices` blocs supplémentaires pour spécifier l'état suivant.
- Si une variable à laquelle vous accédez doit toujours exister, définissez son `Default` état sur `Fail`.

Parallèle

L'`Parallel` état vous permet de définir et d'exécuter de nouvelles machines à états en parallèle les unes avec les 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 vers lequel passer après l'exécution des actions dans l'état actuel.

Branches

Un tableau de définitions de machines à états à exécuter. Chaque définition de machine à états doit contenir ses propres `Fail` états `StartAtSucceed`, et. Les définitions de machines à états de ce tableau ne peuvent pas faire référence à des états en dehors de leur propre définition.

Note

Comme chaque machine d'état de branche partage le même contexte de machine d'état, le fait de définir des variables dans une branche puis de lire ces variables dans une autre branche peut entraîner un comportement inattendu.

L'`Parallel` état passe à l'état suivant uniquement après avoir exécuté toutes les machines d'état de branche. Chaque état nécessitant un appareil attendra de fonctionner 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 parallèle. Si un nombre suffisant de périphériques ne sont pas disponibles, les scénarios de test seront exécutés de manière séquentielle. Comme les scénarios de test sont exécutés dans un ordre aléatoire lorsqu'ils sont exécutés en parallèle, différents appareils peuvent être utilisés pour exécuter des tests à partir du même groupe de test.

Gestion des erreurs

Assurez-vous que la machine d'état de branche et la machine d'état parent passent à l'`Fail` état pour gérer les erreurs d'exécution.

Comme les machines d'état de branche ne transmettent pas d'erreurs d'exécution à la machine d'état parent, vous ne pouvez pas utiliser de `Catch` bloc pour gérer les erreurs d'exécution dans les machines d'état de branche. Utilisez plutôt la `hasExecutionErrors` valeur dans le contexte de la machine à états partagés. Pour un exemple de la façon dont cela fonctionne, voir [Exemple de machine à états : exécuter deux groupes de tests en parallèle](#).

AddProductFeatures

L'`AddProductFeatures` état vous permet d'ajouter des fonctionnalités du produit au `awsiotdevicetester_report.xml` fichier généré par IDT.

Les fonctionnalités d'un produit sont des informations définies par l'utilisateur concernant des critères spécifiques auxquels un appareil peut répondre. Par exemple, la fonctionnalité MQTT du produit peut indiquer que l'appareil publie correctement les messages MQTT. Dans le rapport, les fonctionnalités du produit sont définies sous `supported` la forme d'une valeur ou d'une valeur personnalisée, en fonction de la réussite des tests spécifiés. `not-supported`

Note

L'`AddProductFeatures` État ne produit pas de rapports par lui-même. Cet état doit passer à l'[Report état permettant](#) de 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 vers lequel passer après l'exécution des actions dans l'état actuel.

Features

Un ensemble de fonctionnalités du produit à afficher dans le `awsiotdevicetester_report.xml` fichier.

Feature

Le nom de la fonctionnalité

FeatureValue

Facultatif. La valeur personnalisée à utiliser dans le rapport à la place de `supported`. Si cette valeur n'est pas spécifiée, la valeur de la fonction est définie sur `supported` ou en fonction des résultats des tests `not-supported`.

Si vous utilisez une valeur personnalisée pour `FeatureValue`, vous pouvez tester la même fonctionnalité dans des conditions différentes, et IDT concatène les valeurs des caractéristiques pour les conditions prises en charge. Par exemple, l'extrait suivant montre l'entité `MyFeature` avec deux valeurs de fonction 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 la fonctionnalité est définie sur `first-feature-supported`, `second-feature-supported`.

Groups

Facultatif. Un ensemble de groupes de test IDs. Tous les tests au sein de chaque groupe de test spécifié doivent réussir pour que la fonctionnalité soit prise en charge.

OneOfGroups

Facultatif. Un ensemble de groupes de test IDs. 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. Un ensemble de scénarios de test IDs. Si vous spécifiez cette valeur, les règles suivantes s'appliquent :

- Tous les scénarios de test spécifiés doivent être réussis pour que la fonctionnalité soit prise en charge.
- `Groups` ne doit contenir qu'un seul identifiant de groupe de test.
- `OneOfGroups` ne doit pas être spécifiée.

IsRequired

Facultatif. Réglez `false` sur pour marquer cette fonctionnalité comme fonctionnalité facultative dans le rapport. La valeur par défaut est `true`.

ExecutionMethods

Facultatif. Tableau de méthodes d'exécution correspondant à la `protocol` valeur spécifiée dans le `device.json` fichier. Si cette valeur est spécifiée, les testeurs doivent spécifier une `protocol` valeur correspondant à l'une des valeurs de ce tableau pour inclure la fonctionnalité dans le rapport. Si cette valeur n'est pas spécifiée, la fonctionnalité sera toujours incluse dans le rapport.

Pour utiliser l'`AddProductFeatures` état, vous devez définir la valeur de `ResultVar` in the `RunTask` state sur l'une des valeurs suivantes :

- Si vous avez spécifié un scénario de test individuel IDs, réglez `ResultVar` sur `group-id_test-id_passed`.

- Si vous n'avez pas spécifié de scénario de test individuel IDs, réglez `ResultVar` sur `group-id_passed`.

L'`AddProductFeatures` état vérifie les résultats des tests de la manière suivante :

- Si vous n'avez spécifié aucun scénario de test IDs, le résultat de chaque groupe de test est déterminé à partir de la valeur de la `group-id_passed` variable dans le contexte de la machine à états.
- Si vous avez spécifié un scénario de test IDs, le résultat de chacun des tests est déterminé à partir de la valeur de la `group-id_test-id_passed` variable dans le contexte de la machine à états.

Gestion des erreurs

Si un identifiant de groupe fourni dans cet état n'est pas un identifiant de groupe valide, cet état entraîne une erreur d'`AddProductFeatures` exécution. Si l'état rencontre une erreur d'exécution, il définit également la `hasExecutionErrors` variable dans le contexte de la machine à états sur `true`.

Rapport

L'`Report` état génère les `awsiotdevicetester_report.xml` fichiers `suite-name_Report.xml` et. 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 vers lequel passer après l'exécution des actions dans l'état actuel.

Vous devez toujours passer à l'`Report` état vers la fin du flux d'exécution des tests afin que les testeurs puissent voir 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 l'erreur d'`ReportError`exécution.

LogMessage

L'`LogMessage`état génère le `test_manager.log` fichier et transmet le message du journal à 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 vers lequel passer après l'exécution des actions dans l'état actuel.

Level

Le niveau d'erreur auquel le message de journal doit être créé. Si vous spécifiez un niveau non valide, cet état génère un message d'erreur et le supprime.

Message

Le message à enregistrer.

SelectGroup

L'`SelectGroup`état met à jour le contexte de la machine à états pour indiquer quels groupes sont sélectionnés. Les valeurs définies par cet état sont utilisées par tous `Choice` les états suivants.

```
{
  "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 vers lequel passer après l'exécution des actions dans l'état actuel.

TestGroups

Un ensemble de groupes de test qui seront marqués comme sélectionnés. Pour chaque ID de groupe de test de ce tableau, la *group-id_selected* variable est définie sur `true` dans le contexte. Assurez-vous de fournir un groupe de test valide IDs car IDT ne valide pas l'existence des groupes spécifiés.

Fail

L'Fail état indique que la machine à états ne s'est pas exécutée correctement. Il s'agit d'un état final pour la machine à états, et chaque définition de machine à états doit inclure cet état.

```
{
  "Type": "Fail"
}
```

Succeed

L'Succeed état indique que la machine à états s'est exécutée correctement. Il s'agit d'un état final pour la machine à états, et chaque définition de machine à états doit inclure cet état.

```
{
  "Type": "Succeed"
}
```

Contexte de la machine à états

Le contexte de la machine à états est un document JSON en lecture seule qui contient les données mises à la disposition de la machine à états pendant l'exécution. Le contexte de la machine à états n'est accessible qu'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 testeurs dans le `userdata.json` fichier pour déterminer si un test spécifique est requis pour être exécuté.

Le contexte de la machine à états 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` fichier.

userData

Informations contenues dans le `userdata.json` fichier.

config

Informations épinglez le `config.json` fichier.

suiteFailed

La valeur est définie sur le `false` démarrage de la machine à états. Si un groupe de test échoue dans un `RunTask` état, cette valeur est définie sur la durée restante `true` de l'exécution de la machine à états.

specificTestGroups

Si le testeur sélectionne des groupes de tests spécifiques à exécuter au lieu de l'ensemble de la suite de tests, cette clé est créée et contient la liste des groupes de tests spécifiques IDs.

`specificTestCases`

Si le lanceur de tests sélectionne des cas de test spécifiques à exécuter au lieu de la suite de tests complète, cette clé est créée et contient la liste des cas de test spécifiques IDs.

`hasExecutionErrors`

Ne se ferme pas au démarrage de la machine à états. Si un état rencontre une erreur d'exécution, cette variable est créée et définie `true` pour la durée restante de l'exécution de la machine à états.

Vous pouvez interroger le contexte à l'aide de la JSONPath notation. La syntaxe des JSONPath requêtes dans les définitions d'état est `{$.query}`. Vous pouvez utiliser des JSONPath requêtes comme chaînes d'espace réservé dans certains États. IDT remplace les chaînes d'espace réservé par la valeur de la JSONPath requête évaluée à partir du contexte. Vous pouvez utiliser des espaces réservés pour les valeurs suivantes :

- La `TestCases` valeur en `RunTask` états.
- `ChoiceÉtat` de `Expression` la valeur.

Lorsque vous accédez aux données depuis le contexte de la machine à états, assurez-vous que les conditions suivantes sont remplies :

- Vos chemins JSON doivent commencer par `$`.
- Chaque valeur doit être évaluée sous la forme d'une chaîne, d'un nombre ou d'un booléen.

Pour plus d'informations sur l'utilisation de la JSONPath notation pour accéder aux données depuis le contexte, consultez [Utiliser le contexte IDT](#).

Erreurs d'exécution

Les erreurs d'exécution sont des erreurs dans la définition de la machine à états que la machine à états rencontre lors de l'exécution d'un état. IDT enregistre les informations relatives à chaque erreur dans le `test_manager.log` fichier et transmet le message de journal à la console.

Vous pouvez utiliser les méthodes suivantes pour gérer les erreurs d'exécution :

- Ajoutez un [Catchbloc](#) dans la définition de l'état.
- Vérifiez la valeur de la [hasExecutionErrorsvaleur](#) dans le contexte de la machine à états.

attraper

Pour l'utiliser `Catch`, ajoutez ce qui suit à la définition de votre 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 détecter. Si une erreur d'exécution correspond à l'une des valeurs spécifiées, la machine à états passe à l'état spécifié dans `Catch.Next`. Consultez la définition de chaque état pour obtenir des informations sur le type d'erreur qu'il produit.

`Catch.Next`

État suivant vers lequel passer si l'état actuel rencontre une erreur d'exécution correspondant à l'une des valeurs spécifiées dans `Catch.ErrorEquals`.

Les blocs de capture sont gérés de manière séquentielle jusqu'à ce qu'un d'entre eux corresponde. Si aucune erreur ne correspond à celles répertoriées dans les blocs `Catch`, les machines d'état continuent de s'exécuter. Les erreurs d'exécution étant le résultat de définitions d'état incorrectes, nous vous recommandons de passer à l'état `Fail` lorsqu'un état rencontre une erreur d'exécution.

`hasExecutionError`

Lorsque certains états rencontrent des erreurs d'exécution, en plus d'émettre l'erreur, ils définissent également la `hasExecutionError` valeur sur `true` dans le contexte de la machine à états. Vous pouvez utiliser cette valeur pour détecter lorsqu'une erreur se produit, puis utiliser un `Choice` état pour faire passer la machine à états à `Fail` cet état.

Cette méthode présente les caractéristiques suivantes.

- La machine à états ne démarre avec aucune valeur assignée à `hasExecutionError`, et cette valeur n'est pas disponible tant qu'un état particulier ne la définit pas. Cela signifie que vous

devez définir explicitement la valeur `FallthroughOnError` to `false` pour les `Choice` états qui accèdent à cette valeur afin d'empêcher l'arrêt de la machine à états si aucune erreur d'exécution ne se produit.

- Une fois défini sur `true`, il n'a `hasExecutionError` est jamais défini sur `false` ni 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 suivants, elle ne fournit pas de valeur significative.
- La `hasExecutionError` valeur est partagée avec toutes les machines d'état de branche de l'`Parallel` état, ce qui peut entraîner des résultats inattendus en fonction de l'ordre dans lequel elle est consultée.

En raison de ces caractéristiques, nous vous déconseillons d'utiliser cette méthode si vous pouvez utiliser un bloc `Catch` à la place.

Exemples de machines à états

Cette section fournit des exemples de configurations de machines à états.

Exemples

- [Exemple de machine à états : exécuter un seul groupe de test](#)
- [Exemple de machine à états : exécuter des groupes de test sélectionnés par l'utilisateur](#)
- [Exemple de machine à états : exécuter un seul groupe de test avec les fonctionnalités du produit](#)
- [Exemple de machine à états : exécuter deux groupes de tests en parallèle](#)

Exemple de machine à états : exécuter un seul groupe de test

Cette machine à états :

- Exécute le groupe de test avec un identifiant `GroupA`, qui doit être présent dans la suite dans un `group.json` fichier.
- Vérifie l'absence d'erreurs d'exécution et effectue la transition vers le `Fail` cas échéant.
- Génère un rapport et passe à `Succeed` s'il n'y a pas d'erreur, et `Fail` sinon.

```
{
  "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 à états : exécuter des groupes de test sélectionnés par l'utilisateur

Cette machine à états :

- Vérifie si le testeur a sélectionné des groupes de test spécifiques. La machine à états ne vérifie pas les cas de test spécifiques car les testeurs ne peuvent pas sélectionner de cas de test sans sélectionner également un groupe de test.
- Si des groupes de test sont sélectionnés :

- Exécute les scénarios de test au sein des groupes de test sélectionnés. Pour ce faire, la machine d'état ne spécifie pas explicitement de groupes de test ou de cas de test dans l'RunTask état.
- Génère un rapport après avoir exécuté tous les tests et sorties.
- Si les groupes de test ne sont pas sélectionnés :
 - Exécute des tests dans le groupe de testGroupA.
 - 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 à états : exécuter un seul groupe de test avec les fonctionnalités du produit

Cette machine à états :

- Exécute le groupe de testGroupA.
- Vérifie l'absence d'erreurs d'exécution et effectue la transition vers le `Fail` cas échéant.
- Ajoute la `FeatureThatDependsOnGroupA` fonctionnalité au `awsiotdevicetester_report.xml` fichier :
 - En `GroupA` cas de réussite, la fonctionnalité est réglée `sursupported`.
 - La fonctionnalité n'est pas marquée comme facultative dans le rapport.
- Génère un rapport et passe à `Succeed` s'il n'y a pas d'erreur, et `Fail` sinon

```
{
  "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 à états : exécuter deux groupes de tests en parallèle

Cette machine à états :

- Exécute les groupes GroupA de GroupB test et de test en parallèle. Les ResultVar variables stockées dans le contexte par les RunTask états dans les machines à états de branche sont disponibles pour l'AddProductFeatures état.
- Vérifie l'absence d'erreurs d'exécution et effectue la transition vers le Fail cas échéant. 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 en fonction des groupes qui passent
 - En GroupA cas de réussite, la fonctionnalité est réglée sursupported.
 - La fonctionnalité n'est pas marquée comme facultative dans le rapport.
- Génère un rapport et passe à Succeed s'il n'y a pas d'erreur, et Fail sinon

Si deux appareils sont configurés dans le pool de périphériques, GroupA les deux GroupB peuvent fonctionner en même temps. Toutefois, si l'GroupA ou l'autre GroupB contient plusieurs tests, les deux appareils peuvent être affectés à ces tests. Si un seul appareil est configuré, les groupes de test s'exécuteront 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"
}
}
```

Création d'exécutables de scénarios de test IDT

Vous pouvez créer et placer des exécutables de scénarios 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 provenant des `test.json` fichiers pour déterminer les tests à exécuter, vous pouvez créer un seul scénario de test exécutable 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 en fonction de commandes spécifiées, vous devez créer un fichier exécutable 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 scénario de test en conséquence. Assurez-vous de fournir le chemin exécutable du scénario de test correct dans chaque `test.json` fichier et que le fichier exécutable spécifié s'exécute correctement.

Lorsque tous les appareils sont prêts pour l'exécution d'un scénario de test, IDT lit les fichiers suivants :

- Le `test.json` scénario de test sélectionné détermine les processus à démarrer et les variables d'environnement à définir.

- Le `suite.json` for the test suite détermine les variables d'environnement à définir.

IDT lance le processus exécutable de test requis en fonction des commandes et des arguments spécifiés dans le `test.json` fichier, et transmet les variables d'environnement requises au processus.

Utiliser le SDK du client IDT

Le client IDT vous permet SDKs de simplifier la façon dont vous écrivez la logique de test dans votre exécutable de test grâce à des commandes d'API que vous pouvez utiliser pour interagir avec IDT et vos appareils testés. IDT fournit actuellement les services suivants : SDKs

- SDK client IDT pour Python
- SDK client IDT pour Go
- SDK client IDT pour Java

Ils se SDKs trouvent dans le `<device-tester-extract-location>/sdks` dossier. Lorsque vous créez un nouvel exécutable de scénario de test, vous devez copier le SDK que vous souhaitez utiliser dans le dossier contenant votre exécutable de scénario 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 les exécutables de vos scénarios de test.

Dans cette section

- [Interaction avec les appareils](#)
- [Interaction avec IDT](#)
- [Interaction avec l'hôte](#)

Interaction avec les appareils

Les commandes suivantes vous permettent de communiquer avec l'appareil testé sans avoir à implémenter de fonctions supplémentaires d'interaction avec l'appareil et de gestion de la connectivité.

ExecuteOnDevice

Permet aux suites de tests d'exécuter des commandes shell sur un appareil 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 appareil 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

IDT ne gérant pas les connexions directes aux appareils établies à l'aide des informations d'accès aux appareils issues du contexte, nous vous recommandons d'utiliser ces commandes API d'interaction avec les appareils dans les exécutables de vos scénarios 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 avec l'appareil à partir de la suite de tests. Pour établir une connexion directe, récupérez les informations dans les `resource.devices.connectivity` champs `device.connectivity` et pour votre appareil testé et pour les périphériques ressources, respectivement. Pour plus d'informations sur l'utilisation du contexte IDT, consultez [Utiliser le contexte IDT](#).

Interaction avec IDT

Les commandes suivantes permettent à vos suites de tests de communiquer avec IDT.

PollForNotifications

Permet aux suites de tests de vérifier les notifications provenant d'IDT.

GetContextValue et GetContextString

Permet aux suites de tests de récupérer des valeurs à partir du contexte IDT. Pour de plus amples informations, veuillez consulter [Utiliser le contexte IDT](#).

SendResult

Permet aux suites de tests de communiquer les résultats des scénarios de test à IDT. Cette commande doit être appelée à la fin de chaque scénario de test dans une suite de tests.

Interaction avec l'hôte

La commande suivante permet à vos suites de tests de communiquer avec la machine hôte.

`PollForNotifications`

Permet aux suites de tests de vérifier les notifications provenant d'IDT.

`GetContextValue` et `GetContextString`

Permet aux suites de tests de récupérer des valeurs à partir du contexte IDT. Pour de plus amples informations, veuillez consulter [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 des exécutables des scénarios de test.

Activer les commandes IDT CLI

La `run-suite` commande IDT CLI fournit plusieurs options qui permettent au lanceur de tests de personnaliser l'exécution des tests. Pour permettre aux testeurs d'utiliser ces options pour exécuter votre suite de tests personnalisée, vous implémentez le support de la CLI IDT. Si vous n'implémentez pas le support, les testeurs pourront toujours exécuter des tests, mais certaines options de la CLI ne fonctionneront pas correctement. Pour offrir une expérience client optimale, nous vous recommandons de mettre en œuvre la prise en charge des arguments suivants pour la `run-suite` commande dans la CLI IDT :

`timeout-multiplier`

Spécifie une valeur supérieure à 1,0 qui sera appliquée à tous les délais d'expiration lors de l'exécution des tests.

Les testeurs peuvent utiliser cet argument pour augmenter le délai d'expiration des scénarios de test qu'ils souhaitent exécuter. Lorsqu'un lanceur de tests spécifie cet argument dans sa `run-suite` commande, IDT l'utilise pour calculer la valeur de la variable d'environnement `IDT_TEST_TIMEOUT` et définit le champ dans le contexte IDT. `config.timeoutMultiplier` Pour étayer cet argument, vous devez procéder comme suit :

- Au lieu d'utiliser directement la valeur de délai d'attente du `test.json` fichier, lisez la variable d'environnement `IDT_TEST_TIMEOUT` pour obtenir la valeur de délai d'expiration correctement calculée.

- Récupérez la `config.timeoutMultiplier` valeur dans le contexte IDT et appliquez-la à des délais d'expiration prolongés.

Pour plus d'informations sur la fermeture anticipée en raison d'événements liés au délai imparti, consultez. [Spécifier le comportement de sortie](#)

stop-on-first-failure

Spécifie qu'IDT doit arrêter d'exécuter tous les tests en cas d'échec.

Lorsqu'un lanceur de tests spécifie cet argument dans sa `run-suite` commande, IDT arrête d'exécuter les tests dès qu'il rencontre un échec. Toutefois, si les scénarios de test sont exécutés en parallèle, cela peut entraîner des résultats inattendus. Pour mettre en œuvre le support, assurez-vous que si IDT rencontre cet événement, votre logique de test indique à tous les scénarios de test en cours d'exécution de s'arrêter, de nettoyer les ressources temporaires et de communiquer un résultat de test à IDT. Pour plus d'informations sur la gestion anticipée des défaillances, consultez [Spécifier le comportement de sortie](#).

group-id et test-id

Spécifie qu'IDT ne doit exécuter que les groupes de tests ou les cas de test sélectionnés.

Les testeurs peuvent utiliser ces arguments avec leur `run-suite` commande pour spécifier le comportement d'exécution du test suivant :

- Exécutez tous les tests au sein des groupes de tests spécifiés.
- Exécutez une sélection de tests au sein d'un groupe de tests spécifié.

Pour étayer ces arguments, l'orchestrateur de tests de votre suite de tests doit inclure un ensemble `RunTask` et des `Choice` états spécifiques dans votre orchestrateur de tests. Si vous n'utilisez pas de machine à états personnalisée, l'orchestrateur de test IDT par défaut inclut les états requis pour vous et vous n'avez pas besoin de prendre d'autres mesures. Toutefois, si vous utilisez un orchestrateur de test personnalisé, utilisez-le [Exemple de machine à états : exécuter des groupes de test sélectionnés par l'utilisateur](#) comme exemple pour ajouter les états requis dans votre orchestrateur de test.

Pour plus d'informations sur les commandes IDT CLI, consultez [Déboguer et exécuter des suites de tests personnalisées](#).

Rédiger des journaux d'événements

Pendant le test, vous envoyez des données à la console `stdout` et vous `stderr` devez y écrire des journaux d'événements et des messages d'erreur. Pour plus d'informations sur le format des messages de console, consultez [Format des messages de console](#).

Lorsque l'IDT a terminé d'exécuter la suite de tests, ces informations sont également disponibles dans le `test_manager.log` fichier situé dans le `<devicetester-extract-location>/results/<execution-id>/logs` dossier.

Vous pouvez configurer chaque scénario de test pour écrire les journaux de son exécution, y compris les journaux du périphérique testé, dans le `<group-id>_<test-id>` fichier situé dans le `<device-tester-extract-location>/results/<execution-id>/logs` dossier. Pour ce faire, récupérez le chemin d'accès au fichier journal à partir du contexte IDT contenant la `testData.logFilePath` requête, créez un fichier sur ce chemin et inscrivez le contenu que vous souhaitez y insérer. 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 scénario de test, aucun fichier n'est généré pour ce scénario de test.

Vous pouvez également configurer votre exécutable texte pour créer des fichiers journaux supplémentaires, selon les besoins, dans le `<device-tester-extract-location>/logs` dossier. Nous vous recommandons de spécifier des préfixes uniques pour les noms de fichiers journaux afin que vos fichiers ne soient pas remplacés.

Signaler les résultats à IDT

IDT écrit les résultats des tests dans les fichiers `awsiotdevicetester_report.xml` et les `suite-name_report.xml` fichiers. 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, voir [Consulter les résultats et les journaux des tests IDT](#)

Pour renseigner le contenu du `suite-name_report.xml` fichier, vous devez utiliser la `SendResult` commande pour communiquer les résultats des tests à IDT avant la fin de l'exécution du test. Si IDT ne trouve pas les résultats d'un test, il émet une erreur pour le scénario 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 communiquez pas les 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 le `testData.testArtifactsPath` fichier 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 JUnit XML, 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 de les analyser puis d'utiliser l'API pour les envoyer à IDT.

Si vous utilisez cette méthode, assurez-vous que votre logique de test résume correctement les résultats du test et formatez votre fichier de résultats dans le même format que le `suite-name_report.xml` fichier. IDT n'effectue aucune validation des données que vous fournissez, sauf dans les cas suivants :

- IDT ignore toutes les propriétés de la `testsuites` balise. Au lieu de cela, il calcule les propriétés des balises à partir des résultats d'autres groupes de test rapportés.
- Au moins une `testsuite` balise doit figurer à l'intérieur `testsuites`.

Étant donné qu'IDT utilise le même dossier d'artefacts pour tous les scénarios de test et ne supprime pas les fichiers de résultats entre les tests, cette méthode peut également entraîner des rapports erronés si IDT lit le mauvais fichier. Nous vous recommandons d'utiliser le même nom pour le fichier de résultats XML généré dans tous les scénarios de test afin de remplacer les résultats de chaque scénario de test et de vous assurer que les résultats corrects sont disponibles pour IDT. Bien que vous puissiez utiliser une approche mixte pour créer des rapports dans votre suite de tests, c'est-à-dire utiliser un fichier de résultats XML pour certains cas de test et soumettre les résultats via l'API pour d'autres, nous ne recommandons pas cette approche.

Spécifier le comportement de sortie

Configurez votre exécutable texte pour qu'il se ferme toujours avec un code de sortie de 0, même si un scénario de test indique un échec ou un résultat d'erreur. Utilisez des codes de sortie différents de zéro uniquement pour indiquer qu'un scénario de test n'a pas été exécuté ou si l'exécutable du scénario de test n'a pas pu communiquer de résultats à IDT. Lorsque IDT reçoit un code de sortie différent de zéro, cela indique que le scénario de test a rencontré une erreur qui l'a empêché de s'exécuter.

IDT peut demander ou s'attendre à ce qu'un scénario de test cesse de s'exécuter avant qu'il ne soit terminé dans les événements suivants. Utilisez ces informations pour configurer le fichier exécutable de votre scénario de test afin de détecter chacun de ces événements dans le scénario de test :

Expiration

Se produit lorsqu'un scénario de test s'exécute pendant une durée supérieure à la valeur de délai spécifiée dans le `test.json` fichier. Si le lanceur de test a utilisé l'`timeout-multiplier` argument pour spécifier un multiplicateur de délai d'attente, IDT calcule la valeur du délai d'expiration 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 du délai d'attente 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 temporisateur approprié.

Interrompre

Survient lorsque le lanceur de test interrompt l'IDT. Par exemple, en appuyant sur `Ctrl+C`.

Étant donné que les terminaux propagent les signaux à tous les processus enfants, vous pouvez simplement configurer un gestionnaire de signaux dans vos scénarios de test pour détecter les signaux d'interruption.

Vous pouvez également interroger régulièrement l'API pour vérifier la valeur du `CancellationRequested` booléen dans la réponse de l'`PollForNotificationsAPI`. Lorsque IDT reçoit un signal d'interruption, il définit la valeur du `CancellationRequested` booléen sur `true`.

Arrêt dès le premier échec

Se produit lorsqu'un scénario de test exécuté en parallèle avec le scénario de test en cours échoue et que le lanceur de test a utilisé l'`stop-on-first-failure` argument pour spécifier qu'IDT doit s'arrêter en cas de défaillance.

Pour détecter cet événement, vous pouvez interroger régulièrement l'API afin de vérifier la valeur du `CancellationRequested` booléen dans la réponse de l'`PollForNotificationsAPI`. Lorsqu'IDT rencontre une défaillance et est configuré pour s'arrêter lors du premier échec, il définit la valeur du `CancellationRequested` booléen sur `true`.

Lorsque l'un de ces événements se produit, IDT attend 5 minutes que les scénarios de test en cours soient terminés. Si tous les scénarios de test en cours ne se terminent pas dans les 5 minutes, IDT force l'arrêt de chacun de leurs processus. Si IDT n'a pas reçu les résultats des tests avant la fin des processus, il marquera les cas de test comme ayant expiré. Il est recommandé de veiller à ce que vos scénarios de test exécutent les actions suivantes lorsqu'ils rencontrent l'un des événements :

1. Arrêtez d'exécuter la logique de test normale.
2. Nettoyez toutes les ressources temporaires, telles que les artefacts de test présents sur l'appareil testé.
3. Signalez un résultat de test à IDT, tel qu'un échec ou une erreur.
4. Sortir.

Utiliser le contexte IDT

Lorsque IDT exécute une suite de tests, celle-ci peut accéder à un ensemble de données qui peuvent être utilisées pour déterminer le mode d'exécution de chaque test. Ces données sont appelées contexte IDT. Par exemple, la configuration des données utilisateur fournie par les testeurs dans un `userdata.json` fichier est mise 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 récupérer des données et écrire des données dans le contexte à l'aide de types de données JSON standard tels que des objets, des tableaux, des nombres, etc.

Schéma de contexte

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 contenues dans le [config.json](#) fichier. Le config champ contient également le champ supplémentaire suivant :

`config.timeoutMultiplier`

Le multiplicateur pour toute valeur de délai d'attente utilisée par la suite de tests. Cette valeur est spécifiée par le lanceur de tests à partir de la CLI IDT. La valeur par défaut est 1.

device

Informations sur le périphérique sélectionné pour le test. Ces informations sont équivalentes à l'élément du devices tableau dans le [device.json](#) fichier du périphérique 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` fichier pour le pool de périphériques sélectionné.

resource

Informations sur les périphériques de ressources contenues dans le `resource.json` fichier.

`resource.devices`

Ces informations sont équivalentes au `devices` tableau défini dans le `resource.json` fichier. Chaque `devices` élément inclut le champ supplémentaire suivant :

`resource.device.name`

Nom du périphérique ressource. Cette valeur est définie sur la `requiredResource.name` valeur du `test.json` fichier.

`testData.awsCredentials`

Les AWS informations d'identification utilisées par le test pour se connecter au AWS cloud. Ces informations sont extraites du `config.json` fichier.

`testData.logFilePath`

Le chemin d'accès au fichier journal dans lequel le scénario de test écrit les messages de journal. La suite de tests crée ce fichier s'il n'existe pas.

`userData`

Informations fournies par le testeur dans le [userdata.json](#) fichier.

Accédez aux données dans le contexte

Vous pouvez interroger le contexte en utilisant la JSONPath notation de vos fichiers JSON et de votre exécutable texte avec le `GetContextValue` et `GetContextString` APIs. La syntaxe des JSONPath chaînes permettant d'accéder au contexte IDT varie comme suit :

- Dans `suite.json` et `etest.json`, tu utilises `{{query}}`. En d'autres termes, n'utilisez pas l'élément racine `$`. pour démarrer votre expression.
- Dans `test_orchestrator.yaml`, tu utilises `{{query}}`.

Si vous utilisez la machine à états obsolète, alors dans `state_machine.json`, vous utilisez `{{$.query}}`

- Dans les commandes d'API, vous utilisez `query` ou `{{$.query}}`, selon la commande. Pour plus d'informations, consultez la documentation intégrée dans le SDKs.

Le tableau suivant décrit les opérateurs d'une JSONPath expression typique :

Opérateur	Description
\$	L'élément racine. Comme la valeur de contexte de premier niveau pour IDT est un objet, vous l'utiliserez généralement \$. pour démarrer vos requêtes.
.childName	Accède à l'élément enfant dont le nom childName provient d'un objet. S'il est appliqué à un tableau, il produit un nouveau tableau avec cet opérateur appliqué à chaque élément. Le nom de l'élément distingue les majuscules et minuscules. Par exemple, la requête pour accéder à la awsRegion valeur de l'configobjet est\$.config.awsRegion.
[start:end]	Filtre les éléments d'un tableau, en récupérant les éléments en commençant par l'startindex et en remontant jusqu'à l'endindex, dans les deux cas inclus.
[index1, index2, ... , indexN]	Filtre les éléments d'un tableau, en récupérant les éléments uniquement à partir des indices spécifiés.
[?(expr)]	Filtre les éléments d'un tableau à l'aide de l'expression. Cette expression doit être évaluée à une valeur booléenne.

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 une valeur personnalisée qui utilise la syntaxe JSON standard.
- `operator` est l'un des opérateurs suivants :
 - `<` (Inférieur à)
 - `<=` (Inférieur ou égal à)
 - `==` (Égal à)

Si la valeur JSONPath ou de votre expression est un tableau, une valeur booléenne ou une valeur d'objet, il s'agit du seul opérateur binaire pris en charge que vous pouvez utiliser.

- `>=` (Supérieur ou égal à)
- `>` (Supérieur à)
- `~=` (Correspondance d'expressions régulières). Pour utiliser cet opérateur dans une expression de filtre, la valeur JSONPath ou sur le côté gauche de votre expression doit être une chaîne et le côté droit doit être une valeur de modèle conforme à la [RE2syntaxe](#).

Vous pouvez utiliser JSONPath des requêtes sous la forme `{{query}}` comme chaînes d'espace réservé dans les `environmentVariables` champs `args` et `test.json` des fichiers et dans les `environmentVariables` champs des `suite.json` fichiers. IDT effectue une recherche contextuelle et remplit les champs avec la valeur évaluée de la requête. Par exemple, dans le `suite.json` fichier, vous pouvez utiliser des chaînes d'espace réservé pour spécifier des valeurs de variables d'environnement qui changent avec chaque scénario de test et IDT remplira les variables d'environnement avec la valeur correcte pour chaque cas de test. Toutefois, lorsque vous utilisez des chaînes d'espace réservé dans des `suite.json` fichiers `test.json` et, les considérations suivantes s'appliquent à vos requêtes :

- Vous devez écrire toutes les occurrences de la `devicePool` clé dans votre requête en minuscules. C'est-à-dire, utilisez `devicepool` plutôt.
- Pour les tableaux, vous ne pouvez utiliser que des tableaux de chaînes. De plus, les tableaux utilisent un format non standard. `item1, item2, ..., itemN` Si le tableau ne contient qu'un seul élément, il est sérialisé en tant que `telitem`, ce qui le rend impossible à distinguer d'un champ de chaîne.
- Vous ne pouvez pas utiliser d'espaces réservés pour récupérer des objets depuis le contexte.

En raison de ces considérations, nous vous recommandons, dans la mesure du possible, d'utiliser l'API pour accéder au contexte de votre logique de test au lieu d'utiliser des chaînes de caractères dans `suite.json` les fichiers `test.json` et. Toutefois, dans certains cas, il peut être plus pratique

d'utiliser des JSONPath espaces réservés pour récupérer des chaînes uniques à définir comme variables d'environnement.

Configuration des paramètres pour les testeurs

Pour exécuter des suites de tests personnalisées, les testeurs 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 situés dans le `<device-tester-extract-location>/configs/` dossier. Si nécessaire, les testeurs doivent également configurer des AWS informations d'identification qu'IDT utilisera pour se connecter au AWS cloud.

En tant que rédacteur de tests, vous devrez configurer ces fichiers pour [débuguer votre suite de tests](#). Vous devez fournir des instructions aux testeurs afin qu'ils puissent configurer les paramètres suivants selon les besoins pour exécuter vos suites de tests.

Configurer device.json

Le `device.json` fichier contient des informations sur les appareils sur lesquels les tests sont exécutés (par exemple, adresse IP, informations de connexion, système d'exploitation et architecture du processeur).

Les testeurs peuvent fournir ces informations à l'aide du `device.json` fichier modèle suivant situé dans le `<device-tester-extract-location>/configs/` dossier.

```
[
  {
    "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 suivre les appareils qualifiés.

 Note

Si vous souhaitez mettre votre carte en vente dans le catalogue des AWS Partner appareils, le SKU que vous spécifiez ici doit correspondre au SKU que vous avez utilisé lors du processus de mise en vente.

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 testeurs des informations sur les noms et les valeurs des fonctionnalités à inclure dans le `device.json` fichier. Par exemple, si vous souhaitez tester un périphérique qui fonctionne comme un serveur MQTT pour d'autres appareils, vous pouvez configurer votre logique de test pour valider les niveaux pris en charge spécifiques pour une fonctionnalité nommée `MQTT_QOS`. Les testeurs fournissent le nom de cette fonctionnalité et définissent la valeur de la fonctionnalité en fonction des niveaux de QOS pris en charge par leur appareil. Vous pouvez récupérer les informations fournies depuis le [contexte IDT](#) avec la `devicePool.features` requête ou depuis le [contexte de l'orchestrateur de test](#) avec la `pool.features` requête.

`features.name`

Nom de la fonctionnalité.

`features.value`

Les valeurs des fonctionnalités prises en charge.

`features.configs`

Paramètres de configuration, si nécessaire, 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 du 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, ainsi que `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.port`

Facultatif. Le 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 le périphérique 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 `ssh`.

`connectivity.containerUser`

Facultatif. Le nom de l'utilisateur à l'intérieur du conteneur. La valeur par défaut est l'utilisateur indiqué dans le Dockerfile.

La valeur par défaut est 22.

Cette propriété s'applique uniquement si `connectivity.protocol` est défini sur `ssh`.

Note

Pour vérifier si les testeurs configurent la mauvaise connexion aux appareils pour un test, vous pouvez la récupérer dans le contexte `pool.Devices[0].Connectivity.Protocol` de l'orchestrateur de test et la comparer à la valeur attendue dans un `Choice` état. Si un protocole incorrect est utilisé, imprimez un message en utilisant `LogMessage` état et passez à l'`Fail` état. Vous pouvez également utiliser un code de gestion des erreurs pour signaler un échec de test pour des types de périphériques incorrects.

(Facultatif) Configurer userdata.json

Le `userdata.json` fichier contient toutes les informations supplémentaires requises par une suite de tests mais qui ne sont pas spécifiées dans le `device.json` fichier. Le format de ce fichier dépend du [userdata_scheme.json](#) fichier défini dans la suite de tests. Si vous êtes rédacteur de tests, assurez-vous de fournir ces informations aux utilisateurs qui exécuteront les suites de tests que vous écrivez.

(Facultatif) Configurer resource.json

Le `resource.json` fichier contient des informations sur les périphériques qui seront utilisés comme périphériques de ressources. Les périphériques ressources sont des appareils nécessaires pour tester certaines fonctionnalités d'un périphérique testé. Par exemple, pour tester la capacité Bluetooth d'un appareil, vous pouvez utiliser un périphérique ressource pour vérifier si votre appareil peut s'y connecter correctement. Les périphériques de ressources sont facultatifs et vous pouvez avoir besoin d'autant de périphériques de ressources que nécessaire. En tant que rédacteur de test, vous utilisez le [fichier test.json](#) pour définir les fonctionnalités du périphérique de ressources requises pour un test. Les testeurs utilisent ensuite le `resource.json` fichier pour 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 testeurs peuvent fournir ces informations à l'aide du `resource.json` fichier modèle suivant situé dans le `<device-tester-extract-location>/configs/` dossier.

```
[
  {
    "id": "<pool-id>",
    "features": [
      {
        "name": "<feature-name>",
        "version": "<feature-version>",
        "jobSlots": <job-slots>
      }
    ],
    "devices": [
      {
        "id": "<device-id>",
        "connectivity": {
          "protocol": "ssh | uart | docker",
          // ssh
          "ip": "<ip-address>",
```

```

    "port": <port-number>,
    "auth": {
        "method": "pki | password",
        "credentials": {
            "user": "<user-name>",
            // pki
            "privKeyPath": "/path/to/private/key",

            // password
            "password": "<password>",
        }
    },

    // uart
    "serialPort": "<serial-port>",

    // docker
    "containerId": "<container-id>",
    "containerUser": "<container-user-name>",
}
}
]
}
]

```

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 les [fichiers test.json](#) de la suite de tests et déterminent les tests à exécuter et la manière de les exécuter. Si la suite de tests ne nécessite aucune fonctionnalité, ce champ n'est pas obligatoire.

features.name

Nom de la fonctionnalité.

`features.version`

La version fonctionnelle.

`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 du 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, ainsi que `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.port`

Facultatif. Le 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 le périphérique 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 `ssh`.

`connectivity.containerUser`

Facultatif. Le nom de l'utilisateur à l'intérieur du conteneur. La valeur par défaut est l'utilisateur indiqué dans le Dockerfile.

La valeur par défaut est 22.

Cette propriété s'applique uniquement si `connectivity.protocol` est défini sur `ssh`.

(Facultatif) Configurer `config.json`

Le `config.json` fichier contient des informations de configuration pour IDT. Généralement, les testeurs n'ont pas besoin de modifier ce fichier, sauf pour fournir leurs informations AWS d'identification utilisateur pour IDT et, éventuellement, pour une AWS région. Si des AWS informations d'identification avec les autorisations requises sont fournies, AWS IoT Device Tester collecte et soumet les statistiques d'utilisation à AWS. Il s'agit d'une fonctionnalité opt-in utilisée pour améliorer la fonctionnalité IDT. Pour de plus amples informations, veuillez consulter [Métriques d'utilisation de l'IDT](#).

Les testeurs peuvent configurer leurs AWS informations 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 au cours d'une session SSH ne sont pas disponibles après la fermeture de cette session. IDT peut utiliser les variables d'`AWS_SECRET_ACCESS_KEY` environnement `AWS_ACCESS_KEY_ID` et pour stocker les informations d'identification AWS

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 les AWS informations d'identification pour IDT, les testeurs modifient la auth section du config.json fichier situé dans le *<device-tester-extract-location>*/configs/ dossier.

```
{
  "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 de ce fichier sont définis par rapport à *<device-tester-extract-location>*.

log.location

Le chemin d'accès au dossier des journaux dans le *<device-tester-extract-location>*.

`configFiles.root`

Le chemin d'accès au dossier contenant les fichiers de configuration.

`configFiles.device`

Le chemin d'accès au `device.json` fichier.

`testPath`

Le chemin d'accès au dossier contenant les suites de tests.

`reportPath`

Le chemin d'accès au dossier qui contiendra les résultats des tests une fois qu'IDT aura exécuté une suite de tests.

`awsRegion`

Facultatif. La AWS région que les suites de tests utiliseront. Si ce paramètre n'est pas défini, les suites de tests utiliseront la région par défaut spécifiée dans chaque suite de tests.

`auth.method`

Méthode utilisée par IDT pour récupérer les AWS informations d'identification. Les valeurs prises en charge sont `file` la récupération des informations d'identification à partir d'un fichier d'informations d'identification et `environment` des informations d'identification à l'aide de variables d'environnement.

`auth.credentials.profile`

Le profil d'informations d'identification à utiliser à partir du 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

Une fois la [configuration requise](#) définie, IDT peut exécuter votre suite de tests. Le temps d'exécution de la suite de tests complète dépend du matériel et de la composition de la suite de tests. À titre de référence, il faut environ 30 minutes pour terminer la suite complète de tests de AWS IoT Greengrass 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 testeurs.

Exécutez IDT en mode debug

Étant donné que les suites de tests dépendent de l'IDT pour interagir avec les appareils, fournir le contexte et recevoir les résultats, vous ne pouvez pas simplement déboguer vos suites de tests dans un IDE sans aucune interaction IDT. Pour ce faire, la CLI IDT fournit la `debug-test-suite` commande qui vous permet d'exécuter IDT en mode de 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 de débogage, IDT ne lance pas réellement la suite de tests ni n'exécute l'orchestrateur de test ; il interagit plutôt avec votre IDE pour répondre aux demandes émanant de la suite de tests exécutée dans l'IDE et imprime les journaux sur la console. L'IDT n'expire pas et attend de sortir jusqu'à ce qu'il soit interrompu manuellement. En mode débogage, IDT n'exécute pas non plus l'orchestrateur de test et ne génère aucun fichier de rapport. Pour déboguer votre suite de tests, vous devez utiliser votre IDE pour fournir certaines informations que IDT obtient généralement à partir des fichiers JSON de configuration. Assurez-vous de fournir les informations suivantes :

- Variables d'environnement et arguments pour chaque test. IDT ne lira pas ces informations depuis `test.json` ou `suite.json`.
- Arguments pour sélectionner les 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 des paramètres requis pour exécuter la suite de tests. Par exemple, si votre suite de tests nécessite le `device.json` `resource.json`, et `user data.json`, assurez-vous de tous les configurer selon vos besoins.
2. Exécutez la commande suivante pour placer IDT en mode de débogage et sélectionnez les appareils nécessaires pour exécuter le 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 traitement des dossiers pour le SDK client IDT.

3. Dans votre IDE, utilisez la debug configuration `run or` 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 suite `test.json` and.
 - c. Définissez les points d'arrêt selon vos besoins.
4. Exécutez la suite de tests dans votre IDE.

Vous pouvez déboguer et réexécuter la suite de tests autant de fois que nécessaire. Le délai d'expiration de l'IDT n'est pas dépassé en mode débogage.

5. Une fois le débogage terminé, interrompez IDT pour quitter le mode de débogage.

Commandes IDT CLI pour exécuter des tests

La section suivante décrit les commandes de la CLI 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, en l'occurrence AWS IoT Greengrass les versions, et les versions de la suite de tests de AWS IoT Greengrass 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`. Le groupe de test à rechercher. Cette option est obligatoire et doit spécifier un groupe unique.

run-suite

Exécute une suite de tests sur un groupe d'appareils. Les options les plus fréquemment utilisées sont les suivantes :

- `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 test à exécuter, sous forme de 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`. Les cas de test à exécuter, sous forme de liste séparée par des virgules. Lorsqu'il est spécifié, `group-id` doit spécifier un seul groupe.
- `pool-id`. Le pool d'appareils à tester. Les testeurs doivent spécifier un pool s'ils ont plusieurs pools d'appareils définis dans votre `device.json` 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 lors du premier échec. 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 requises pour exécuter la suite de tests. Cela n'est obligatoire que si `userdataRequired` est défini sur `true` dans le `suite.json` fichier de 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 debug. Pour de plus amples informations, veuillez consulter [Exécutez IDT en mode debug](#).

Consulter les résultats et les journaux des tests IDT

Cette section décrit le format dans lequel IDT génère les journaux de console et les rapports de test.

Format des messages de console

AWS IoT Device Tester utilise un format standard pour imprimer des messages sur la console lorsqu'elle démarre une suite de tests. L'extrait suivant montre 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 console contiennent les champs suivants :

time

Un horodatage ISO 8601 complet pour l'événement enregistré.

level

Le niveau du message pour l'événement enregistré. En général, le niveau du message enregistré est l'un des niveaux suivants : `error`. IDT émet un `panic` message `fatal` OR s'il rencontre un événement attendu qui entraîne sa fermeture anticipée.

msg

Le message enregistré.

executionId

Chaîne d'identification unique pour le processus IDT en cours. Cet identifiant est utilisé pour différencier les essais IDT individuels.

Les messages de console générés à partir d'une suite de tests fournissent des informations supplémentaires sur le périphérique testé, ainsi que sur la suite de tests, le groupe de test et les scénarios de test exécutés par IDT. L'extrait suivant montre 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 à la suite de tests du message de 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 scénario de test en cours.

Pour imprimer un résumé de test sur la console lorsqu'un IDT a terminé d'exécuter un test, vous devez inclure un [Report état](#) dans votre orchestrateur de test. Le résumé des tests contient des informations sur la suite de tests, les résultats des tests pour chaque groupe exécuté, ainsi que l'emplacement des journaux et des fichiers de rapport générés. L'exemple suivant montre un message récapitulatif du test.

```
===== Test Summary =====
Execution Time:      5m00s
Tests Completed:    4
Tests Passed:       3
Tests Failed:       1
Tests Skipped:      0
-----
Test Groups:
  GroupA:           PASSED
  GroupB:           FAILED
-----
Failed Tests:
  Group Name: GroupB
    Test Name: TestB1
      Reason: Something bad happened
-----
Path to IoT Device Tester Report: /path/to/awsiotdevicetester_report.xml
Path to Test Execution Logs: /path/to/logs
Path to Aggregated JUnit Report: /path/to/MyTestSuite_Report.xml
```

AWS IoT Device Tester schéma de rapport

`awsiotdevicetester_report.xml` est un rapport signé qui contient les informations suivantes :

- La version IDT.
- La version de la suite de tests.
- Signature du rapport et clé utilisées pour signer le rapport.
- Le SKU de l'appareil et le nom du pool d'appareils spécifiés dans le `device.json` 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` 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>
</apnreport>
```

```
</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. Les fonctionnalités marquées comme `required` étant requises pour que la suite de tests valide le dispositif. L'extrait de code suivant montre comment ces informations apparaissent dans le fichier `awsiotdevicetester_report.xml`.

```
<feature name="ssh" value="supported" type="required"></feature>
```

Les fonctionnalités marquées comme `optional` ne sont pas requises pour 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 la suite de tests

Le `suite-name_Result.xml` rapport est au [format JUnit XML](#). Vous pouvez intégrer des plateformes de déploiement/d'intégration continues tels que [Jenkins](#), [Bamboo](#), etc. Le rapport contient un résumé global 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 du rapport figurant à la fois dans le `awsiotdevicetester_report.xml` ou `suite-name_report.xml` répertorie les tests effectués et les 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

Le temps, en secondes, nécessaire à l'exécution de 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 débogage. 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 soumet les statistiques d'utilisation à AWS. Il s'agit d'une fonctionnalité opt-in utilisée pour améliorer la fonctionnalité IDT. IDT collecte des informations telles que les suivantes :

- L' Compte AWS ID utilisé pour exécuter IDT
- Les AWS CLI commandes IDT utilisées pour exécuter les tests
- Les suites de tests exécutées
- Les suites de tests dans le *<device-tester-extract-location>* dossier
- 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 `config.json` fichier 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 : Créer un Compte AWS

Dans cette étape, créez et configurez un Compte AWS. Si vous disposez déjà d'un Compte AWS, passez directement à l'étape [the section called “Étape 2 : Configurer les autorisations pour IDT”](#).

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

Pour vous inscrire à un Compte AWS

1. Ouvrez l'<https://portal.aws.amazon.com/billing/inscription>.
2. Suivez les instructions en ligne.

Dans le cadre de la procédure d'inscription, vous recevrez un appel téléphonique ou un SMS 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 AWS est créé. Par défaut, seul l'utilisateur racine a accès à l'ensemble des Services AWS et des ressources de ce compte. La meilleure pratique de sécurité consiste à attribuer un accès administratif à un utilisateur, et à utiliser uniquement l'utilisateur racine pour effectuer les [tâches nécessitant un accès utilisateur racine](#).

Afin de créer un utilisateur administrateur, choisissez l'une des options suivantes :

Choisissez un moyen de gérer votre administrateur	À	En	Vous pouvez également
Dans IAM	Utiliser des informations d'identification	Suivant les instructions fournies dans Mise en	Configurez l'accès par programmation en configura

Choisissez un moyen de gérer votre administrateur	À	En	Vous pouvez également
Identity Center (Recommandé)	à court terme pour accéder à AWS. C'est conforme aux bonnes pratiques en matière de sécurité. Pour plus d'informations sur les bonnes pratiques, consultez Bonnes pratiques de sécurité dans IAM dans le Guide de l'utilisateur IAM.	route dans le Guide de l'utilisateur AWS IAM Identity Center .	nt le AWS CLI à utiliser AWS IAM Identity Center dans le guide de l'AWS Command Line Interface utilisateur.
Dans IAM (Non recommandé)	Utiliser les informations d'identification à long terme pour accéder à AWS.	Suivant les instructions fournies dans Création d'un utilisateur IAM pour l'accès d'urgence dans le Guide de l'utilisateur IAM.	Configurer l'accès par programmation en suivant les instructions fournies dans Gestion des clés d'accès pour les utilisateurs IAM dans le Guide de l'utilisateur IAM.

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.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot-device-tester:SendMetrics"
      ],
      "Resource": "*"
    }
  ]
}
```

- c. Choisissez Examiner une politique.
 - d. Pour Nom, saisissez **IDTUsageMetricsIAMPermissions**. Sous Résumé, vérifiez les autorisations accordées par votre stratégie.
 - e. Choisissez Create Policy (Créer une politique).
3. Créez un utilisateur IAM et associez-lui des autorisations.
 - 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 Attacher directement les stratégies existantes à l'utilisateur.
 - ii. Recherchez la IAMPermissions politique de IDTUsagemesures 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 (clé d'accès IDs et clés d'accès secrètes), choisissez Afficher à côté 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)

Suivez ces étapes pour utiliser le AWS CLI afin de configurer les autorisations pour IDT pour AWS IoT Greengrass.

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 des autorisations pour gérer l'IDT et AWS IoT Greengrass les rôles.

Linux or Unix

```
aws iam create-policy --policy-name IDTUsageMetricsIAMPermissions --policy-document '{
```

```
"Version": "2012-10-17",
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "iot-device-tester:SendMetrics"
    ],
    "Resource": "*"
  }
]
```

Windows command prompt

```
aws iam create-policy --policy-name IDTUsageMetricsIAMPermissions --policy-
document '{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot-device-tester:SendMetrics"
      ],
      "Resource": "*"
    }
  ]
}'
```

Note

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.

PowerShell

```
aws iam create-policy --policy-name IDTUsageMetricsIAMPermissions --policy-
document '{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot-device-tester:SendMetrics"
      ],
      "Resource": "*"
    }
  ]
}'
```

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. *user-name* Remplacez-le 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 statistiques AWS, 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 les commandes suivantes.

Linux or Unix

```
export AWS_ACCESS_KEY_ID=access-key  
export AWS_SECRET_ACCESS_KEY=secret-access-key
```

Windows Command Prompt (CMD)

```
set AWS_ACCESS_KEY_ID=access-key
```

```
set AWS_SECRET_ACCESS_KEY=secret-access-key
```

PowerShell

```
$env:AWS_ACCESS_KEY_ID="access-key"  
$env:AWS_SECRET_ACCESS_KEY="secret-access-key"
```

- b. Pour utiliser le fichier d'informations d'identification, ajoutez-y les ~/.aws/credentials informations suivantes.

```
[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 de plus amples informations, veuillez consulter [\(Facultatif\) Configurer config.json](#).

Résolution des problèmes liés à IDT pour V2 AWS IoT Greengrass

IDT pour AWS IoT Greengrass V2 écrit les erreurs à différents emplacements en fonction du type d'erreur. IDT écrit les erreurs dans la console, les fichiers journaux et les rapports de test.

Où rechercher les erreurs

Les erreurs de haut niveau s'affichent sur la console pendant l'exécution du test, et un résumé des tests ayant échoué s'affiche lorsque tous les tests sont terminés. awsiotdevicetester_report.xml contient un résumé de toutes les erreurs à l'origine de l'échec d'un test. IDT stocke les fichiers journaux de chaque test dans un répertoire avec un UUID pour l'exécution du test, affiché sur la console pendant l'exécution du test.

Le répertoire des journaux de test IDT est <*device-tester-extract-location*>/results/<*execution-id*>/logs/. Ce répertoire contient les fichiers suivants affichés dans le tableau. Ce paramètre est utile pour le débogage.

Fichier	Description
test_manager.log	Les journaux écrits sur la console pendant l'exécution du test. Le résumé des résultats à la

Fichier	Description
	fin de ce fichier inclut une liste des tests qui ont échoué. Les journaux des erreurs et des avertissements de ce fichier peuvent vous donner des informations sur les défaillances.
<code>test-group-id /test-case-id /test-name .log</code>	Journaux détaillés du test spécifique effectué dans un groupe de test. Pour les tests qui déploient des composants Greengrass, le fichier journal des scénarios de test est appelé. <code>greengrass-test-run.log</code>
<code>test-group-id /test-case-id /greengrass.log</code>	Journaux détaillés AWS IoT Greengrass du logiciel Core. IDT copie ce fichier depuis le périphérique testé lorsqu'il exécute des tests d'installation du logiciel AWS IoT Greengrass Core sur l'appareil. Pour plus d'informations sur les messages contenus dans ce fichier journal, consultez Résolution des problèmes AWS IoT Greengrass V2 .
<code>test-group-id /test-case-id/component-name .log</code>	Journaux détaillés des composants Greengrass déployés lors des tests. IDT copie les fichiers journaux des composants depuis le périphérique testé lorsqu'il exécute des tests qui déploient des composants spécifiques. Le nom du fichier journal de chaque composant correspond au nom du composant déployé. Pour plus d'informations sur les messages contenus dans ce fichier journal, consultez Résolution des problèmes AWS IoT Greengrass V2 .

Résolution des erreurs IDT pour la AWS IoT Greengrass V2

Avant d'exécuter IDT for AWS IoT Greengrass, installez les fichiers de configuration appropriés. Si vous recevez des erreurs d'analyse et de configuration, la première étape consiste à rechercher et à utiliser un modèle de configuration adapté à votre environnement.

Si le problème persiste, consultez les processus de débogage suivants.

Rubriques

- [Erreurs de résolution d'alias](#)
- [Erreurs liées aux conflits](#)
- [Erreur indiquant l'impossibilité de démarrer le test](#)
- [L'image de qualification Docker existe des erreurs](#)
- [Impossible de lire les informations d'identification](#)
- [Erreurs de guidage avec Greengrass PreInstalled](#)
- [Exception de signature non valide](#)
- [Erreurs de qualification liées à l'apprentissage automatique](#)
- [Les déploiements d'Open Test Framework \(OTF\) ont échoué](#)
- [Erreurs d'analyse](#)
- [Erreurs de type « Autorisation refusée »](#)
- [Erreur lors de la génération du rapport de qualification](#)
- [Erreur liée à un paramètre obligatoire manquant](#)
- [Exception de sécurité sur macOS](#)
- [Erreurs de connexion SSH](#)
- [Erreurs de qualification du gestionnaire de flux](#)
- [Erreurs de délai d'attente](#)
- [Erreurs de vérification de version](#)

Erreurs de résolution d'alias

Lorsque vous exécutez des suites de tests personnalisées, l'erreur suivante peut s'afficher dans la console et dans `letest_manager.log`.

```
Couldn't resolve placeholders: couldn't do a json lookup: index out of range
```

Cette erreur peut se produire lorsque les alias configurés dans l'orchestrateur de test IDT ne sont pas résolus correctement ou si les valeurs résolues ne sont pas présentes dans les fichiers de configuration. Pour résoudre cette erreur, assurez-vous que votre `device.json` et `userdata.json` contient les informations correctes requises pour votre suite de tests. Pour plus d'informations sur la configuration requise pour la AWS IoT Greengrass qualification, consultez [Configurer les paramètres IDT pour exécuter la suite de AWS IoT Greengrass qualification](#).

Erreurs liées aux conflits

L'erreur suivante peut s'afficher lorsque vous exécutez la suite de AWS IoT Greengrass qualification simultanément sur plusieurs appareils.

```
ConflictException: Component [com.example.IDTHelloWorld : 1.0.0] for account [account-id] already exists with state: [DEPLOYABLE] { RespMetadata: { StatusCode: 409, RequestID: "id" }, Message_: "Component [com.example.IDTHelloWorld : 1.0.0] for account [account-id] already exists with state: [DEPLOYABLE]" }
```

L'exécution simultanée de tests n'est pas encore prise en charge pour la suite de AWS IoT Greengrass qualification. Exécutez la suite de qualification de manière séquentielle pour chaque appareil.

Erreur indiquant l'impossibilité de démarrer le test

Vous pouvez rencontrer des erreurs indiquant des échecs survenus lors de la tentative de démarrage du test. Il y a plusieurs causes possibles. Par conséquent, procédez de la façon suivante :

- Assurez-vous que le nom du pool indiqué dans votre commande d'exécution existe bien. IDT référence le nom du pool directement à partir de votre `device.json` fichier.
- Assurez-vous que les appareils du groupe ont des paramètres de configuration corrects.

L'image de qualification Docker existe des erreurs

Les tests de qualification du gestionnaire d'applications Docker utilisent l'image du `amazon/amazon-ec2-metadata-mock` conteneur dans Amazon ECR pour qualifier l'appareil testé.

Le message d'erreur suivant peut s'afficher si l'image est déjà présente dans un conteneur Docker sur l'appareil testé.

```
The Docker image amazon/amazon-ec2-metadata-mock:version already exists on the device.
```

Si vous avez déjà téléchargé cette image et exécuté le `amazon/amazon-ec2-metadata-mock` conteneur sur votre appareil, assurez-vous de supprimer cette image de l'appareil testé avant d'exécuter les tests de qualification.

Impossible de lire les informations d'identification

Lorsque vous testez des appareils Windows, vous pouvez rencontrer l'`Failed to read credential` erreur dans le `greengrass.log` fichier si l'utilisateur que vous utilisez pour vous connecter à l'appareil testé n'est pas configuré dans le gestionnaire d'informations d'identification de cet appareil.

Pour résoudre cette erreur, configurez l'utilisateur et le mot de passe de l'utilisateur IDT dans le gestionnaire d'informations d'identification de l'appareil testé.

Pour de plus amples informations, veuillez consulter [Configuration des informations d'identification utilisateur pour les appareils Windows](#).

Erreurs de guidage avec Greengrass PreInstalled

Lors de PreInstalled l'exécution d'IDT avec Greengrass, si vous rencontrez une erreur `Guice ErrorInCustomProvider` ou vérifiez si le `userdata.json` fichier est défini dans le dossier d'`InstalledDirRootOnDevice` installation de Greengrass. IDT vérifie la présence du fichier ci-dessous `effectiveConfig.yaml`. `<InstallationDirRootOnDevice>/config/effectiveConfig.yaml`

Pour de plus amples informations, veuillez consulter [Configuration des informations d'identification utilisateur pour les appareils Windows](#).

Exception de signature non valide

Lorsque vous exécutez des tests de qualification Lambda, vous pouvez rencontrer l'`invalidsignatureexception` erreur si votre machine hôte IDT rencontre des problèmes d'accès au réseau. Réinitialisez votre routeur et relancez les tests.

Erreurs de qualification liées à l'apprentissage automatique

Lorsque vous exécutez des tests de qualification de machine learning (ML), vous risquez de rencontrer des échecs de qualification si votre appareil ne répond pas aux [exigences](#) pour déployer les composants ML AWS fournis. Pour résoudre les erreurs de qualification ML, procédez comme suit :

- Recherchez les détails des erreurs dans les journaux des composants déployés lors du test. Les journaux des composants se trouvent dans le `<device-tester-extract-location>/results/<execution-id>/logs/<test-group-id>` répertoire.
- Ajoutez l'-Dgg.persist=installed.softwareargument au test.json fichier pour le scénario de test qui a échoué. Le test.json fichier se trouve dans `<device-tester-extract-location>/tests/GGV2Q_version` directory.

Les déploiements d'Open Test Framework (OTF) ont échoué

Si les tests OTF ne parviennent pas à terminer le déploiement, cela peut être dû aux autorisations définies pour le dossier parent de TempResourcesDirOnDevice et InstallationDirRootOnDevice. Pour définir correctement les autorisations de ce dossier, exécutez la commande suivante. Remplacez `folder-name` par le nom du dossier parent.

```
sudo chmod 755 folder-name
```

Erreurs d'analyse

Les fautes de frappe dans une configuration JSON peuvent 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 être suffisantes pour vous aider à corriger l'erreur, mais si vous ne trouvez toujours pas l'erreur, vous pouvez effectuer la validation manuellement dans votre IDE, un éditeur de texte tel qu'Atom ou Sublime, ou via un outil en ligne tel que JSONLint.

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`.

Erreur lors de la génération du rapport de qualification

IDT prend en charge les quatre dernières *major.minor* versions de la suite de qualification AWS IoT Greengrass V2 (GGV2Q) pour générer des rapports de qualification que vous pouvez soumettre AWS Partner Network pour inclure vos appareils dans le catalogue d' AWS Partner appareils. Les versions antérieures de la suite de qualification ne génèrent pas de rapports de qualification.

Si vous avez des questions concernant la politique d'assistance, contactez [AWS Support](#).

Erreur liée à un paramètre obligatoire manquant

Lorsque IDT ajoute de nouvelles fonctionnalités, il peut introduire des modifications dans les fichiers de configuration. 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 valide également vos schémas de fichiers de configuration JSON pour vérifier que vous utilisez la dernière version prise en charge.

Exception de sécurité sur macOS

Lorsque vous exécutez IDT sur un ordinateur hôte macOS, il bloque l'exécution d'IDT. Pour exécuter IDT, accordez une exception de sécurité aux exécutables faisant partie de la fonctionnalité d'exécution IDT. Lorsque le message d'avertissement s'affiche sur votre ordinateur hôte, procédez comme suit pour chacun des exécutables applicables :

Pour accorder une exception de sécurité aux exécutables IDT

1. Sur l'ordinateur macOS, dans le menu Apple, ouvrez les Préférences système.

2. Choisissez Sécurité et confidentialité, puis dans l'onglet Général, cliquez sur l'icône représentant un cadenas pour modifier les paramètres de sécurité.
3. En cas de blocage de `devicetester_mac_x86-64`, recherchez le message `"devicetester_mac_x86-64" was blocked from use because it is not from an identified developer.` et choisissez Autoriser quand même.
4. Reprenez les tests IDT jusqu'à ce que vous ayez vérifié tous les exécutables concernés.

Erreurs de connexion SSH

Lorsqu'IDT ne parvient pas à se connecter à un appareil testé, il enregistre les échecs de connexion. `/results/<execution-id>/logs/<test-case-id>.log` Les messages 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 TTY terminal Pu pour se connecter aux hôtes Linux. Cette application nécessite que vous convertissiez les fichiers de clé privée PEM standard dans un format Windows propriétaire appelé PPK. Si vous configurez SSH dans votre `device.json` fichier, utilisez des fichiers PEM. Si vous utilisez un fichier PPK, IDT ne peut pas créer de connexion SSH avec le AWS IoT Greengrass périphérique et ne peut pas exécuter de tests.

À partir de IDT v4.4.0, si vous n'avez pas activé le SFTP sur votre appareil testé, l'erreur suivante peut s'afficher dans le fichier journal.

```
SSH connection failed with EOF
```

Pour résoudre cette erreur, activez le protocole SFTP sur votre appareil.

Erreurs de qualification du gestionnaire de flux

Lorsque vous exécutez des tests de qualification du gestionnaire de flux, l'erreur suivante peut s'afficher dans le `com.aws.StreamManagerExport.log` fichier.

```
Failed to upload data to S3
```

Cette erreur peut se produire lorsque le gestionnaire de flux utilise les AWS informations d'identification du `~/root/.aws/credentials` fichier sur votre appareil au lieu d'utiliser les informations d'identification de l'environnement qu'IDT exporte vers l'appareil testé. Pour éviter ce problème, supprimez le `credentials` fichier sur votre appareil et relancez le test de qualification.

Erreurs de délai d'attente

Vous pouvez augmenter le délai d'expiration de chaque test en spécifiant un multiplicateur de délai appliqué à la valeur par défaut du délai d'expiration 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 GGV2Q_1.0.0 --pool-id DevicePool1 --timeout-multiplier 2.5
```

Pour de plus amples informations, exécutez `run-suite --help`.

Certaines erreurs de temporisation se produisent lorsque les scénarios de test IDT ne peuvent pas être terminés en raison de problèmes de configuration. Vous ne pouvez pas résoudre ces erreurs en augmentant le multiplicateur du délai d'expiration. Utilisez les journaux du test pour résoudre les problèmes de configuration sous-jacents.

- Si les journaux des composants MQTT ou Lambda Access `denied` contiennent des erreurs, il est possible que votre dossier d'installation Greengrass ne dispose pas des autorisations de fichier correctes. Exécutez la commande suivante pour chaque dossier du chemin d'installation que vous avez défini dans votre `userdata.json` fichier.

```
sudo chmod 755 folder-name
```

- Si les journaux de Greengrass indiquent que le déploiement de la CLI Greengrass n'est pas terminé, procédez comme suit :
 - Vérifiez qu'`bashil` est installé sur le périphérique testé.
 - Si votre `userdata.json` fichier inclut le paramètre `GreengrassCliVersion` de configuration, supprimez-le. Ce paramètre est obsolète dans IDT v4.1.0 et versions ultérieures. Pour de plus amples informations, veuillez consulter [Configurer userdata.json](#).
- Si le test de déploiement Lambda a échoué avec le message d'erreur « Validation de la publication Lambda : expiration du délai » et que vous recevez une erreur dans le fichier journal de test `idt-gg2-lambda-function-idt-<resource-id>.log()` `Error: Could not find or load main class com.amazonaws.greengrass.runtime.LambdaRuntime.` indiquant que vous devez procéder comme suit :
 - Vérifiez le dossier utilisé `InstallationDirRootOnDevice` dans le `userdata.json` fichier.

- Assurez-vous que les autorisations utilisateur appropriées sont configurées sur votre appareil. Pour plus de détails, consultez [Configurer les autorisations utilisateur sur votre appareil](#).

Erreurs de vérification de version

IDT émet l'erreur suivante lorsque les informations AWS d'identification de l'utilisateur IDT ne disposent pas des autorisations IAM requises.

```
Failed to check version compatibility
```

L' AWS utilisateur qui ne dispose pas des autorisations IAM requises.

Politique de support AWS IoT Device Tester pour AWS IoT Greengrass

AWS IoT Device Tester for AWS IoT Greengrass est un outil d'automatisation des tests utilisé pour valider et [qualifier](#) vos AWS IoT Greengrass appareils en vue de leur inclusion dans le [catalogue AWS Partner d'appareils](#). Nous vous recommandons d'utiliser la version la plus récente de AWS IoT Greengrass et de AWS IoT Device Tester tester ou de qualifier vos appareils.

Au moins une version de AWS IoT Device Tester est disponible pour chaque version prise en charge de AWS IoT Greengrass. Pour les versions prises en charge de AWS IoT Greengrass, voir les versions de [Greengrass noyau](#). Pour les versions prises en charge de AWS IoT Device Tester, voir [Versions prises en charge de AWS IoT Device Tester for AWS IoT Greengrass V2](#).

Vous pouvez également utiliser l'une des versions prises en charge de AWS IoT Greengrass et AWS IoT Device Tester pour tester ou qualifier vos appareils. Bien que vous puissiez continuer à utiliser des versions non prises en charge de AWS IoT Device Tester, ces versions ne reçoivent ni corrections de bogues ni mises à jour. Si vous avez des questions concernant la politique d'assistance, contactez [AWS Support](#).

Solutions IoT basées sur Greengrass

Everyware d'Eurotech GreenEdge est en version préliminaire AWS IoT Greengrass et est sujet à modification. Cette solution n'est pas prise en charge par AWS. Vous devez contacter Eurotech pour tout problème concernant cet appareil.

AWS IoT Greengrass propose des solutions proposées par des partenaires pour optimiser votre expérience d'installation de Greengrass. Voici une solution proposée en AWS partenariat avec Eurotech. Cette solution est dotée d'un environnement d'exécution AWS IoT Greengrass Core Edge et de fonctionnalités supplémentaires préinstallées.

Eurotech

AWS s'est associé à Eurotech pour proposer une solution IoT aux clients qui recherchent un appareil livré avec le logiciel AWS IoT Greengrass Core préinstallé. Everyware d'Eurotech GreenEdge est un logiciel IoT de pointe préconfiguré et préqualifié par AWS. Cette solution combine les capacités de Greengrass et de l'Eurotech Everyware Software Framework (ESF) pour offrir aux clients une connectivité étendue vers le sud via des adaptateurs de protocole tels que : Modbus, OPC-UA, etc. Client/Server, S7, TwinCAT, J1939, DNP3 Master/Outstation Avec cette solution, vous pouvez également envoyer des données AWS Cloud et vous connecter à tous les AWS services en provenance du nord (tels que AWS IoT Core, AWS IoT SiteWise, AWS IoT Analytics, Amazon S3 et Amazon Kinesis Video Streams). Combinée à Everyware Cloud, la solution de gestion des appareils d'Eurotech, cette solution introduit un nouveau service Zero-Touch Provisioning, qui simplifie l'intégration des appareils et leur déploiement en masse.

Pour plus d'informations sur Eurotech, consultez [Eurotech](#).

Résolution des problèmes AWS IoT Greengrass V2

Utilisez les informations de dépannage et les solutions de cette section pour résoudre les problèmes liés à AWS IoT Greengrass Version 2.

Rubriques

- [Afficher les journaux des logiciels AWS IoT Greengrass principaux et des composants](#)
- [AWS IoT Greengrass Principaux problèmes liés au logiciel](#)
- [AWS IoT Greengrass problèmes liés au cloud](#)
- [Principaux problèmes liés au déploiement des appareils](#)
- [Principaux problèmes liés aux composants de l'appareil](#)
- [Principaux problèmes liés aux composants de la fonction Lambda de l'appareil](#)
- [Version du composant abandonnée](#)
- [Problèmes liés à l'interface de ligne de commande Greengrass](#)
- [AWS Command Line Interface problèmes](#)
- [Codes d'erreur de déploiement détaillés](#)
- [Codes d'état détaillés des composants](#)

Afficher les journaux des logiciels AWS IoT Greengrass principaux et des composants

Le logiciel AWS IoT Greengrass Core écrit des journaux dans le système de fichiers local que vous pouvez utiliser pour consulter des informations en temps réel sur le périphérique principal. Vous pouvez également configurer les périphériques principaux pour écrire des journaux dans les CloudWatch journaux, afin de pouvoir dépanner à distance les périphériques principaux. Ces journaux peuvent vous aider à identifier les problèmes liés aux composants, aux déploiements et aux principaux appareils. Pour de plus amples informations, veuillez consulter [AWS IoT Greengrass Journaux de surveillance](#).

AWS IoT Greengrass Principaux problèmes liés au logiciel

Résoudre les problèmes liés AWS IoT Greengrass au logiciel de base.

Rubriques

- [ThrottlingException depuis l' ListDeployments API](#)
- [Impossible de configurer le périphérique principal](#)
- [Impossible de démarrer le logiciel AWS IoT Greengrass Core en tant que service système](#)
- [Impossible de configurer Nucleus en tant que service système](#)
- [Impossible de se connecter à AWS IoT Core](#)
- [Erreur de mémoire insuffisante](#)
- [Impossible d'installer Greengrass CLI](#)
- [User root is not allowed to execute](#)
- [com.aws.greengrass.lifecyclemanager.GenericExternalService: Could not determine user/group to run with](#)
- [Failed to map segment from shared object: operation not permitted](#)
- [Impossible de configurer le service Windows](#)
- [com.aws.greengrass.util.exceptions.TLSAuthException: Failed to get trust manager](#)
- [com.aws.greengrass.deployment.lotJobsHelper: No connection available during subscribing to lot Jobs descriptions topic. Will retry in sometime](#)
- [software.amazon.awssdk.services.iam.model.IamException: The security token included in the request is invalid](#)
- [software.amazon.awssdk.services.iot.model.IotException: User: <user> is not authorized to perform: iot:GetPolicy](#)
- [Error: com.aws.greengrass.shadowmanager.sync.model.FullShadowSyncRequest: Could not execute cloud shadow get request](#)
- [Operation aws.greengrass#<operation> is not supported by Greengrass](#)
- [java.io.FileNotFoundException: <stream-manager-store-root-dir>/stream_manager_metadata_store \(Permission denied\)](#)
- [com.aws.greengrass.security.provider.pkcs11.PKCS11CryptoKeyService: Private key or certificate with label <label> does not exist](#)
- [software.amazon.awssdk.services.secretsmanager.model.SecretsManagerException: User: <user> is not authorized to perform: secretsmanager:GetSecretValue on resource: <arn>](#)
- [software.amazon.awssdk.services.secretsmanager.model.SecretsManagerException: Access to KMS is not allowed](#)
- [java.lang.NoClassDefFoundError: com/aws/greengrass/security/CryptoKeySpi](#)

- [com.aws.greengrass.security.provider.pkcs11.PKCS11CryptoKeyService:CKR_OPERATION_NOT_INITIALIZED](#)
- [Greengrass core device stuck on nucleus v2.12.3](#)
- [Greengrass nucleus v2.14.0 systemd template issue](#)

ThrottlingException depuis l' ListDeployments API

ThrottlingException depuis ListDeployments l'API : cela peut s'afficher lorsque vous avez un grand nombre de déploiements en compte.

Pour résoudre ce problème, effectuez l'une des opérations suivantes :

- Si vous utilisez le SDK, veuillez spécifier le MaxResult paramètre. Par exemple, pour [JavasDK](#) avec une petite valeur (par exemple 5).
- Vous pouvez utiliser les [Quotas de AWS Service](#) pour demander une augmentation de la limite de débit de l'DescribeJobAPI. Vous pouvez accéder à la console des quotas de service, sélectionner les quotas de AWS IoT et le nom de la limite est limite d'DescribeJob accélération. Vous pouvez l'augmenter de 10 à 50.

Impossible de configurer le périphérique principal

Si le programme d'installation du logiciel AWS IoT Greengrass Core échoue et que vous ne parvenez pas à configurer un appareil principal, vous devrez peut-être désinstaller le logiciel et réessayer. Pour de plus amples informations, veuillez consulter [Désinstallez le logiciel AWS IoT Greengrass Core](#).

Impossible de démarrer le logiciel AWS IoT Greengrass Core en tant que service système

Si le logiciel AWS IoT Greengrass Core ne démarre pas, [consultez les journaux de service du système](#) pour identifier le problème. Un problème courant est que Java n'est pas disponible sur la variable d'environnement PATH (Linux) ou la variable système PATH (Windows).

Impossible de configurer Nucleus en tant que service système

Cette erreur peut s'afficher lorsque le programme d'installation du logiciel AWS IoT Greengrass Core ne parvient pas à être configuré AWS IoT Greengrass en tant que service système. Sur les appareils

Linux, cette erreur se produit généralement si le périphérique principal ne possède pas le [système d'initialisation systemd](#). Le programme d'installation peut configurer correctement le logiciel AWS IoT Greengrass Core même s'il ne parvient pas à configurer le service système.

Effectuez l'une des actions suivantes :

- Configurez et exécutez le logiciel AWS IoT Greengrass Core en tant que service système. Vous devez configurer le logiciel en tant que service système pour utiliser toutes les fonctionnalités de AWS IoT Greengrass. Vous pouvez installer [systemd](#) ou utiliser un autre système d'initialisation. Pour de plus amples informations, veuillez consulter [Configurer le noyau Greengrass en tant que service système](#).
- Exécutez le logiciel AWS IoT Greengrass Core sans service système. Vous pouvez exécuter le logiciel à l'aide d'un script de chargement que le programme d'installation configure dans le dossier racine de Greengrass. Pour de plus amples informations, veuillez consulter [Exécutez le logiciel AWS IoT Greengrass Core sans service système](#).

Impossible de se connecter à AWS IoT Core

Cette erreur peut s'afficher lorsque le logiciel AWS IoT Greengrass Core ne parvient pas à se connecter AWS IoT Core pour récupérer des tâches de déploiement, par exemple. Procédez comme suit :

- Vérifiez que votre appareil principal peut se connecter à Internet et AWS IoT Core. Pour plus d'informations sur le AWS IoT Core point de terminaison auquel votre appareil se connecte, consultez [Configuration du logiciel AWS IoT Greengrass de base](#).
- Vérifiez que l'appareil AWS IoT principal de votre appareil utilise un certificat qui autorise les `iot:Subscribe` autorisations `iot:Connect` `iot:Publish` `iot:Receive`, et.
- Si votre appareil principal utilise un [proxy réseau](#), vérifiez qu'il a un [rôle d'appareil](#) et que son rôle autorise les `iot:Subscribe` autorisations `iot:Connect` `iot:Publish` `iot:Receive`, et.

Erreur de mémoire insuffisante

Cette erreur se produit généralement si votre appareil ne dispose pas de suffisamment de mémoire pour allouer un objet dans le tas Java. Sur les appareils dont la mémoire est limitée, vous devrez peut-être spécifier une taille de segment maximale pour contrôler l'allocation de mémoire. Pour de plus amples informations, veuillez consulter [Contrôlez l'allocation de mémoire grâce aux options JVM](#).

Impossible d'installer Greengrass CLI

Le message de console suivant peut s'afficher lorsque vous utilisez l'`--deploy-dev-tools` argument dans votre commande d'installation de AWS IoT Greengrass Core.

```
Thing group exists, it could have existing deployment and devices, hence NOT creating
deployment for Greengrass first party dev tools, please manually create a deployment
if you wish to
```

Cela se produit lorsque le composant Greengrass CLI n'est pas installé parce que votre périphérique principal est membre d'un groupe d'objets déjà déployé. Si ce message s'affiche, vous pouvez déployer manuellement le composant Greengrass CLI (`aws.greengrass.Cli`) sur le périphérique pour installer la Greengrass CLI. Pour de plus amples informations, veuillez consulter [Installation de la CLI Greengrass](#).

User root is not allowed to execute

Cette erreur peut s'afficher lorsque l'utilisateur qui exécute le logiciel AWS IoT Greengrass Core (généralement `root`) n'est pas autorisé à s'exécuter `sudo` avec un utilisateur ou un groupe. Pour l'utilisateur `ggc_user` du système par défaut, cette erreur se présente comme suit :

```
Sorry, user root is not allowed to execute <command> as ggc_user:ggc_group.
```

Vérifiez que votre `/etc/sudoers` fichier autorise l'utilisateur à s'exécuter `sudo` en tant qu'autre groupe. L'autorisation accordée à l'utilisateur `/etc/sudoers` doit ressembler à l'exemple suivant.

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

com.aws.greengrass.lifecyclemanager.GenericExternalService: Could not determine user/group to run with

Cette erreur peut s'afficher lorsque le périphérique principal essaie d'exécuter un composant et que le noyau Greengrass ne spécifie pas d'utilisateur système par défaut à utiliser pour exécuter les composants.

Pour résoudre ce problème, configurez le noyau Greengrass pour spécifier l'utilisateur du système par défaut qui exécute les composants. Pour plus d'informations, consultez [Configurer l'utilisateur qui exécute les composants](#) et [Configuration de l'utilisateur du composant par défaut](#).

Failed to map segment from shared object: operation not permitted

Cette erreur peut s'afficher lorsque le logiciel AWS IoT Greengrass Core ne démarre pas parce que le /tmp dossier est monté avec des noexec autorisations. La [bibliothèque AWS Common Runtime \(CRT\)](#) utilise le /tmp dossier par défaut.

Effectuez l'une des actions suivantes :

- Exécutez la commande suivante pour remonter le /tmp dossier avec exec les autorisations et réessayez.

```
sudo mount -o remount,exec /tmp
```

- Si vous exécutez Greengrass nucleus v2.5.0 ou version ultérieure, vous pouvez définir une option JVM pour modifier le dossier utilisé par la bibliothèque CRT. AWS Vous pouvez spécifier le `jvmOptions` paramètre dans la configuration du composant Greengrass Nucleus lors d'un déploiement ou lors de l'installation du logiciel AWS IoT Greengrass Core. Remplacez `/path/to/use` par le chemin d'accès à un dossier que la bibliothèque AWS CRT peut utiliser.

```
{  
  "jvmOptions": "-Daws.crt.lib.dir=\"/path/to/use\""  
}
```

Impossible de configurer le service Windows

Cette erreur peut s'afficher si vous installez le logiciel AWS IoT Greengrass Core sur un appareil Microsoft Windows 2016. Le logiciel AWS IoT Greengrass Core n'est pas pris en charge sous Windows 2016. Pour obtenir la liste des systèmes d'exploitation pris en charge, voir [Plateformes prises en charge](#).

Si vous devez utiliser Windows 2016, vous pouvez effectuer les opérations suivantes :

1. Décompressez l'archive d'installation AWS IoT Greengrass Core téléchargée
2. Dans le Greengrass répertoire, ouvrez le `bin/greengrass.xml.template` fichier.
3. Ajoutez le `<autoRefresh>` tag à la fin du fichier juste avant le `</service>` tag.

```
</log>  
<autoRefresh>false</autoRefresh>
```



```
</service>
```

com.aws.greengrass.util.exceptions.TLSAuthException: Failed to get trust manager

Cette erreur peut s'afficher lorsque vous installez le logiciel AWS IoT Greengrass Core sans fichier d'autorité de certification (CA) racine.

```
2022-06-05T10:00:39.556Z [INFO] (main) com.aws.greengrass.lifecyclemanager.Kernel:
  service-loaded. {serviceName=DeploymentService}
2022-06-05T10:00:39.943Z [WARN] (main)
  com.aws.greengrass.componentmanager.ClientConfigurationUtils: configure-greengrass-
  mutual-auth. Error during configure greengrass client mutual auth. {}
com.aws.greengrass.util.exceptions.TLSAuthException: Failed to get trust manager
```

Vérifiez que vous spécifiez un fichier CA racine valide avec le `rootCaPath` paramètre dans le fichier de configuration que vous fournissez au programme d'installation. Pour de plus amples informations, veuillez consulter [Installation du logiciel AWS IoT Greengrass de base](#).

com.aws.greengrass.deployment.lotJobsHelper: No connection available during subscribing to lot Jobs descriptions topic. Will retry in sometime

Ce message d'avertissement peut s'afficher lorsque l'appareil principal ne parvient pas à se connecter pour s'abonner AWS IoT Core aux notifications de tâches de déploiement. Procédez comme suit :

- Vérifiez que l'appareil principal est connecté à Internet et peut atteindre le point de terminaison de AWS IoT données que vous avez configuré. Pour plus d'informations sur les terminaux utilisés par les appareils principaux, consultez [Autoriser le trafic des appareils via un proxy ou un pare-feu](#).
- Consultez les journaux de Greengrass pour détecter d'autres erreurs révélant d'autres causes profondes.

software.amazon.awssdk.services.iam.model.IamException: The security token included in the request is invalid

Cette erreur peut s'afficher lorsque vous [installez le logiciel AWS IoT Greengrass Core avec un provisionnement automatique](#) et que le programme d'installation utilise un jeton de AWS session non valide. Procédez comme suit :

- Si vous utilisez des informations d'identification de sécurité temporaires, vérifiez que le jeton de session est correct et que vous copiez et collez le jeton de session complet.
- Si vous utilisez des informations d'identification de sécurité à long terme, vérifiez que l'appareil ne possède pas de jeton de session datant d'une époque où vous utilisiez auparavant des informations d'identification temporaires. Procédez comme suit :

1. Exécutez la commande suivante pour désactiver la variable d'environnement du jeton de session.

Linux or Unix

```
unset AWS_SESSION_TOKEN
```

Windows Command Prompt (CMD)

```
set AWS_SESSION_TOKEN=
```

PowerShell

```
Remove-Item Env:\AWS_SESSION_TOKEN
```

2. Vérifiez si le fichier AWS d'informations d'identification contient un jeton de session, `aws_session_token`. `~/.aws/credentials` Si tel est le cas, supprimez cette ligne du fichier.

```
aws_session_token = AQoEXAMPLEH4aoAH0gNCAPyJxz4BlCFFxWNE1OPTgk5TthT
+FvqwqNkRc0IfRrh3c/LTo6UDdyJw00vEVPvLXCrrrUtdnniCEXAMPLE/
IvU1dYUg2RVAJBanLiHb4IgRmpRV3zrkuWJ0gQs8IZZaIv2BXIa2R40lgk
```

Vous pouvez également installer le logiciel AWS IoT Greengrass Core sans fournir AWS d'informations d'identification. Pour plus d'informations, consultez [Installation AWS IoT Greengrass](#)

[du logiciel Core avec provisionnement manuel des ressources](#) ou [Installation AWS IoT Greengrass du logiciel de base avec provisionnement du AWS IoT parc](#).

software.amazon.awssdk.services.iot.model.IotException: User: <user> is not authorized to perform: iot:GetPolicy

Cette erreur peut s'afficher lorsque vous [installez le logiciel AWS IoT Greengrass Core avec un provisionnement automatique](#) et que le programme d'installation utilise des AWS informations d'identification ne disposant pas des autorisations requises. Pour plus d'informations sur les autorisations requises, consultez [Politique IAM minimale permettant au programme d'installation de provisionner les ressources](#).

Vérifiez les autorisations relatives à l'identité IAM des informations d'identification et accordez à l'identité IAM toutes les autorisations requises manquantes.

Error:

com.aws.greengrass.shadowmanager.sync.model.FullShadowSyncRequest:
Could not execute cloud shadow get request

Cette erreur peut s'afficher lorsque vous utilisez le [composant Shadow Manager](#) pour [synchroniser les ombres de l'appareil avec AWS IoT Core](#). Le code d'état HTTP 403 indique que cette erreur s'est produite parce que la AWS IoT politique du périphérique principal n'accorde pas l'autorisation d'appeler `GetThingShadow`.

```
com.aws.greengrass.shadowmanager.sync.model.FullShadowSyncRequest: Could not execute
cloud shadow get request. {thing name=MyGreengrassCore, shadow name=MyShadow}
2021-07-14T21:09:02.456Z [ERROR] (pool-2-thread-109)
com.aws.greengrass.shadowmanager.sync.SyncHandler: sync. Skipping sync request. {thing
name=MyGreengrassCore, shadow name=MyShadow}
com.aws.greengrass.shadowmanager.exception.SkipSyncRequestException:
software.amazon.awssdk.services.iotdataplane.model.IotDataPlaneException:
null (Service: IotDataPlane, Status Code: 403, Request ID:
f6e713ba-1b01-414c-7b78-5beb3f3ad8f6, Extended Request ID: null)
```

Pour synchroniser les ombres locales avec AWS IoT Core, la AWS IoT politique de l'appareil principal doit accorder les autorisations suivantes :

- `iot:GetThingShadow`

- `iot:UpdateThingShadow`
- `iot:DeleteThingShadow`

Vérifiez la AWS IoT politique de l'appareil principal et ajoutez les autorisations requises manquantes. Pour plus d'informations, consultez les ressources suivantes :

- [AWS IoT Core actions politiques décrites](#) dans le guide du AWS IoT développeur
- [Mettre à jour la AWS IoT politique d'un appareil principal](#)

Operation `aws.greengrass#<operation>` is not supported by Greengrass

Cette erreur peut s'afficher lorsque vous utilisez une [opération de communication interprocessus \(IPC\)](#) dans un composant Greengrass personnalisé et que le composant AWS fourni n'est pas installé sur le périphérique principal.

Pour résoudre ce problème, ajoutez le composant requis en tant que [dépendance dans votre recette de composant](#), afin que le logiciel AWS IoT Greengrass Core installe le composant requis lorsque vous déployez votre composant.

- [Récupérez les valeurs secrètes](#) — `aws.greengrass.SecretManager`
- [Interagissez avec les ombres locales](#) — `aws.greengrass.ShadowManager`
- [Gestion des déploiements et des composants locaux](#) (version `aws.greengrass.Cli` 2.6.0 ou ultérieure)
- [Authentifier et autoriser les appareils clients](#) — `aws.greengrass.clientdevices.Auth` v2.2.0 ou version ultérieure

`java.io.FileNotFoundException: <stream-manager-store-root-dir>/
stream_manager_metadata_store (Permission denied)`

Cette erreur peut s'afficher dans le fichier journal du gestionnaire de flux (`aws.greengrass.StreamManager.log`) lorsque vous configurez le [gestionnaire de flux](#) pour qu'il utilise un dossier racine qui n'existe pas ou qui ne dispose pas des autorisations appropriées. Pour plus d'informations sur la configuration de ce dossier, consultez la section [Configuration du gestionnaire de flux](#).

com.aws.greengrass.security.provider.pkcs11.PKCS11CryptoKeyService: Private key or certificate with label <label> does not exist

Cette erreur se produit lorsque le [composant fournisseur PKCS #11](#) ne parvient pas à trouver ou à charger la clé privée ou le certificat que vous spécifiez lorsque vous configurez le logiciel AWS IoT Greengrass Core pour utiliser un [module de sécurité matériel \(HSM\)](#). Procédez comme suit :

- Vérifiez que la clé privée et le certificat sont stockés dans le HSM à l'aide de l'emplacement, du code PIN utilisateur et de l'étiquette d'objet que vous configurez pour utiliser dans le logiciel AWS IoT Greengrass Core.
- Vérifiez que la clé privée et le certificat utilisent la même étiquette d'objet dans le HSM.
- Si votre HSM prend en charge l'objet IDs, vérifiez que la clé privée et le certificat utilisent le même identifiant d'objet dans le HSM.

Consultez la documentation de votre HSM pour savoir comment demander des informations sur les jetons de sécurité contenus dans le HSM. Si vous devez modifier l'emplacement, l'étiquette ou l'ID d'objet d'un jeton de sécurité, consultez la documentation de votre HSM pour savoir comment procéder.

software.amazon.awssdk.services.secretsmanager.model.SecretsManagerException: User: <user> is not authorized to perform: secretsmanager:GetSecretValue on resource: <arn>

Cette erreur peut se produire lorsque vous utilisez le [composant du gestionnaire de secrets](#) pour déployer un AWS Secrets Manager secret. Si le [rôle IAM d'échange de jetons](#) du périphérique principal n'autorise pas l'obtention du secret, le déploiement échoue et les journaux de Greengrass incluent cette erreur.

Pour autoriser un appareil principal à télécharger un secret

1. Ajoutez l'`secretsmanager:GetSecretValue` autorisation au rôle d'échange de jetons de l'appareil principal. L'exemple de déclaration de politique suivant accorde l'autorisation d'obtenir la valeur d'un secret.

```
{  
    "Effect": "Allow",  
    "Action": [
```

```
        "secretsmanager:GetSecretValue"
    ],
    "Resource": [
        "arn:aws:secretsmanager:us-west-2:123456789012:secret:MyGreengrassSecret-
        abcdef"
    ]
}
```

Pour de plus amples informations, veuillez consulter [Autoriser les appareils principaux à interagir avec les AWS services](#).

2. Réappliquez le déploiement au périphérique principal. Effectuez l'une des actions suivantes :
 - Réviser le déploiement sans aucune modification. Le périphérique principal essaie de télécharger à nouveau le secret lorsqu'il reçoit le déploiement révisé. Pour de plus amples informations, veuillez consulter [Réviser les déploiements](#).
 - Redémarrez le logiciel AWS IoT Greengrass Core pour réessayer le déploiement. Pour de plus amples informations, consultez [Exécutez le logiciel AWS IoT Greengrass Core](#).

Le déploiement réussit si le gestionnaire de secrets télécharge le secret avec succès.

software.amazon.awssdk.services.secretsmanager.model.SecretsManagerException: Access to KMS is not allowed

Cette erreur peut se produire lorsque vous utilisez le [composant du gestionnaire de secrets](#) pour déployer un AWS Secrets Manager secret chiffré par une AWS Key Management Service clé. Si le [rôle IAM d'échange de jetons](#) du périphérique principal n'autorise pas le déchiffrement du secret, le déploiement échoue et les journaux de Greengrass incluent cette erreur.

Pour résoudre le problème, ajoutez l'`kms:Decrypt` autorisation au rôle d'échange de jetons de l'appareil principal. Pour plus d'informations, consultez les ressources suivantes :

- [Chiffrement et déchiffrement secrets](#) dans le guide de l'AWS Secrets Manager utilisateur
- [Autoriser les appareils principaux à interagir avec les AWS services](#)

java.lang.NoClassDefFoundError: com/aws/greengrass/security/CryptoKeySpi

Cette erreur peut s'afficher lorsque vous essayez d'installer le logiciel AWS IoT Greengrass Core avec [sécurité matérielle](#) et que vous utilisez une version antérieure de Greengrass Nucleus qui ne prend pas en charge l'intégration de la sécurité matérielle. Pour utiliser l'intégration de la sécurité matérielle, vous devez utiliser Greengrass nucleus v2.5.3 ou version ultérieure.

com.aws.greengrass.security.provider.pkcs11.PKCS11CryptoKeyService: CKR_OPERATION_NOT_INITIALIZED

Cette erreur peut s'afficher lorsque vous utilisez la TPM2 bibliothèque lors de l'exécution de AWS IoT Greengrass Core en tant que service système.

Cette erreur indique que vous devez ajouter une variable d'environnement indiquant l'emplacement du magasin PKCS #11 dans le fichier de service AWS IoT Greengrass Core systemd.

Pour plus d'informations, consultez la section Exigences de la documentation du [Fournisseur PKCS #11](#) composant.

Greengrass core device stuck on nucleus v2.12.3

Si votre appareil principal Greengrass ne veut pas réviser votre déploiement à partir de la version 2.12.3 de Nucleus, vous devrez peut-être télécharger et remplacer le fichier Greengrass.jar par Greengrass nucleus version 2.12.2. Procédez comme suit :

1. Sur votre appareil Greengrass Core, exécutez la commande suivante pour arrêter le logiciel Greengrass Core.

Linux or Unix

```
sudo systemctl stop greengrass
```

Windows Command Prompt (CMD)

```
sc stop "greengrass"
```

PowerShell

```
Stop-Service -Name "greengrass"
```

2. Sur votre appareil principal, téléchargez le AWS IoT Greengrass logiciel dans un fichier nommé `greengrass-2.12.2.zip`.

Linux or Unix

```
curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-2.12.2.zip >  
greengrass-2.12.2.zip
```

Windows Command Prompt (CMD)

```
curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-2.12.2.zip >  
greengrass-2.12.2.zip
```

PowerShell

```
iwr -Uri https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-2.12.2.zip -  
OutFile greengrass-2.12.2.zip
```

3. Décompressez le logiciel AWS IoT Greengrass Core dans un dossier de votre appareil.
GreengrassInstaller Remplacez-le par le dossier que vous souhaitez utiliser.

Linux or Unix

```
unzip greengrass-2.12.2.zip -d GreengrassInstaller && rm greengrass-2.12.2.zip
```

Windows Command Prompt (CMD)

```
mkdir GreengrassInstaller && tar -xf greengrass-2.12.2.zip -  
C GreengrassInstaller && del greengrass-2.12.2.zip
```

PowerShell

```
Expand-Archive -Path greengrass-2.12.2.zip -DestinationPath .\  
\ GreengrassInstaller  
rm greengrass-2.12.2.zip
```


4. Exécutez la commande suivante pour remplacer le fichier JAR Greengrass de la version 2.12.3 du noyau par le fichier JAR Greengrass de la version 2.12.2 du noyau.

Linux or Unix

```
sudo cp ./GreengrassInstaller/lib/Greengrass.jar /greengrass/v2/packages/artifacts-unarchived/aws.greengrass.Nucleus/2.12.3/aws.greengrass.nucleus/lib
```

Windows Command Prompt (CMD)

```
robocopy ./GreengrassInstaller/lib/Greengrass.jar /greengrass/v2/packages/artifacts-unarchived/aws.greengrass.Nucleus/2.12.3/aws.greengrass.nucleus/lib /E
```

PowerShell

```
cp -Path ./GreengrassInstaller/lib/Greengrass.jar -Destination /greengrass/v2/packages/artifacts-unarchived/aws.greengrass.Nucleus/2.12.3/aws.greengrass.nucleus/lib
```

5. Exécutez la commande suivante pour démarrer le logiciel Greengrass Core.

Linux or Unix

```
sudo systemctl start greengrass
```

Windows Command Prompt (CMD)

```
sc start "greengrass"
```

PowerShell

```
Start-Service -Name "greengrass"
```

Greengrass nucleus v2.14.0 systemd template issue

Vous pouvez rencontrer ce problème si vous avez installé Greengrass nucleus v2.14.0 sur un appareil Linux avec son modèle de service systemd par défaut. Procédez comme suit :

1. Sur votre périphérique principal Greengrass, exécutez la commande suivante pour rétablir le fichier de service systemd tel qu'il est dans Nucleus v2.13.0 et versions antérieures.

Linux or Unix

```
sudo sed -i 's|ExecStart=/bin/sh -c "\(.*\)" >> ./logs/loader.log 2>&1|  
ExecStart=/bin/sh \1|' /etc/systemd/system/greengrass.service
```

2. Appliquez les modifications.

Linux or Unix

```
sudo systemctl daemon-reload  
sudo systemctl restart greengrass
```

AWS IoT Greengrass problèmes liés au cloud

Utilisez les informations suivantes pour résoudre les problèmes liés à la AWS IoT Greengrass console et à l'API. Chaque entrée correspond à un message d'erreur qui peut s'afficher lorsque vous effectuez une action.

An error occurred (AccessDeniedException) when calling the CreateComponentVersion operation: User: arn:aws:iam::123456789012:user/<username> is not authorized to perform: null

Cette erreur peut s'afficher lorsque vous créez une version de composant à partir de la AWS IoT Greengrass console ou lors de l'[CreateComponentVersion](#) opération.

Cette erreur indique que votre recette n'est pas un JSON ou YAML valide. Vérifiez la syntaxe de votre recette, corrigez les éventuels problèmes de syntaxe et réessayez. Vous pouvez utiliser un vérificateur de syntaxe JSON ou YAML en ligne pour identifier les problèmes de syntaxe dans votre recette.

Invalid Input: Encountered following errors in Artifacts: {<s3ArtifactUri> = Specified artifact resource cannot be accessed}

Cette erreur peut s'afficher lorsque vous créez une version de composant à partir de la AWS IoT Greengrass console ou lors de l'[CreateComponentVersion](#) opération. Cette erreur indique qu'un artefact S3 de la recette du composant n'est pas valide.

Procédez comme suit :

- Vérifiez que le compartiment S3 se trouve dans le même emplacement que celui dans Région AWS le quel vous créez le composant. AWS IoT Greengrass ne prend pas en charge les demandes interrégionales pour les artefacts de composants.
- Vérifiez que l'URI de l'artefact est une URL d'objet S3 valide et vérifiez que l'artefact existe à cette URL d'objet S3.
- Vérifiez que vous êtes Compte AWS autorisé à accéder à l'artefact via l'URL de son objet S3.

INACTIVE deployment status

Vous pouvez obtenir un statut de INACTIVE déploiement lorsque vous appelez l'[ListDeployments](#) API sans les AWS IoT politiques dépendantes requises. Vous devez disposer des autorisations nécessaires pour obtenir un statut de déploiement précis. Vous pouvez trouver les actions dépendantes en consultant les [actions définies par AWS IoT Greengrass V2 et en](#) suivant les autorisations nécessaires pour `ListDeployments`. Sans les AWS IoT autorisations dépendantes requises, vous verrez toujours l'état du déploiement, mais vous pourriez voir un état de déploiement inexact de `INACTIVE`.

Principaux problèmes liés au déploiement des appareils

Résolvez les problèmes de déploiement sur les appareils principaux de Greengrass. Chaque entrée correspond à un message de journal que vous pourriez voir sur votre appareil principal.

Rubriques

- [Error: com.aws.greengrass.componentmanager.exceptions.PackageDownloadException: Failed to download artifact](#)
- [Error: com.aws.greengrass.componentmanager.exceptions.ArtifactChecksumMismatchException: Integrity check for downloaded artifact failed. Probably due to file corruption.](#)

- Error:
`com.aws.greengrass.componentmanager.exceptions.NoAvailableComponentVersionException: Failed to negotiate component <name> version with cloud and no local applicable version satisfying requirement <requirements>`
- `software.amazon.awssdk.services.greengrassv2data.model.ResourceNotFoundException: The latest version of Component <componentName> doesn't claim platform <coreDevicePlatform> compatibility`
- `com.aws.greengrass.componentmanager.exceptions.PackagingException: The deployment attempts to update the nucleus from aws.greengrass.Nucleus-<version> to aws.greengrass.Nucleus-<version> but no component of type nucleus was included as target component`
- Error: `com.aws.greengrass.deployment.exceptions.DeploymentException: Unable to process deployment. Greengrass launch directory is not set up or Greengrass is not set up as a system service`
- Info:
`com.aws.greengrass.deployment.exceptions.RetryableDeploymentDocumentDownloadException: Greengrass Cloud Service returned an error when getting full deployment configuration`
- Warn: `com.aws.greengrass.deployment.DeploymentService: Failed to get thing group hierarchy`
- Info: `com.aws.greengrass.deployment.DeploymentDocumentDownloader: Calling Greengrass cloud to get full deployment configuration`
- Caused by:
`software.amazon.awssdk.services.greengrassv2data.model.GreengrassV2DataException: null (Service: GreengrassV2Data, Status Code: 403, Request ID: <some_request_id>, Extended Request ID: null)`

Error:

`com.aws.greengrass.componentmanager.exceptions.PackageDownloadException`
Failed to download artifact

Cette erreur peut s'afficher lorsque le logiciel AWS IoT Greengrass Core ne parvient pas à télécharger un artefact de composant lorsque le périphérique principal effectue un déploiement. Le déploiement échoue à cause de cette erreur.

Lorsque vous recevez cette erreur, le journal inclut également une trace de pile que vous pouvez utiliser pour identifier le problème spécifique. Chacune des entrées suivantes correspond à un

message que vous pourriez voir dans la pile de traces du message Failed to download artifact d'erreur.

Rubriques

- [software.amazon.awssdk.services.s3.model.S3Exception: null \(Service: S3, Status Code: 403, Request ID: null, ...\)](#)
- [software.amazon.awssdk.services.s3.model.S3Exception: Access Denied \(Service: S3, Status Code: 403, Request ID: <requestID>\)](#)

software.amazon.awssdk.services.s3.model.S3Exception: null (Service: S3, Status Code: 403, Request ID: null, ...)

L'[PackageDownloadException erreur](#) peut inclure cette trace de pile dans les cas suivants :

- L'artefact du composant n'est pas disponible à l'URL de l'objet S3 que vous spécifiez dans la recette du composant. Vérifiez que vous avez chargé l'artefact dans le compartiment S3 et que l'URI de l'artefact correspond à l'URL de l'objet S3 de l'artefact dans le compartiment.
- Le [rôle d'échange de jetons](#) du périphérique principal ne permet pas au logiciel AWS IoT Greengrass Core de télécharger l'artefact du composant à partir de l'URL de l'objet S3 que vous spécifiez dans la recette du composant. Vérifiez que le rôle d'échange de jetons autorise `s3:GetObject` l'URL de l'objet S3 où l'artefact est disponible.

software.amazon.awssdk.services.s3.model.S3Exception: Access Denied (Service: S3, Status Code: 403, Request ID: <requestID>)

L'[PackageDownloadException erreur](#) peut inclure cette trace de pile lorsque le périphérique principal n'est pas autorisé à appeler `s3:GetBucketLocation`. Le message d'erreur inclut également le message suivant.

```
reason: Failed to determine S3 bucket location
```

Vérifiez que le [rôle d'échange de jetons](#) du périphérique principal autorise `s3:GetBucketLocation` le compartiment S3 dans lequel l'artefact est disponible.

Error:

`com.aws.greengrass.componentmanager.exceptions.ArtifactChecksumMismatchException`
Integrity check for downloaded artifact failed. Probably due to file corruption.

Cette erreur peut s'afficher lorsque le logiciel AWS IoT Greengrass Core ne parvient pas à télécharger un artefact de composant lorsque le périphérique principal effectue un déploiement. Le déploiement échoue car la somme de contrôle du fichier d'artefact téléchargé ne correspond pas à la somme de contrôle AWS IoT Greengrass calculée lors de la création du composant.

Procédez comme suit :

- Vérifiez si le fichier d'artefact a changé dans le compartiment S3 où vous l'hébergez. Si le fichier a changé depuis que vous avez créé le composant, restaurez la version précédente attendue par le périphérique principal. Si vous ne parvenez pas à restaurer la version précédente du fichier ou si vous souhaitez utiliser la nouvelle version du fichier, créez une nouvelle version du composant avec le fichier d'artefact.
- Vérifiez la connexion Internet de votre appareil principal. Cette erreur peut se produire si le fichier d'artefact est endommagé lors du téléchargement. Créez un nouveau déploiement et réessayez.

Error:

`com.aws.greengrass.componentmanager.exceptions.NoAvailableComponentVersionException`
Failed to negotiate component <name> version with cloud and no local applicable version satisfying requirement <requirements>

Cette erreur peut s'afficher lorsqu'un périphérique principal ne trouve pas de version de composant répondant aux exigences des déploiements pour ce périphérique principal. Le périphérique principal vérifie la présence du composant dans le AWS IoT Greengrass service et sur le périphérique local. Le message d'erreur inclut la cible de chaque déploiement et les exigences de version de ce déploiement pour le composant. La cible de déploiement peut être un objet, un groupe d'objets ou `LOCAL_DEPLOYMENT` un objet représentant le déploiement local sur le périphérique principal.

Ce problème peut se produire dans les cas suivants :

- Le périphérique principal est la cible de plusieurs déploiements dont les exigences en matière de version des composants sont contradictoires. Par exemple, le périphérique principal peut être la cible de plusieurs déploiements incluant un `com.example.HelloWorld` composant, l'un

nécessitant la version 1.0.0 et l'autre la version 1.0.1. Comme il est impossible de disposer d'un composant répondant aux deux exigences, le déploiement échoue.

- La version du composant n'existe pas dans le AWS IoT Greengrass service ou sur l'appareil local. Le composant a peut-être été supprimé, par exemple.
- Il existe des versions de composants qui répondent aux exigences de version, mais aucune n'est compatible avec la plate-forme de l'appareil principal.
- La AWS IoT politique de l'appareil principal n'accorde pas l'`greengrass:ResolveComponentCandidates` autorisation. Recherchez Status Code : 403 dans le journal des erreurs pour identifier ce problème. Pour résoudre ce problème, ajoutez l'`greengrass:ResolveComponentCandidates` à la AWS IoT politique de l'appareil principal. Pour de plus amples informations, veuillez consulter [AWS IoT Politique minimale pour les appareils AWS IoT Greengrass V2 principaux](#).

Pour résoudre ce problème, modifiez les déploiements afin d'inclure les versions de composants compatibles ou supprimez les versions incompatibles. Pour plus d'informations sur la révision des déploiements dans le cloud, consultez [Réviser les déploiements](#). Pour plus d'informations sur la façon de réviser les déploiements locaux, consultez la commande [AWS IoT Greengrass CLI deployment create](#).

```
software.amazon.awssdk.services.greengrassv2data.model.ResourceNotFoundException: The latest version of Component <componentName> doesn't claim platform <coreDevicePlatform> compatibility
```

Cette erreur peut s'afficher lorsque vous déployez un composant sur un périphérique principal et que le composant ne répertorie aucune plate-forme compatible avec la plate-forme de l'appareil principal. Effectuez l'une des actions suivantes :

- S'il s'agit d'un composant Greengrass personnalisé, vous pouvez le mettre à jour pour qu'il soit compatible avec le périphérique principal. Ajoutez un nouveau manifeste correspondant à la plate-forme de l'appareil principal ou mettez à jour un manifeste existant pour qu'il corresponde à la plate-forme de l'appareil principal. Pour de plus amples informations, veuillez consulter [AWS IoT Greengrass référence de recette de composant](#).
- Si le composant est fourni par AWS, vérifiez si une autre version du composant est compatible avec le périphérique principal. Si aucune version n'est compatible, contactez-nous en [AWS re:Post](#) utilisant le [AWS IoT Greengrass tag](#), ou contactez [Support](#).

`com.aws.greengrass.componentmanager.exceptions.PackagingException:`
The deployment attempts to update the nucleus from
`aws.greengrass.Nucleus-<version>` to `aws.greengrass.Nucleus-<version>`
but no component of type nucleus was included as target component

Cette erreur peut s'afficher lorsque vous déployez un composant qui dépend du [noyau Greengrass](#) et que le périphérique principal exécute une version du noyau de Greengrass antérieure à la dernière version mineure disponible. Cette erreur se produit car le logiciel AWS IoT Greengrass Core essaie de mettre à jour automatiquement les composants vers la dernière version compatible. Cependant, le logiciel AWS IoT Greengrass Core empêche la mise à jour du noyau Greengrass vers une nouvelle version mineure, car plusieurs composants AWS fournis dépendent de versions mineures spécifiques du noyau Greengrass. Pour de plus amples informations, veuillez consulter [Comportement de mise à jour du noyau Greengrass](#).

Vous devez [revoir le déploiement](#) pour spécifier la version du noyau Greengrass que vous souhaitez utiliser. Effectuez l'une des actions suivantes :

- Réviser le déploiement pour spécifier la version du noyau Greengrass que le périphérique principal exécute actuellement.
- Réviser le déploiement pour spécifier une version mineure ultérieure du noyau Greengrass.
Si vous choisissez cette option, vous devez également mettre à jour les versions de tous les composants AWS fournis qui dépendent de versions mineures spécifiques du noyau Greengrass. Pour de plus amples informations, veuillez consulter [AWS-composants fournis](#).

`Error: com.aws.greengrass.deployment.exceptions.DeploymentException:`
Unable to process deployment. Greengrass launch directory is not set up or
Greengrass is not set up as a system service

Cette erreur peut s'afficher lorsque vous déplacez un appareil Greengrass d'un groupe d'objets à un autre, puis que vous revenez au groupe d'origine avec des déploiements nécessitant le redémarrage de Greengrass.

Pour résoudre ce problème, recréez le répertoire de lancement de l'appareil. Nous vous recommandons également vivement de passer à la version 2.9.6 ou ultérieure du noyau Greengrass.

Voici un script Linux permettant de recréer le répertoire de lancement. Enregistrez le script dans un fichier appelé `fix_directory.sh`.

```
#!/bin/bash

set -e

GG_ROOT=$1
GG_VERSION=$2

CURRENT="$GG_ROOT/alts/current"

if [ ! -L "$CURRENT" ]; then
    mkdir -p $GG_ROOT/alts/directory_fix
    echo "Relinking $GG_ROOT/alts/directory_fix to $CURRENT"
    ln -sf $GG_ROOT/alts/directory_fix $CURRENT
fi

TARGET=$(readlink $CURRENT)

if [[ ! -d "$TARGET" ]]; then
    echo "Creating directory: $TARGET"
    mkdir -p "$TARGET"
fi

DISTRO_LINK="$TARGET/distro"
DISTRO="$GG_ROOT/packages/artifacts-unarchived/aws.greengrass.Nucleus/$GG_VERSION/
aws.greengrass.nucleus/"
echo "Relinking Nucleus artifacts to $DISTRO_LINK"
ln -sf $DISTRO $DISTRO_LINK
```

Pour exécuter le script, exécutez la commande suivante :

```
[root@ip-172-31-27-165 ~]# ./fix_directory.sh /greengrass/v2 2.9.5
Relinking /greengrass/v2/alts/directory_fix to /greengrass/v2/alts/current
Relinking Nucleus artifacts to /greengrass/v2/alts/directory_fix/distro
```

Info:

`com.aws.greengrass.deployment.exceptions.RetryableDeploymentDocumentDownloadException`

Greengrass Cloud Service returned an error when getting full deployment configuration

Cette erreur peut s'afficher lorsque le périphérique principal reçoit un document de déploiement volumineux, qui est un document de déploiement supérieur à 7 Ko (pour les déploiements ciblant des objets) ou 31 Ko (pour les déploiements ciblant des groupes d'objets). Pour récupérer un document de déploiement volumineux, la AWS IoT politique d'un appareil principal doit autoriser `greengrass:GetDeploymentConfiguration` autorisation. Cette erreur peut se produire lorsque le périphérique principal ne dispose pas de cette autorisation. Lorsque cette erreur se produit, le déploiement recommence indéfiniment et son statut est En cours (IN_PROGRESS).

Pour résoudre ce problème, ajoutez `greengrass:GetDeploymentConfiguration` autorisation à la AWS IoT politique de l'appareil principal. Pour de plus amples informations, veuillez consulter [Mettre à jour la AWS IoT politique d'un appareil principal](#).

Warn: `com.aws.greengrass.deployment.DeploymentService: Failed to get thing group hierarchy`

Cet avertissement peut s'afficher lorsque le périphérique principal reçoit un déploiement et que la AWS IoT politique du périphérique principal n'autorise pas cette `greengrass:ListThingGroupsForCoreDevice` autorisation. Lorsque vous créez un déploiement, le périphérique principal utilise cette autorisation pour identifier ses groupes d'objets et supprimer des composants pour tous les groupes d'objets dont vous avez supprimé le périphérique principal. Si le périphérique principal exécute [Greengrass nucleus](#) v2.5.0, le déploiement échoue. Si le périphérique principal exécute `Greengrass nucleus` v2.5.1 ou version ultérieure, le déploiement se poursuit mais aucun composant n'est supprimé. Pour plus d'informations sur le comportement de suppression des groupes d'objets, consultez [Déployer AWS IoT Greengrass des composants sur des appareils](#).

Pour mettre à jour le comportement de l'appareil principal afin de supprimer les composants des groupes d'objets dont vous supprimez le périphérique principal, ajoutez `greengrass:ListThingGroupsForCoreDevice` autorisation à la AWS IoT politique du périphérique principal. Pour de plus amples informations, veuillez consulter [Mettre à jour la AWS IoT politique d'un appareil principal](#).

Info: com.aws.greengrass.deployment.DeploymentDocumentDownloader: Calling Greengrass cloud to get full deployment configuration

Ce message d'information peut s'afficher plusieurs fois sans erreur, car le périphérique principal enregistre l'erreur au niveau du DEBUG journal. Ce problème peut se produire lorsque le périphérique principal reçoit un document de déploiement volumineux. Lorsque ce problème se produit, le déploiement recommence indéfiniment et son statut est En cours (IN_PROGRESS). Pour plus d'informations sur la manière de résoudre ce problème, consultez [cette entrée de résolution des problèmes](#).

Caused by:

software.amazon.awssdk.services.greengrassv2data.model.GreengrassV2DataException: null (Service: GreengrassV2Data, Status Code: 403, Request ID: <some_request_id>, Extended Request ID: null)

Cette erreur peut s'afficher lorsqu'une API de plan de données n'est pas `iot:Connect` autorisée. Si vous n'avez pas la bonne politique, vous recevrez un `GreengrassV2DataException: 403`. Pour créer une politique d'autorisation, suivez ces instructions : [Création d'une AWS IoT politique](#).

Principaux problèmes liés aux composants de l'appareil

Résolvez les problèmes liés aux composants Greengrass sur les principaux appareils.

Rubriques

- [Warn: '<command>' is not recognized as an internal or external command](#)
- [Le script Python n'enregistre pas les messages](#)
- [La configuration des composants ne se met pas à jour lors de la modification de la configuration par défaut](#)
- [awsiot.greengrasscoreipc.model.UnauthorizedError](#)
- [com.aws.greengrass.authorization.exceptions.AuthorizationException: Duplicate policy ID "<id>" for principal "<componentList>"](#)
- [com.aws.greengrass.tes.CredentialRequestHandler: Error in retrieving AwsCredentials from TES \(HTTP 400\)](#)
- [com.aws.greengrass.tes.CredentialRequestHandler: Error in retrieving AwsCredentials from TES \(HTTP 403\)](#)

- [com.aws.greengrass.tes.CredentialsProviderError: Could not load credentials from any providers](#)
- [Received error when attempting to retrieve ECS metadata: Could not connect to the endpoint URL: "<tokenExchangeServiceEndpoint>"](#)
- [copyFrom: <configurationPath> is already a container, not a leaf](#)
- [com.aws.greengrass.componentmanager.plugins.docker.exceptions.DockerLoginException: Error logging into the registry using credentials - 'The stub received bad data.'](#)
- [java.io.IOException: Cannot run program "cmd" ...: \[LogonUser\] The password for this account has expired.](#)
- [aws.greengrass.StreamManager: Instant exceeds minimum or maximum instant](#)

Warn: '<command>' is not recognized as an internal or external command

Cette erreur peut s'afficher dans les journaux d'un composant Greengrass lorsque le logiciel AWS IoT Greengrass Core ne parvient pas à exécuter une commande dans le script de cycle de vie du composant. L'état du composant BROKEN résulte de cette erreur. Cette erreur peut se produire si l'utilisateur du système qui exécute le composant, par exemple `ggc_user`, ne trouve pas le fichier exécutable de la commande dans les dossiers du [PATH](#).

Sur les appareils Windows, vérifiez que le dossier contenant le fichier exécutable est destiné à l'utilisateur du système qui exécute le composant. S'il ne figure pas dans le `PATH`, effectuez l'une des opérations suivantes :

- Ajoutez le dossier de l'exécutable à la variable `PATH` système, qui est accessible à tous les utilisateurs. Redémarrez ensuite le composant.

Si vous exécutez Greengrass nucleus 2.5.0, après avoir mis à jour la variable `PATH` système, vous devez redémarrer le logiciel AWS IoT Greengrass Core pour exécuter les composants avec la mise à jour. `PATH` Si le logiciel AWS IoT Greengrass Core n'utilise pas la mise à jour `PATH` après le redémarrage du logiciel, redémarrez l'appareil et réessayez. Pour de plus amples informations, veuillez consulter [Exécutez le logiciel AWS IoT Greengrass Core](#).

- Ajoutez le dossier de l'exécutable à la variable `PATH` utilisateur correspondant à l'utilisateur du système qui exécute le composant.

Le script Python n'enregistre pas les messages

Les appareils Greengrass Core collectent des journaux que vous pouvez utiliser pour identifier les problèmes liés aux composants. Si vos scripts `stdout` et `stderr` messages Python n'apparaissent pas dans les journaux de vos composants, vous devrez peut-être vider la mémoire tampon ou désactiver la mise en mémoire tampon pour ces flux de sortie standard en Python. Effectuez l'une des actions suivantes :

- Exécutez Python avec l'argument `-u` pour désactiver la mise en mémoire tampon sur `stdout` et `stderr`

Linux or Unix

```
python3 -u hello_world.py
```

Windows

```
py -3 -u hello_world.py
```

- Utilisez [Setenv](#) dans la recette de votre composant pour définir la variable d'environnement [PYTHONUNBUFFERED](#) sur une chaîne non vide. Cette variable d'environnement désactive la mise en mémoire tampon sur `stdout` et `stderr`
- Videz la mémoire tampon pour les `stderr` flux `stdout` or. Effectuez l'une des actions suivantes :
 - Videz un message lorsque vous l'imprimez.

```
import sys

print('Hello, error!', file=sys.stderr, flush=True)
```

- Videz un message après l'avoir imprimé. Vous pouvez envoyer plusieurs messages avant de vider le flux.

```
import sys

print('Hello, error!', file=sys.stderr)
sys.stderr.flush()
```

Pour plus d'informations sur la façon de vérifier que votre script Python génère des messages de journal, consultez [AWS IoT Greengrass Journaux de surveillance](#).

La configuration des composants ne se met pas à jour lors de la modification de la configuration par défaut

Lorsque vous modifiez la recette `DefaultConfiguration` d'un composant, la nouvelle configuration par défaut ne remplacera pas la configuration existante du composant lors d'un déploiement. Pour appliquer la nouvelle configuration par défaut, vous devez rétablir les paramètres par défaut du composant. Lorsque vous déployez le composant, spécifiez une seule chaîne vide comme [mise à jour de réinitialisation](#).

Console

Réinitialiser les chemins

```
[ ]
```

AWS CLI

La commande suivante crée un déploiement sur un périphérique principal.

```
aws greengrassv2 create-deployment --cli-input-json file://reset-configuration-deployment.json
```

Le `reset-configuration-deployment.json` fichier contient le document JSON suivant.

```
{
  "targetArn": "arn:aws:iot:us-west-2:123456789012:thing/MyGreengrassCore",
  "deploymentName": "Deployment for MyGreengrassCore",
  "components": {
    "com.example.HelloWorld": {
      "componentVersion": "1.0.0",
      "configurationUpdate": {,
        "reset": [ ]
      }
    }
  }
}
```

Greengrass CLI

La commande [Greengrass CLI](#) suivante crée un déploiement local sur un périphérique principal.

```
sudo greengrass-cli deployment create \  
  --recipeDir recipes \  
  --artifactDir artifacts \  
  --merge "com.example.HelloWorld=1.0.0" \  
  --update-config reset-configuration-deployment.json
```

Le `reset-configuration-deployment.json` fichier contient le document JSON suivant.

```
{  
  "com.example.HelloWorld": {  
    "RESET": [""]  
  }  
}
```

`awsiot.greengrasscoreipc.model.UnauthorizedError`

Vous pouvez voir cette erreur dans les journaux d'un composant Greengrass lorsque le composant n'est pas autorisé à effectuer une opération IPC sur une ressource. Pour autoriser un composant à appeler une opération IPC, définissez une politique d'autorisation IPC dans la configuration du composant. Pour de plus amples informations, veuillez consulter [Autoriser les composants à effectuer des opérations IPC](#).

Tip

Si vous modifiez la `DefaultConfiguration` recette d'un composant, vous devez rétablir la configuration du composant à sa nouvelle configuration par défaut. Lorsque vous déployez le composant, spécifiez une seule chaîne vide comme [mise à jour de réinitialisation](#). Pour de plus amples informations, veuillez consulter [La configuration des composants ne se met pas à jour lors de la modification de la configuration par défaut](#).

`com.aws.greengrass.authorization.exceptions.AuthorizationException:` `Duplicate policy ID "<id>" for principal "<componentList>"`

Cette erreur peut s'afficher si plusieurs politiques d'autorisation IPC, y compris pour tous les composants du périphérique principal, utilisent le même ID de stratégie.

Vérifiez les politiques d'autorisation IPC de vos composants, corrigez les doublons et réessayez. Pour créer une politique unique IDs, nous vous recommandons de combiner le nom du composant, le nom du service IPC et un compteur. Pour de plus amples informations, veuillez consulter [Autoriser les composants à effectuer des opérations IPC](#).

 Tip

Si vous modifiez la DefaultConfiguration recette d'un composant, vous devez rétablir la configuration du composant à sa nouvelle configuration par défaut. Lorsque vous déployez le composant, spécifiez une seule chaîne vide comme [mise à jour de réinitialisation](#). Pour de plus amples informations, veuillez consulter [La configuration des composants ne se met pas à jour lors de la modification de la configuration par défaut](#).

com.aws.greengrass.tes.CredentialRequestHandler: Error in retrieving AwsCredentials from TES (HTTP 400)

Cette erreur peut s'afficher lorsqu'un appareil principal ne parvient pas à obtenir les AWS informations d'identification du [service d'échange de jetons](#). Le code d'état HTTP 400 indique que cette erreur s'est produite parce que le [rôle IAM d'échange de jetons](#) du périphérique principal n'existe pas ou n'a aucune relation de confiance permettant au fournisseur AWS IoT d'informations d'identification de l'assumer.

Procédez comme suit :

1. Identifiez le rôle d'échange de jetons utilisé par le périphérique principal. Le message d'erreur inclut l'alias de AWS IoT rôle du périphérique principal, qui pointe vers le rôle d'échange de jetons. Exécutez la commande suivante sur votre ordinateur de développement et remplacez-la *MyGreengrassCoreTokenExchangeRoleAlias* par le nom de l'alias de AWS IoT rôle indiqué dans le message d'erreur.

```
aws iot describe-role-alias --role-alias MyGreengrassCoreTokenExchangeRoleAlias
```

La réponse inclut le nom de ressource Amazon (ARN) du rôle IAM d'échange de jetons.

```
{
  "roleAliasDescription": {
    "roleAlias": "MyGreengrassCoreTokenExchangeRoleAlias",
```



```

    "roleAliasArn": "arn:aws:iot:us-west-2:123456789012:rolealias/
MyGreengrassCoreTokenExchangeRoleAlias",
    "roleArn": "arn:aws:iam::123456789012:role/MyGreengrassV2TokenExchangeRole",
    "owner": "123456789012",
    "credentialDurationSeconds": 3600,
    "creationDate": "2021-02-05T16:46:18.042000-08:00",
    "lastModifiedDate": "2021-02-05T16:46:18.042000-08:00"
  }
}

```

2. Vérifiez que le rôle existe. Exécutez la commande suivante et remplacez-la *MyGreengrassV2TokenExchangeRole* par le nom du rôle d'échange de jetons.

```
aws iam get-role --role-name MyGreengrassV2TokenExchangeRole
```

Si la commande renvoie une `NoSuchEntity` erreur, le rôle n'existe pas et vous devez le créer. Pour plus d'informations sur la création et la configuration de ce rôle, consultez [Autoriser les appareils principaux à interagir avec les AWS services](#).

3. Vérifiez que le rôle possède une relation de confiance qui permet au fournisseur AWS IoT d'informations d'identification de l'assumer. La réponse de l'étape précédente contient un `AssumeRolePolicyDocument`, qui définit les relations de confiance du rôle. Le rôle doit définir une relation de confiance permettant `credentials.iot.amazonaws.com` de l'assumer. Ce document doit ressembler à l'exemple suivant.

JSON

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "credentials.iot.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}

```

Si les relations de confiance du rôle ne permettent pas `credentials.iot.amazonaws.com` de l'assumer, vous devez ajouter cette relation de confiance au rôle. Pour plus d'informations, consultez [Modification d'un rôle](#) dans le Guide de l'utilisateur Gestion des identités et des accès AWS .

`com.aws.greengrass.tes.CredentialRequestHandler: Error in retrieving AwsCredentials from TES (HTTP 403)`

Cette erreur peut s'afficher lorsqu'un appareil principal ne parvient pas à obtenir les AWS informations d'identification du [service d'échange de jetons](#). Le code d'état HTTP 403 indique que cette erreur s'est produite parce que les AWS IoT politiques du périphérique principal n'accordent pas `iot:AssumeRoleWithCertificate` autorisation d'utiliser l'alias de AWS IoT rôle du périphérique principal.

Passez en revue les AWS IoT politiques de l'appareil principal et ajoutez `iot:AssumeRoleWithCertificate` autorisation pour l'alias de AWS IoT rôle de l'appareil principal. Le message d'erreur inclut l'alias de AWS IoT rôle actuel du périphérique principal. Pour plus d'informations sur cette autorisation et sur la manière de mettre à jour les AWS IoT politiques de l'appareil principal, consultez [AWS IoT Politique minimale pour les appareils AWS IoT Greengrass V2 principaux](#) et [Mettre à jour la AWS IoT politique d'un appareil principal](#).

`com.aws.greengrass.tes.CredentialsProviderError: Could not load credentials from any providers`

Cette erreur peut s'afficher lorsque le composant essaie de demander des AWS informations d'identification et ne parvient pas à se connecter au [service d'échange de jetons](#).

Procédez comme suit :

- Vérifiez que le composant déclare une dépendance à l'égard du composant du service d'échange de jetons, `aws.greengrass.TokenExchangeService`. Si ce n'est pas le cas, ajoutez la dépendance et redéployez le composant.
- Si le composant s'exécute dans docker, assurez-vous d'appliquer les bons paramètres réseau et variables d'environnement, en fonction de [Utiliser les AWS informations d'identification dans les composants du conteneur Docker \(Linux\)](#)

- [Si le composant est écrit en NodeJS, définissez dns.setDefaultResultCommandez à `ipv4first`](#).
- Vérifiez `/etc/hosts` s'il s'agit d'une entrée commençant par `::1` et contenant `localhost`. Supprimez l'entrée pour voir si le composant s'est connecté au service d'échange de jetons à la mauvaise adresse.

Received error when attempting to retrieve ECS metadata: Could not connect to the endpoint URL: "<tokenExchangeServiceEndpoint>"

Cette erreur peut s'afficher lorsque le composant n'exécute pas le [service d'échange de jetons](#) et qu'un composant essaie de demander des AWS informations d'identification.

Procédez comme suit :

- Vérifiez que le composant déclare une dépendance à l'égard du composant du service d'échange de jetons, `aws.greengrass.TokenExchangeService`. Si ce n'est pas le cas, ajoutez la dépendance et redéployez le composant.
- Vérifiez si le composant utilise des AWS informations d'identification dans son `install` cycle de vie. AWS IoT Greengrass ne garantit pas la disponibilité du service d'échange de jetons pendant le `install` cycle de vie. Mettez à jour le composant pour déplacer le code qui utilise les AWS informations d'identification dans le `startup` run cycle de vie de l'unité d'exploitation, puis redéployez le composant.

copyFrom: <configurationPath> is already a container, not a leaf

Cette erreur peut s'afficher lorsque vous modifiez une valeur de configuration d'un type de conteneur (liste ou objet) à un type non conteneur (chaîne, nombre ou booléen). Procédez comme suit :

1. Vérifiez la recette du composant pour voir si sa configuration par défaut définit cette valeur de configuration comme une liste ou un objet. Si tel est le cas, supprimez ou modifiez cette valeur de configuration.
2. Créez un déploiement pour rétablir cette valeur de configuration à sa valeur par défaut. Pour plus d'informations, consultez [Créer des déploiements](#) et [Mettre à jour les configurations des composants](#).

Vous pouvez ensuite définir cette valeur de configuration sur une chaîne, un nombre ou un booléen.

`com.aws.greengrass.componentmanager.plugins.docker.exceptions.DockerLoginException`
Error logging into the registry using credentials - 'The stub received bad data.'

Cette erreur peut s'afficher dans les journaux du noyau de Greengrass lorsque le [composant du gestionnaire d'applications Docker](#) essaie de télécharger une image Docker depuis un dépôt privé dans Amazon Elastic Container Registry (Amazon ECR). Cette erreur se produit si vous utilisez l'[assistant d'identification wincred Docker](#) (`docker-credential-wincred`). Par conséquent, Amazon ECR n'est pas en mesure de stocker les informations de connexion.

Effectuez l'une des actions suivantes :

- Si vous n'utilisez pas l'assistant d'identification wincred Docker, supprimez le `docker-credential-wincred` programme de l'appareil principal.
- Si vous utilisez l'assistant d'identification wincred Docker, procédez comme suit :
 1. Renommez le `docker-credential-wincred` programme sur l'appareil principal. Remplacez-le par un nouveau nom pour l'assistant d'identification Windows Docker. Par exemple, vous pouvez le renommer `endocker-credential-wincredreal`.
 2. Mettez à jour l'`credsStore` option dans le fichier de configuration Docker (`.docker/config.json`) pour utiliser le nouveau nom de l'assistant d'identification Windows Docker. Par exemple, si vous avez renommé le programme `endocker-credential-wincredreal`, mettez à jour l'`credsStore` option en `wincredreal`.

```
{  
  "credsStore": "wincredreal"  
}
```

`java.io.IOException: Cannot run program "cmd" ...: [LogonUser] The password for this account has expired.`

Cette erreur peut s'afficher sur un périphérique Windows principal lorsque l'utilisateur du système qui exécute les processus du composant, par exemple `ggc_user`, a un mot de passe expiré. Par conséquent, le logiciel AWS IoT Greengrass Core n'est pas en mesure d'exécuter les processus des composants en tant qu'utilisateur du système.

Pour mettre à jour le mot de passe d'un utilisateur du système Greengrass

1. Exécutez la commande suivante en tant qu'administrateur pour définir le mot de passe de l'utilisateur. Remplacez *ggc_user* par l'utilisateur du système et remplacez *password* par le mot de passe à définir.

```
net user ggc_user password
```

2. Utilisez l'[PsExec utilitaire](#) pour enregistrer le nouveau mot de passe de l'utilisateur dans l'instance Credential Manager du LocalSystem compte. *password* Remplacez-le par le mot de passe utilisateur que vous avez défini.

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

Tip

En fonction de votre configuration Windows, le mot de passe de l'utilisateur peut être configuré pour expirer à une date ultérieure. Pour vous assurer que vos applications Greengrass continuent de fonctionner, suivez la date d'expiration du mot de passe et mettez-le à jour avant son expiration. Vous pouvez également définir le mot de passe de l'utilisateur pour qu'il n'expire jamais.

- Pour vérifier la date d'expiration d'un utilisateur et de son mot de passe, exécutez la commande suivante.

```
net user ggc_user | findstr /C:expires
```

- Pour définir le mot de passe d'un utilisateur afin qu'il n'expire jamais, exécutez la commande suivante.

```
wmic UserAccount where "Name='ggc_user'" set PasswordExpires=False
```

- Si vous utilisez Windows 10 ou une version ultérieure où la [wmi commande est obsolète](#), exécutez la commande suivante PowerShell .

```
Get-CimInstance -Query "SELECT * from Win32_UserAccount WHERE name =  
'ggc_user'" | Set-CimInstance -Property @{PasswordExpires="False"}
```

aws.greengrass.StreamManager: Instant exceeds minimum or maximum instant

Lorsque vous mettez à niveau le gestionnaire de flux v2.0.7 vers une version comprise entre v2.0.8 et v2.0.11, vous pouvez voir l'erreur suivante dans les journaux du composant Stream Manager si le composant 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 avez déployé le gestionnaire de flux v2.0.7 et que vous souhaitez effectuer une mise à niveau vers une version ultérieure, vous devez passer directement au gestionnaire de flux v2.0.12. Pour plus d'informations sur le composant du gestionnaire de flux, consultez [Gestionnaire de flux](#).

Principaux problèmes liés aux composants de la fonction Lambda de l'appareil

Résolvez les problèmes liés aux composants de la fonction Lambda sur les appareils principaux.

Rubriques

- [The following cgroup subsystems are not mounted: devices, memory](#)
- [ipc_client.py:64,HTTP Error 400:Bad Request, b'No subscription exists for the source <label-or-lambda-arn> and subject <label-or-lambda-arn>](#)

The following cgroup subsystems are not mounted: devices, memory

Cette erreur peut s'afficher lorsque vous exécutez une fonction Lambda conteneurisée dans les cas suivants :

- Le cgroup v1 n'est pas activé pour la mémoire ou les cgroups de périphériques sur le périphérique principal.
- Le périphérique principal a activé cgroups v2. Les fonctions Lambda de Greengrass nécessitent des cgroups v1, et les cgroups v1 et v2 s'excluent mutuellement.

Pour activer cgroups v1, démarrez le périphérique avec les paramètres du noyau Linux suivants.

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

Tip

Sur un Raspberry Pi, modifiez le `/boot/cmdline.txt` fichier pour définir les paramètres du noyau de l'appareil.

`ipc_client.py:64,HTTP Error 400:Bad Request, b'No subscription exists for the source <label-or-lambda-arn> and subject <label-or-lambda-arn>`

Cette erreur peut s'afficher lorsque vous exécutez une fonction Lambda V1, qui utilise le SDK AWS IoT Greengrass Core, sur un périphérique principal V2 sans spécifier d'abonnement dans l'[ancien composant routeur d'abonnement](#). Pour résoudre ce problème, déployez et configurez l'ancien routeur d'abonnement afin de spécifier les abonnements requis. Pour de plus amples informations, veuillez consulter [Importer les fonctions Lambda de la V1](#).

Version du composant abandonnée

Une notification peut s'afficher sur votre Personal Health Dashboard (PHD) lorsqu'une version d'un composant de votre appareil principal est abandonnée. La version du composant envoie cette notification à votre PHD dans les 60 minutes suivant son arrêt.

Pour voir quels déploiements vous devez réviser, procédez comme suit à l'aide de AWS Command Line Interface :

1. Exécutez la commande suivante pour obtenir la liste de vos principaux appareils.

```
aws greengrassv2 list-core-devices
```

2. Exécutez la commande suivante pour récupérer l'état des composants de chaque périphérique principal à partir de l'étape 1. Remplacez *coreDeviceName* par le nom de chaque périphérique principal à interroger.

```
aws greengrassv2 list-installed-components --core-device-thing-name coreDeviceName
```

3. Rassemblez les principaux appareils sur lesquels la version du composant abandonnée est installée comme indiqué dans les étapes précédentes.
4. Exécutez la commande suivante pour récupérer l'état de toutes les tâches de déploiement pour chaque périphérique principal à partir de l'étape 3. Remplacez *coreDeviceName* par le nom du périphérique principal à interroger.

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

La réponse contient la liste des tâches de déploiement pour le périphérique principal. Vous pouvez revoir le déploiement pour choisir une autre version du composant. Pour plus d'informations sur la façon de réviser un déploiement, consultez la section [Réviser les déploiements](#).

Problèmes liés à l'interface de ligne de commande Greengrass

Résolvez les problèmes liés à la CLI [Greengrass](#).

Rubriques

- [java.lang.RuntimeException: Unable to create ipc client](#)

java.lang.RuntimeException: Unable to create ipc client

Cette erreur peut s'afficher lorsque vous exécutez une commande Greengrass CLI et que vous spécifiez un dossier racine différent de celui dans lequel le logiciel AWS IoT Greengrass Core est installé.

Procédez de l'une des manières suivantes pour définir le chemin racine et remplacez-le */greengrass/v2* par le chemin d'installation de votre logiciel AWS IoT Greengrass Core :

- Définissez la variable d'environnement GGC_ROOT_PATH sur */greengrass/v2*.

- Ajoutez l'option `--ggcRootPath /greengrass/v2` à votre commande comme indiqué dans l'exemple suivant.

```
greengrass-cli --ggcRootPath /greengrass/v2 <command> <subcommand> [arguments]
```

AWS Command Line Interface problèmes

Résoudre les AWS CLI problèmes liés à. AWS IoT Greengrass V2

Rubriques

- [Error: Invalid choice: 'greengrassv2'](#)

Error: Invalid choice: 'greengrassv2'

Cette erreur peut s'afficher lorsque vous exécutez une AWS IoT Greengrass V2 commande avec le AWS CLI (par exemple, `aws greengrassv2 list-core-devices`).

Cette erreur indique que vous disposez d'une version AWS CLI qui n'est pas compatible AWS IoT Greengrass V2. Pour l'utiliser AWS IoT Greengrass V2 avec le AWS CLI, vous devez disposer de l'une des versions suivantes ou d'une version ultérieure :

- Version AWS CLI V1 minimale : v1.18.197
- Version AWS CLI V2 minimale : v2.1.11

Tip

Vous pouvez exécuter la commande suivante pour vérifier la version AWS CLI dont vous disposez.

```
aws --version
```

Pour résoudre ce problème, mettez à jour le AWS CLI vers une version ultérieure qui prend en charge AWS IoT Greengrass V2. Pour plus d'informations, consultez la section [Installation, mise à jour et désinstallation du AWS CLI dans le](#) guide de l'AWS Command Line Interface utilisateur.

Codes d'erreur de déploiement détaillés

Utilisez les codes d'erreur et les solutions présentés dans ces sections pour résoudre les problèmes liés au déploiement des composants lors de l'utilisation du noyau Greengrass version 2.8.0 ou ultérieure.

Le noyau Greengrass signale les erreurs de déploiement sous la forme d'une hiérarchie allant du code le moins spécifique au code le plus spécifique disponible. Vous pouvez utiliser cette hiérarchie pour identifier la raison d'une erreur de déploiement. Par exemple, voici une hiérarchie d'erreurs possible :

- ÉCHEC DU DÉPLOIEMENT
 - ERREUR DE TÉLÉCHARGEMENT D'ARTÉFACT
 - IO_ERROR
 - DISK_SPACE_CRITICAL

Les codes d'erreur sont organisés en types. Chaque type représente une classe d'erreurs susceptibles de se produire. AWS IoT Greengrass signale ces types d'erreurs dans la console, l'API et AWS CLI. Il peut y avoir plusieurs types d'erreur, en fonction des erreurs signalées dans la hiérarchie des erreurs. Pour l'exemple précédent, le type d'erreur renvoyé est `DEVICE_ERROR`.

Les types sont les suivants :

- `PERMISSION_ERROR` — L'accès à une opération nécessitant une autorisation a été refusé.
- `REQUEST_ERROR` — Une erreur s'est produite en raison d'un problème dans le document de déploiement.
- `COMPONENT_RECIPES_ERROR` — Une erreur s'est produite en raison d'un problème dans la recette d'un composant.
- `AWS_COMPONENT_ERROR` — Une erreur s'est produite lors du démarrage ou de la suppression d'un composant AWS fourni.
- `USER_COMPONENT_ERROR` — Une erreur s'est produite lors du démarrage ou de la suppression d'un composant utilisateur.
- `COMPONENT_ERROR` — Une erreur s'est produite lors du démarrage ou de la suppression d'un composant, mais le noyau Greengrass n'a pas pu déterminer si le composant est un composant AWS fourni ou un composant utilisateur.

- **DEVICE_ERROR** — Une erreur s'est produite avec les E/S locales ou une autre erreur de périphérique s'est produite.
- **DEPENDENCY_ERROR** — Un déploiement n'a pas réussi à télécharger un artefact depuis Amazon S3 ou à extraire une image d'un registre ECR.
- **HTTP_ERROR** — Une erreur s'est produite lors d'une requête HTTP.
- **NETWORK_ERROR** — Une erreur s'est produite sur le réseau de l'appareil.
- **NUCLEUS_ERROR** — Le noyau de Greengrass n'a pas pu localiser un composant ou n'a pas pu trouver la version du noyau actif.
- **SERVER_ERROR** — Un serveur a renvoyé une erreur 500 en réponse à une demande.
- **CLOUD_SERVICE_ERROR** — Une erreur s'est produite avec le service cloud. AWS IoT Greengrass
- **UNKNOWN_ERROR** — Une exception non vérifiée a été émise par le composant.

La plupart des erreurs de cette section signalent des informations supplémentaires dans les journaux de AWS IoT Greengrass base. Ces journaux sont stockés dans le système de fichiers local de l'appareil principal. Il existe des journaux pour le logiciel AWS IoT Greengrass principal et pour chaque composant individuel. Pour plus d'informations sur l'accès aux journaux, consultez [Accéder aux journaux du système de fichiers](#).

Erreur d'autorisation

ACCÈS REFUSÉ

Cette erreur peut se produire lorsqu'une opération de AWS service renvoie une erreur 403 parce que les autorisations ne sont pas correctement configurées. Consultez le code d'erreur le plus spécifique pour plus de détails.

GET_DEPLOYMENT_CONFIGURATION_ACCESS_DENIED

Cette erreur peut se produire lorsque la AWS IoT politique ne permet pas d'appeler l'GetDeploymentConfigurationopération. Ajoutez l'greengrass::GetDeploymentConfigurationautorisation à la politique de l'appareil principal.

GET_COMPONENT_VERSION_ARTIFACT_ACCESS_DENIED

Cette erreur peut se produire lorsque la AWS IoT politique principale de l'appareil n'autorise pas `greengrass:GetComponentVersionArtifact` autorisation. Ajoutez l'autorisation à la politique de l'appareil principal.

RESOLVE_COMPONENT_CANDIDATES_ACCESS_DENIED

Cette erreur peut se produire lorsque la AWS IoT politique principale de l'appareil n'autorise pas `greengrass:ResolveComponentCandidates` autorisation. Ajoutez l'autorisation à la politique de l'appareil principal.

GET_ECR_CREDENTIAL_ERROR

Cette erreur peut se produire lorsque le déploiement n'a pas pu s'authentifier auprès d'un registre privé dans ECR. Vérifiez la présence d'une erreur spécifique dans le journal, puis réessayez le déploiement.

UTILISATEUR NON AUTORISÉ POUR DOCKER

Cette erreur peut se produire lorsque l'utilisateur de Greengrass n'est pas autorisé à utiliser Docker. Assurez-vous que vous utilisez Greengrass en tant que root ou que l'utilisateur est ajouté au docker groupe. Réessayez ensuite le déploiement.

S3_ACCESS_DENIED

Cette erreur peut se produire lorsqu'une opération Amazon S3 renvoie une erreur 403. Consultez tous les codes d'erreur ou journaux supplémentaires pour plus de détails.

S3_HEAD_OBJECT_ACCESS_DENIED

Cette erreur peut se produire soit lorsque le rôle d'échange de jetons de l'appareil n'autorise pas le logiciel AWS IoT Greengrass Core à télécharger l'artefact du composant à partir de l'URL de l'objet S3 que vous spécifiez dans la recette du composant, soit lorsque l'artefact du composant n'est pas disponible. Vérifiez que le rôle d'échange de jetons autorise `s3:GetObject` l'URL de l'objet S3 où l'artefact est disponible et que l'artefact est présent.

S3_GET_BUCKET_LOCATION_ACCESS_DENIED

Cette erreur peut se produire lorsque le rôle d'échange de jetons de l'appareil n'autorise pas `s3:GetBucketLocation` accès au compartiment Amazon S3 dans lequel l'artefact est disponible. Vérifiez que l'appareil autorise l'autorisation, puis réessayez le déploiement.

S3_GET_OBJECT_ACCESS_DENIED

Cette erreur peut se produire soit lorsque le rôle d'échange de jetons de l'appareil n'autorise pas le logiciel AWS IoT Greengrass Core à télécharger l'artefact du composant à partir de l'URL de l'objet S3 que vous spécifiez dans la recette du composant, soit lorsque l'artefact du composant n'est pas disponible. Vérifiez que le rôle d'échange de jetons autorise `s3:GetObject` l'URL de l'objet S3 où l'artefact est disponible et que l'artefact est présent.

Erreur de demande

CAPACITÉS REQUISES_MANQUANTES DU NOYAU

Cette erreur peut se produire lorsque la version du noyau du déploiement ne permet pas une opération demandée, telle que le téléchargement d'une configuration volumineuse ou la définition de limites de ressources Linux. Réessayez le déploiement avec une version du noyau compatible avec l'opération.

ERREUR MULTIPLE_NUCLEUS_RESOLVED_ERROR

Cette erreur peut se produire lorsqu'un déploiement tente de déployer plusieurs composants du noyau. Consultez le journal pour voir la cause de l'erreur, puis consultez la page de mise à jour du logiciel Nucleus pour voir si le problème a été corrigé dans une version ultérieure du Nucleus, ou contactez Support.

COMPONENT_CIRCULAR_DEPENDENCY_ERROR

Cette erreur peut se produire lorsque deux composants de votre déploiement dépendent l'un de l'autre. Réviser la configuration des composants afin que les composants de votre déploiement ne dépendent pas les uns des autres.

UNAUTHORIZED_NUCLEUS_MINOR_VERSION_UPDATE

Cette erreur peut se produire lorsqu'un composant de votre déploiement nécessite une mise à jour d'une version mineure du noyau, mais que cette version n'est pas spécifiée dans le déploiement. Cela permet de réduire les mises à jour mineures accidentelles pour les composants qui dépendent d'une version différente. Incluez la nouvelle version mineure du noyau dans le déploiement.

GESTIONNAIRE D'APPLICATIONS DOCKER_MANQUANT

Cette erreur peut se produire lorsque vous déployez un composant Docker sans déployer le gestionnaire d'applications Docker. Assurez-vous que votre déploiement inclut le gestionnaire d'applications Docker.

SERVICE D'ÉCHANGE DE JETONS MANQUANTS

Cette erreur peut se produire lorsque le déploiement souhaite télécharger un artefact d'image Docker à partir d'un registre ECR privé sans déployer le service d'échange de jetons. Assurez-vous que votre déploiement inclut le service d'échange de jetons.

LES EXIGENCES RELATIVES À LA VERSION DU COMPOSANT NE SONT PAS SATISFAITES

Cette erreur peut s'afficher en cas de conflit de contrainte de version ou en cas d'inexistence d'une version de composant. Pour de plus amples informations, veuillez consulter [Error: com.aws.greengrass.componentmanager.exceptions.NoAvailableComponentVersionException: Failed to negotiate component <name> version with cloud and no local applicable version satisfying requirement <requirements>](https://docs.aws.amazon.com/greengrass/componentmanager/exceptions/NoAvailableComponentVersionException-Failed-to-negotiate-component-Name-version-with-cloud-and-no-local-applicable-version-satisfying-requirement-Requirements).

THROTTLING_ERROR

Cette erreur peut se produire lorsqu'une opération AWS de service a dépassé un quota de débit. Retentez le déploiement.

DEMANDE_CONFLICTUELLE

Cette erreur peut se produire lorsqu'une opération de AWS service renvoie une erreur 409 parce que votre déploiement tente d'effectuer plusieurs opérations à la fois. Retentez le déploiement.

RESSOURCE INTROUVABLE

Cette erreur peut se produire lorsqu'une opération de AWS service renvoie une erreur 404 parce qu'une ressource est introuvable. Vérifiez la ressource manquante dans le journal.

EXÉCUTER_AVEC_CONFIG_NOT_VALID

Cette erreur peut se produire lorsque les windowsUser informations posixUserposixGroup, ou spécifiées pour exécuter le composant ne sont pas valides. Vérifiez que l'utilisateur est valide, puis réessayez le déploiement.

RÉGION_NON PRISE EN CHARGE

Cette erreur peut se produire lorsque la région spécifiée pour le déploiement n'est pas prise en charge par AWS IoT Greengrass. Vérifiez la région et réessayez le déploiement.

IOT_CRED_ENDPOINT_NOT_VALID

Cette erreur peut se produire lorsque le point de terminaison des AWS IoT informations d'identification spécifié dans la configuration n'est pas valide. Vérifiez le point de terminaison et réessayez votre demande.

IOT_DATA_ENDPOINT_NOT_VALID

Cette erreur peut se produire lorsque le point de terminaison de AWS IoT données spécifié dans la configuration n'est pas valide. Vérifiez le point de terminaison et réessayez votre demande.

S3_HEAD_OBJECT_RESOURCE_NOT_FOUND

Cette erreur peut se produire lorsque l'artefact du composant n'est pas disponible à l'URL de l'objet S3 que vous spécifiez dans la recette du composant. Vérifiez que vous avez chargé l'artefact dans le compartiment S3 et que l'URI de l'artefact correspond à l'URL de l'objet S3 de l'artefact dans le compartiment.

S3_GET_BUCKET_LOCATION_RESOURCE_NOT_FOUND

Cette erreur peut s'afficher lorsque le compartiment Amazon S3 est introuvable. Vérifiez que le compartiment existe et réessayez le déploiement.

S3_GET_OBJECT_RESOURCE_NOT_FOUND

Cette erreur peut se produire lorsque l'artefact du composant n'est pas disponible à l'URL de l'objet S3 que vous spécifiez dans la recette du composant. Vérifiez que vous avez chargé l'artefact dans le compartiment S3 et que l'URI de l'artefact correspond à l'URL de l'objet S3 de l'artefact dans le compartiment.

IO_MAPPING_ERROR

Cette erreur peut se produire lorsqu'une erreur d'E/S se produit lors de l'analyse d'un document ou d'une recette de déploiement. Consultez tous les codes d'erreur ou journaux supplémentaires pour plus de détails.

Erreur de recette du composant

ERREUR D'ANALYSE DE LA RECETTE

Cette erreur peut se produire lorsque la recette de déploiement n'a pas pu être analysée en raison d'une erreur dans la structure de la recette. Vérifiez que la recette est correctement formatée et réessayez le déploiement.

RECEIPE_METADATA_PARSE_ERROR

Cette erreur peut se produire lorsque les métadonnées de la recette de déploiement téléchargées depuis le cloud ne peuvent pas être analysées. Contacter Support.

ARTEFACT_URI_NOT_VALID

Cette erreur peut se produire lorsqu'un URI d'artefact dans une recette n'est pas formaté correctement. Vérifiez dans le journal l'URI qui n'est pas valide, mettez à jour l'URI dans la recette, puis réessayez le déploiement.

S3_ARTIFACT_URI_NOT_VALID

Cette erreur peut se produire lorsque l'URI Amazon S3 d'un artefact dans une recette n'est pas valide. Vérifiez dans le journal l'URI qui n'est pas valide, mettez à jour l'URI dans la recette, puis réessayez le déploiement.

DOCKER_ARTIFACT_URI_NOT_VALID

Cette erreur peut se produire lorsque l'URI Docker d'un artefact dans une recette n'est pas valide. Vérifiez dans le journal l'URI qui n'est pas valide, mettez à jour l'URI dans la recette, puis réessayez le déploiement.

EMPTY_ARTIFACT_URI

Cette erreur peut se produire lorsque l'URI d'un artefact n'est pas spécifié dans une recette. Vérifiez dans le journal l'artefact pour lequel il manque un URI, mettez à jour l'URI dans la recette, puis réessayez le déploiement.

SCHÉMA D'ARTEFACT_VIDE

Cette erreur peut se produire lorsqu'aucun schéma d'URI n'est défini pour un artefact. Vérifiez dans le journal l'URI qui n'est pas valide, mettez à jour l'URI dans la recette, puis réessayez le déploiement.

SCHEME_ARTEFACT_NON PRIS EN CHARGE

Cette erreur peut se produire lorsqu'un schéma d'URI n'est pas pris en charge par la version de Nucleus en cours d'exécution. Soit un URI n'est pas valide, soit vous devez mettre à jour la version du noyau. Si l'URI n'est pas valide, recherchez dans le journal l'URI non valide, mettez à jour l'URI dans la recette, puis réessayez le déploiement.

MANIFESTE MANQUANT DE LA RECETTE

Cette erreur peut se produire lorsque la section du manifeste n'est pas incluse dans la recette. Ajoutez le manifeste à la recette et réessayez le déploiement.

RECIPE_MISSING_ARTEFACT_HASH_ALGORITHM

Cette erreur peut se produire lorsqu'un artefact non local est spécifié dans une recette sans algorithme de hachage. Ajoutez l'algorithme à l'artefact, puis réessayez la demande.

ARTEFACT_CHECKSUM_MISMATCH

Cette erreur peut se produire lorsqu'un artefact téléchargé possède un condensé différent de celui spécifié dans la recette. Assurez-vous que la recette contient le bon résumé, puis réessayez le déploiement. Pour plus d'informations, voir [Error: com.aws.greengrass.componentmanager.exceptions.ArtifactChecksumMismatchException: Integrity check for downloaded artifact failed. Probably due to file corruption.](https://docs.aws.amazon.com/greengrass/v2.0/developerguide/artifact-checksum-mismatch-exception.html)

COMPONENT_DEPENDENCY_NOT_VALID

Cette erreur peut se produire lorsque le type de dépendance spécifié dans une recette de déploiement n'est pas valide. Vérifiez la recette, puis réessayez votre demande.

ERREUR D'INTERPOLATION DE CONFIGURATION

Cette erreur peut se produire lors de l'interpolation d'une variable de recette. Consultez le journal pour plus de détails.

IO_MAPPING_ERROR

Cette erreur peut se produire lorsqu'une erreur d'E/S se produit lors de l'analyse d'un document ou d'une recette de déploiement. Consultez tous les codes d'erreur ou journaux supplémentaires pour plus de détails.

AWS erreur de composant, erreur de composant utilisateur, erreur de composant

Les codes d'erreur suivants sont renvoyés en cas de problème avec un composant. Le type d'erreur réellement signalé dépend du composant spécifique à l'origine de l'erreur. Si le noyau Greengrass identifie le composant comme étant fourni par AWS IoT Greengrass, il revient. **AWS_COMPONENT_ERROR** Si le composant est identifié comme un composant utilisateur, le noyau Greengrass revient. **USER_COMPONENT_ERROR** Si le noyau de Greengrass ne le sait pas, il revient. **COMPONENT_ERROR**

ERREUR DE MISE À JOUR DU COMPOSANT

Cette erreur peut se produire lorsqu'un composant n'est pas mis à jour lors d'un déploiement. Vérifiez tout code d'erreur supplémentaire ou consultez le journal pour déterminer la cause de l'erreur.

COMPOSANT CASSÉ

Cette erreur peut se produire lorsqu'un composant est cassé lors d'un déploiement. Consultez le journal des composants pour obtenir des informations détaillées sur les erreurs, puis réessayez le déploiement.

REMOVE_COMPONENT_ERROR

Cette erreur peut se produire lorsque le noyau ne parvient pas à supprimer un composant lors d'un déploiement. Consultez le journal pour obtenir des informations détaillées sur les erreurs, puis réessayez le déploiement.

COMPONENT_BOOTSTRAP_TIMEOUT

Cette erreur peut se produire lorsque la tâche d'amorçage d'un composant prend plus de temps que le délai d'attente configuré. Augmentez le délai d'expiration ou réduisez le temps d'exécution de la tâche de démarrage, puis réessayez le déploiement.

COMPONENT_BOOTSTRAP_ERROR

Cette erreur peut se produire lorsque la tâche d'amorçage d'un composant contient une erreur. Consultez le journal pour obtenir des informations détaillées sur les erreurs, puis réessayez le déploiement.

LA CONFIGURATION DU COMPOSANT N'EST PAS VALIDE

Cette erreur peut se produire lorsque le noyau ne parvient pas à valider la configuration déployée pour le composant. Consultez le journal pour obtenir des informations détaillées sur les erreurs, puis réessayez le déploiement.

Erreur de l'appareil

IO_WRITE_ERROR

Cette erreur peut s'afficher lorsque vous écrivez dans un fichier. Consultez le journal pour plus de détails.

IO_READ_ERROR

Cette erreur peut s'afficher lors de la lecture d'un fichier. Consultez le journal pour plus de détails.

DISK_SPACE_CRITICAL

Cette erreur peut se produire lorsqu'il n'y a pas assez d'espace disque pour terminer une demande de déploiement. Vous devez disposer d'au moins 20 Mo d'espace disponible, ou suffisamment pour contenir un artefact de plus grande taille. Libérez de l'espace disque, puis réessayez le déploiement.

IO_FILE_ATTRIBUTE_ERROR

Cette erreur peut se produire lorsque la taille de fichier existante ne peut pas être récupérée à partir du système de fichiers. Consultez le journal pour plus de détails.

SET_PERMISSION_ERROR

Cette erreur peut s'afficher lorsque les autorisations ne peuvent pas être définies sur un artefact ou un répertoire d'artefacts téléchargé. Consultez le journal pour plus de détails.

IO_UNZIP_ERROR

Cette erreur peut s'afficher lorsqu'un artefact ne peut pas être décompressé. Consultez le journal pour plus de détails.

RECETTE_LOCAL_INTROUVABLE

Cette erreur peut se produire lorsque la copie locale d'un fichier de recette est introuvable. Réessayez le déploiement.

LOCAL_RECIPE_CORRUPTED

Cette erreur peut se produire lorsque la copie locale de la recette a changé depuis son téléchargement. Supprimez la copie existante de la recette et réessayez le déploiement.

LOCAL_RECIPE_METADATA_NOT_FOUND

Cette erreur peut se produire lorsque la copie locale du fichier de métadonnées de la recette est introuvable. Réessayez le déploiement.

LAUNCH_DIRECTORY_CORRUPTED

Cette erreur peut se produire lorsque le répertoire utilisé pour lancer le noyau Greengrass (/greengrass/v2/alts/current) a été modifié depuis le dernier démarrage du noyau. Redémarrez le noyau, puis réessayez le déploiement.

ALGORITHME DE HACHAGE INDISPONIBLE

Cette erreur peut se produire lorsque la distribution Java de l'appareil ne prend pas en charge l'algorithme de hachage requis ou lorsque l'algorithme de hachage spécifié dans une recette de composant n'est pas valide.

DEVICE_CONFIG_NOT_VALID_FOR_ARTIFACT_DOWNLOAD

Cette erreur peut se produire lorsqu'une erreur de configuration de l'appareil empêche le déploiement de télécharger l'artefact depuis Amazon S3 ou le cloud Greengrass. Consultez le journal pour détecter une erreur de configuration spécifique, puis réessayez le déploiement.

Erreur de dépendance

ERREUR_DOCKER

Cette erreur peut se produire lorsque vous extrayez une image Docker. Consultez tous les codes d'erreur ou journaux supplémentaires pour plus de détails.

DOCKER_SERVICE_INDISPONIBLE

Cette erreur peut se produire lorsque Greengrass ne parvient pas à se connecter au registre Docker. Vérifiez la présence d'une erreur spécifique dans le journal, puis réessayez le déploiement.

ERREUR_DOCKER_LOGIN_ERROR

Cette erreur peut se produire lorsqu'une erreur inattendue se produit lors de la connexion à Docker. Vérifiez la présence d'une erreur spécifique dans le journal, puis réessayez le déploiement.

ERREUR_DOCKER_PULL_ERROR

Cette erreur peut se produire lorsqu'une erreur inattendue se produit lors de l'extraction d'une image Docker depuis le registre. Vérifiez la présence d'une erreur spécifique dans le journal, puis réessayez le déploiement.

DOCKER_IMAGE_NON_VALIDE

Cette erreur peut s'afficher lorsque l'image Docker demandée n'existe pas. Consultez le journal pour détecter une erreur spécifique et réessayez le déploiement.

DOCKER_IMAGE_QUERY_ERROR

Cette erreur peut se produire lorsqu'une défaillance inattendue se produit lors de l'interrogation de Docker pour obtenir les images disponibles. Consultez le journal pour détecter l'erreur spécifique et réessayez le déploiement.

S3_ERROR

Cette erreur peut s'afficher lors du téléchargement d'un artefact Amazon S3. Consultez tous les codes d'erreur ou journaux supplémentaires pour plus de détails.

S3_RESOURCE_NOT_FOUND

Cette erreur peut se produire lorsqu'une opération Amazon S3 renvoie une erreur 404. Consultez tous les codes d'erreur ou journaux supplémentaires pour plus de détails.

S3_BAD_REQUEST

Cette erreur peut se produire lorsqu'une opération Amazon S3 renvoie une erreur 400. Consultez le journal pour détecter une erreur spécifique et réessayez la demande.

Erreur HTTP

HTTP_REQUEST_ERROR

Cette erreur peut s'afficher lorsqu'une erreur s'est produite lors d'une requête HTTP. Consultez le journal pour détecter l'erreur spécifique.

DOWNLOAD_DEPLOYMENT_DOCUMENT_ERROR

Cette erreur peut se produire lorsqu'une erreur HTTP s'est produite lors du téléchargement du document de déploiement. Vérifiez le journal pour détecter l'erreur HTTP spécifique.

GET_GREENGRASS_ARTIFACT_SIZE_ERROR

Cette erreur peut se produire lorsqu'une erreur HTTP s'est produite lors de l'obtention de la taille d'un artefact de composant public. Vérifiez le journal pour détecter l'erreur HTTP spécifique.

DOWNLOAD_GREENGRASS_ARTIFACT_ERROR

Cette erreur peut se produire lorsqu'une erreur HTTP s'est produite lors du téléchargement d'un artefact de composant public. Vérifiez le journal pour détecter l'erreur HTTP spécifique.

Erreur réseau

ERREUR_RÉSEAU

Cette erreur peut s'afficher en cas de problème de connexion lors d'un déploiement. Vérifiez la connexion de l'appareil à Internet et réessayez le déploiement.

Erreur Nucleus

MAUVAISE DEMANDE

Cette erreur peut se produire lorsqu'une opération AWS sur le cloud renvoie une erreur 400. Consultez le journal pour voir quelle API est à l'origine de l'erreur, puis consultez la page de mise à jour du logiciel Nucleus pour voir si le problème a été corrigé dans une version ultérieure du Nucleus, ou contactez Support.

NUCLEUS_VERSION_INTROUVABLE

Cette erreur peut s'afficher lorsqu'un périphérique principal ne trouve pas la version du noyau actif. Consultez le journal pour voir la cause de l'erreur, puis consultez la page de mise à jour du logiciel Nucleus pour voir si le problème a été corrigé dans une version ultérieure du Nucleus, ou contactez Support.

ÉCHEC DU REDÉMARRAGE DU NOYAU

Cette erreur peut se produire lorsque le noyau ne redémarre pas lors d'un déploiement nécessitant un redémarrage du noyau. Consultez le journal du chargeur pour déterminer la cause de l'erreur, puis consultez la page de mise à jour du logiciel Nucleus pour voir si le problème a été corrigé dans une version ultérieure du Nucleus, ou contactez Support.

COMPOSANT_INSTALLED_NOT_FOUND

Cette erreur peut se produire lorsque le noyau ne parvient pas à localiser un composant installé. Consultez le journal pour voir la cause de l'erreur, puis consultez la page de mise à jour du logiciel Nucleus pour voir si le problème a été corrigé dans une version ultérieure du Nucleus, ou contactez Support.

DOCUMENT_DE DÉPLOIEMENT NON VALIDE

Cette erreur peut se produire lorsque l'appareil reçoit un document de déploiement non valide. Vérifiez tout code d'erreur supplémentaire ou consultez le journal pour déterminer la cause de l'erreur.

DEMANDE_DE_DÉPLOIEMENT VIDE

Cette erreur peut se produire lorsqu'un appareil reçoit une demande de déploiement vide. Consultez le journal pour voir la cause de l'erreur, puis consultez la page de mise à jour du logiciel Nucleus pour voir si le problème a été corrigé dans une version ultérieure du Nucleus, ou contactez Support.

ERREUR D'ANALYSE DU DOCUMENT DE DÉPLOIEMENT

Cette erreur peut se produire lorsque le format de demande de déploiement ne correspond pas au format attendu. Consultez le journal pour voir la cause de l'erreur, puis consultez la page de mise à jour du logiciel Nucleus pour voir si le problème a été corrigé dans une version ultérieure du Nucleus, ou contactez Support.

LES MÉTADONNÉES DES COMPOSANTS NE SONT PAS VALIDES LORS DU DÉPLOIEMENT

Cette erreur peut se produire lorsque la demande de déploiement contient des métadonnées de composant non valides. Consultez le journal pour voir la cause de l'erreur, puis consultez la page de mise à jour du logiciel Nucleus pour voir si le problème a été corrigé dans une version ultérieure du Nucleus, ou contactez Support.

LAUNCH_DIRECTORY_CORRUPTED

Cette erreur peut se produire lorsque vous déplacez un appareil Greengrass d'un groupe d'objets à un autre, puis que vous revenez au groupe d'origine avec des déploiements nécessitant le redémarrage de Greengrass. Pour résoudre l'erreur, recréez le répertoire de lancement de Greengrass sur l'appareil.

Pour de plus amples informations, veuillez consulter [Error: com.amazonaws.greengrass.deployment.exceptions.DeploymentException: Unable to process deployment. Greengrass launch directory is not set up or Greengrass is not set up as a system service.](https://docs.aws.amazon.com/greengrass/v2/developerguide/exceptions.html#com.amazonaws.greengrass.deployment.exceptions.DeploymentException:UnableToProcessDeployment)

Erreur du serveur

ERREUR_SERVEUR

Cette erreur peut se produire lorsqu'une opération de AWS service renvoie une erreur 500 car le service ne peut pas traiter la demande pour le moment. Réessayez le déploiement ultérieurement.

S3_SERVER_ERROR

Cette erreur peut se produire lorsqu'une opération Amazon S3 renvoie une erreur 500. Consultez tous les codes d'erreur ou journaux supplémentaires pour plus de détails.

Erreur du service cloud

RESOLVE_COMPONENT_CANDIDATES_BAD_RESPONSE

Cette erreur peut se produire lorsque le service cloud Greengrass envoie une réponse incompatible à l'ResolveComponentCandidatesopération. Consultez le journal pour voir la cause de l'erreur, puis consultez la page de mise à jour du logiciel Nucleus pour voir si le problème a été corrigé dans une version ultérieure du Nucleus, ou contactez Support.

TAILLE DU DOCUMENT DE DÉPLOIEMENT DÉPASSÉE

Cette erreur peut se produire lorsque le document de déploiement demandé a dépassé le quota de taille maximale. Réduisez la taille du document de déploiement et réessayez le déploiement.

GREENGRASS_ARTIFACT_SIZE_INTROUVABLE

Cette erreur peut se produire lorsque Greengrass ne parvient pas à obtenir la taille d'un artefact de composant public. Consultez le journal pour voir la cause de l'erreur, puis consultez la page de mise à jour du logiciel Nucleus pour voir si le problème a été corrigé dans une version ultérieure du Nucleus, ou contactez Support.

DOCUMENT_DE DÉPLOIEMENT NON VALIDE

Cette erreur peut se produire lorsque l'appareil reçoit un document de déploiement non valide. Vérifiez tout code d'erreur supplémentaire ou consultez le journal pour déterminer la cause de l'erreur.

DEMANDE_DE DÉPLOIEMENT VIDE

Cette erreur peut se produire lorsqu'un appareil reçoit une demande de déploiement vide. Consultez le journal pour voir la cause de l'erreur, puis consultez la page de mise à jour du logiciel Nucleus pour voir si le problème a été corrigé dans une version ultérieure du Nucleus, ou contactez Support.

ERREUR D'ANALYSE DU DOCUMENT DE DÉPLOIEMENT

Cette erreur peut se produire lorsque le format de demande de déploiement ne correspond pas au format attendu. Consultez le journal pour voir la cause de l'erreur, puis consultez la page de

mise à jour du logiciel Nucleus pour voir si le problème a été corrigé dans une version ultérieure du Nucleus, ou contactez Support.

LES MÉTADONNÉES DES COMPOSANTS NE SONT PAS VALIDES LORS DU DÉPLOIEMENT

Cette erreur peut se produire lorsque la demande de déploiement contient des métadonnées de composant non valides. Consultez le journal pour voir la cause de l'erreur, puis consultez la page de mise à jour du logiciel Nucleus pour voir si le problème a été corrigé dans une version ultérieure du Nucleus, ou contactez Support.

Erreurs génériques

Aucun type d'erreur n'est associé à ces erreurs génériques.

DÉPLOIEMENT INTERROMPU

Cette erreur peut s'afficher lorsqu'un déploiement ne peut pas être terminé en raison d'un arrêt du noyau ou d'un autre événement externe. Consultez tous les codes d'erreur ou journaux supplémentaires pour plus de détails.

ERREUR DE TÉLÉCHARGEMENT D'ARTÉFACT

Cette erreur peut s'afficher en cas de problème lors du téléchargement d'un artefact. Consultez tous les codes d'erreur ou journaux supplémentaires pour plus de détails.

AUCUNE VERSION DE COMPOSANT DISPONIBLE

Cette erreur peut se produire lorsqu'une version de composant n'existe pas dans le cloud ou localement, ou en cas de conflit de résolution de dépendance. Consultez tous les codes d'erreur ou journaux supplémentaires pour plus de détails.

COMPONENT_PACKAGE_LOADING_ERROR

Cette erreur peut se produire lors d'une erreur lors du traitement des artefacts téléchargés. Consultez tous les codes d'erreur ou journaux supplémentaires pour plus de détails.

NUAGE_API_ERROR

Cette erreur peut se produire lorsqu'une erreur s'est produite lors de l'appel d'une API AWS de service. Consultez tous les codes d'erreur ou journaux supplémentaires pour plus de détails.

IO_ERROR

Cette erreur peut se produire lorsqu'une erreur d'E/S se produit lors d'un déploiement. Consultez tous les codes d'erreur ou journaux supplémentaires pour plus de détails.

ERREUR DE MISE À JOUR DU COMPOSANT

Cette erreur peut se produire lorsqu'un composant n'est pas mis à jour lors d'un déploiement. Vérifiez tout code d'erreur supplémentaire ou consultez le journal pour déterminer la cause de l'erreur.

Erreur inconnue

ÉCHEC DU DÉPLOIEMENT

Cette erreur peut se produire lorsqu'un déploiement échoue parce qu'une exception non contrôlée a été déclenchée. Consultez le journal pour voir la cause de l'erreur, puis consultez la page de mise à jour du logiciel Nucleus pour voir si le problème a été corrigé dans une version ultérieure du Nucleus, ou contactez Support.

TYPE_DE_DÉPLOIEMENT_NON VALIDE

Cette erreur peut s'afficher lorsque le type de déploiement n'est pas valide. Consultez le journal pour voir la cause de l'erreur, puis consultez la page de mise à jour du logiciel Nucleus pour voir si le problème a été corrigé dans une version ultérieure du Nucleus, ou contactez Support.

Codes d'état détaillés des composants

Utilisez les codes d'état et les solutions présentés dans ces sections pour résoudre les problèmes liés aux composants lors de l'utilisation de Greengrass nucleus version 2.8.0 ou ultérieure.

La plupart des statuts présentés dans cette rubrique indiquent des informations supplémentaires dans les journaux de AWS IoT Greengrass base. Ces journaux sont stockés dans le système de fichiers local de l'appareil principal. Il existe des journaux pour chaque composant individuel. Pour plus d'informations sur l'accès aux journaux, consultez [Accéder aux journaux du système de fichiers](#).

ERREUR_INSTALLATION

Cela peut se produire lorsqu'une erreur se produit lors de l'exécution d'un script d'installation. Le code d'erreur est indiqué dans le journal des composants. Vérifiez la présence d'erreurs dans le script d'installation et déployez à nouveau votre composant.

INSTALL_CONFIG_NOT_VALID

Cette erreur peut se produire lorsque l'installation d'un composant n'a pas pu être terminée car la `install` section de la recette n'est pas valide. Vérifiez la présence d'erreurs dans la section d'installation de votre recette et réessayez le déploiement.

ERREUR_INSTALL_IO_ERROR

Cela peut se produire lorsqu'une erreur d'E/S s'est produite lors de l'installation d'un composant. Consultez le journal des erreurs du composant pour plus de détails sur l'erreur.

INSTALL_MISSING_DEFAULT_RUNWITH

Cette erreur peut s'afficher si vous ne AWS IoT Greengrass parvenez pas à déterminer l'utilisateur ou le groupe à utiliser lors de l'installation d'un composant. Vérifiez que la `runWith` section de votre recette d'installation inclut un utilisateur ou un groupe valide.

TIMEOUT D'INSTALLATION

Cette erreur peut se produire lorsque le script d'installation ne s'est pas terminé dans le délai d'expiration configuré. Augmentez la `Timeout` période spécifiée dans la `install` section de la recette ou modifiez votre script d'installation pour qu'il se termine dans le délai défini.

ERREUR_DE DÉMARRAGE

Cela peut se produire lorsqu'une erreur se produit lors de l'exécution d'un script de démarrage. Le code d'erreur est indiqué dans le journal des composants. Vérifiez la présence d'erreurs dans le script d'installation et déployez à nouveau votre composant.

START_CONFIG_NOT_VALID

Cette erreur peut se produire lorsque l'installation d'un composant n'a pas pu être terminée car la `startup` section de la recette n'est pas valide. Vérifiez la présence d'erreurs dans la section de démarrage de votre recette et réessayez le déploiement.

STARTUP_IO_ERROR

Cela peut se produire lorsqu'une erreur d'E/S s'est produite lors du démarrage d'un composant. Consultez le journal des erreurs du composant pour plus de détails sur l'erreur.

START-UP_MISSING_DEFAULT_RUN WITH

Cette erreur peut se produire si vous ne AWS IoT Greengrass parvenez pas à déterminer l'utilisateur ou le groupe à utiliser lors de l'exécution d'un composant. Vérifiez que la `runWith` section de votre recette de démarrage inclut un utilisateur ou un groupe valide.

TIMEOUT DE DÉMARRAGE

Cette erreur peut se produire lorsque le script de démarrage ne s'est pas terminé dans le délai d'expiration configuré. Augmentez la Timeout période spécifiée dans la `startup` section de la recette ou modifiez votre script de démarrage pour qu'il se termine dans le délai défini.

ERREUR D'EXÉCUTION

Cela peut se produire lorsqu'une erreur se produit lors de l'exécution d'un script de composant. Le code d'erreur est indiqué dans le journal des composants. Vérifiez la présence d'erreurs dans le script d'exécution et déployez à nouveau votre composant.

RUN_MISSING_DEFAULT_RUNWITH

Cette erreur peut se produire si vous ne AWS IoT Greengrass parvenez pas à déterminer l'utilisateur ou le groupe à utiliser lors de l'exécution d'un composant. Vérifiez que la `runWith` section de votre recette d'exécution inclut un utilisateur ou un groupe valide.

RUN_CONFIG_NOT_VALID

Cette erreur peut se produire lorsqu'un composant n'a pas pu être exécuté car la `run` section de la recette n'est pas valide. Vérifiez la présence d'erreurs dans la section Exécuter de votre recette et réessayez le déploiement.

ERREUR RUN_IO_ERROR

Cela peut se produire lorsqu'une erreur d'E/S s'est produite pendant l'exécution du composant. Consultez le journal des erreurs du composant pour plus de détails sur l'erreur.

RUN_TIMEOUT

Cette erreur peut se produire lorsque le script d'exécution ne s'est pas terminé dans le délai d'expiration configuré. Augmentez la Timeout période spécifiée dans la `run` section de la recette ou modifiez votre script d'exécution pour qu'il se termine dans le délai défini.

ERREUR_ARRÊT

Cela peut se produire lorsqu'une erreur se produit lors de l'arrêt d'un script de composant. Le code d'erreur est indiqué dans le journal des composants. Vérifiez la présence d'erreurs dans le script d'arrêt et déployez à nouveau votre composant.

SHUTDOWN_TIMEOUT

Cette erreur peut se produire lorsque le script d'arrêt ne s'est pas terminé dans le délai d'expiration configuré. Augmentez la Timeout période spécifiée dans la `shutdown` section de la recette ou modifiez votre script d'exécution pour qu'il se termine dans le délai défini.

Marquez vos AWS IoT Greengrass Version 2 ressources

Avec les balises, vous pouvez organiser et gérer vos ressources dans AWS IoT Greengrass. Vous pouvez utiliser des balises pour attribuer des métadonnées à vos ressources, et vous pouvez utiliser des balises dans les politiques IAM pour définir un accès conditionnel à vos ressources.

Note

Actuellement, les balises de ressources Greengrass ne sont pas prises en charge pour les groupes AWS IoT de facturation ou les rapports de répartition des coûts.

Utilisation de balises dans AWS IoT Greengrass V2

Vous pouvez utiliser des balises pour classer vos AWS IoT Greengrass ressources par objectif, propriétaire, environnement ou toute autre classification adaptée à votre cas d'utilisation. Lorsque vous disposez de nombreuses ressources du même type, les balises vous aident à identifier plus facilement une ressource spécifique.

Chaque balise est constituée d'une clé et d'une valeur facultative que vous définissez. Par exemple, vous pouvez définir un ensemble de balises pour vos principaux appareils afin de vous aider à les suivre par les clients propriétaires de ces appareils. Nous vous recommandons de créer un ensemble de clés de balise répondant à vos besoins pour chaque type de ressource. En utilisant un ensemble cohérent de clés de balise, vous pouvez gérer plus facilement vos ressources.

Marquez avec le AWS Management Console

L'éditeur de balises AWS Management Console fournit un moyen centralisé et unifié de créer et de gérer vos balises pour les ressources de tous les AWS services. Pour de plus amples informations, veuillez consulter [Éditeur de balises](#) dans le Guide de l'utilisateur Groupes de ressources AWS .

Marquer avec l' AWS IoT Greengrass V2 API

Vous pouvez également utiliser l' AWS IoT Greengrass V2 API pour travailler avec des balises. 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 Références générales AWS.

- Pour ajouter des balises lorsque vous créez une ressource, définissez-les dans la propriété `tags` de la ressource.
- Pour ajouter des balises à une ressource existante ou pour mettre à jour les valeurs des balises, utilisez l'[TagResource](#) opération.
- Pour supprimer des balises d'une ressource, utilisez l'[UntagResource](#) opération.
- Pour récupérer les balises associées à une ressource, utilisez l'[ListTagsForResource](#) opération ou décrivez la ressource et inspectez ses tags propriétés.

Le tableau suivant répertorie les ressources que vous pouvez baliser à l'aide de l' AWS IoT Greengrass V2 API et les Get opérations `Describe` et/ou ou correspondantes `Create`.

Ressources taguables AWS IoT Greengrass V2

Ressource	Opération de création	Décrire ou obtenir le fonctionnement
Appareil principal	Aucune. Exécutez le logiciel AWS IoT Greengrass Core sur un appareil pour créer un appareil principal.	GetCoreDevice
Composant	CreateComponentVersion	DescribeComponent , GetComponent
Déploiement	CreateDeployment	GetDeployment

Utilisez les opérations suivantes afin d'afficher et de gérer des balises pour les ressources qui prennent en charge le balisage :

- [TagResource](#)— Ajoute des balises à une ressource ou met à jour la valeur d'une balise existante.
- [ListTagsForResource](#)— Répertorie les balises d'une ressource.
- [UntagResource](#)— Supprime les balises d'une ressource.

Vous pouvez ajouter ou supprimer des balises pour 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 la valeur précédente. 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.

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.

Note

Si vous utilisez des balises pour autoriser ou refuser l'accès des utilisateurs aux ressources, vous devez refuser aux utilisateurs la possibilité d'ajouter ou de supprimer ces balises pour les mêmes ressources. Sinon, il sera possible pour un utilisateur de contourner vos restrictions et d'obtenir l'accès à une ressource en modifiant ses balises.

Vous pouvez utiliser les clés et valeurs de contexte de condition suivantes dans l'Conditionélément, également appelé Condition bloc, d'une déclaration de politique.

`greengrassv2: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`

Exigez qu'une balise spécifique soit utilisée ou non lors de la création ou de la modification d'une ressource balisable.

`aws:TagKeys: [tag-key, ...]`

Exigez qu'un ensemble spécifique de clés de balise soit utilisé, ou non, lors de la création ou de la modification d'une ressource balisable.

 Note

Les clés et valeurs de contexte de condition d'une politique IAM s'appliquent uniquement aux actions dont le paramètre obligatoire est une ressource balisable. Par exemple, vous pouvez définir un accès conditionnel basé sur des balises pour [GetComponent](#).

Pour plus d'informations, consultez la section [Contrôle de l'accès aux AWS ressources à l'aide de balises de ressources](#) et la [référence de politique IAM JSON](#) dans le guide de l'utilisateur IAM.

Création de AWS IoT Greengrass ressources avec AWS CloudFormation

AWS IoT Greengrass est intégré à AWS CloudFormation un service qui vous aide à modéliser et à configurer vos AWS ressources afin que vous puissiez passer moins de temps à créer et à gérer vos ressources et votre infrastructure. Vous créez un modèle qui décrit toutes les AWS ressources souhaitées (telles que les versions des composants et les déploiements), puis vous CloudFormation approvisionnez et configurez ces ressources pour vous.

Lorsque vous l'utilisez CloudFormation, vous pouvez réutiliser votre modèle pour configurer vos AWS IoT Greengrass ressources de manière cohérente et répétée. Décrivez vos ressources une seule fois, puis fournissez les mêmes ressources encore et encore dans plusieurs Comptes AWS régions.

AWS IoT Greengrass et CloudFormation modèles

Pour fournir et configurer des ressources AWS IoT Greengrass et des services associés, vous devez comprendre les [CloudFormation modèles](#). Les modèles sont des fichiers texte formatés en JSON ou YAML. Ces modèles décrivent les ressources que vous souhaitez mettre à disposition dans vos CloudFormation piles. Si vous n'êtes pas familiarisé avec JSON ou YAML, vous pouvez utiliser CloudFormation Designer pour vous aider à démarrer avec les CloudFormation modèles. Pour plus d'informations, consultez [Qu'est-ce que CloudFormation Designer ?](#) dans le AWS CloudFormation Guide de l'utilisateur.

AWS IoT Greengrass prend en charge la création de versions de composants et de déploiements dans. CloudFormation Pour plus d'informations, notamment des exemples de modèles JSON et YAML pour les versions et les déploiements de composants, consultez la [référence au type de AWS IoT Greengrass ressource](#) dans le guide de l'AWS CloudFormation utilisateur.

ComponentVersion exemple de modèle

Voici le modèle YAML pour une version d'un composant simple. La recette JSON inclut des sauts de ligne pour plus de lisibilité.

```
Parameters:
  ComponentVersion:
    Type: String
Resources:
```

```

TestSimpleComponentVersion:
  Type: AWS::GreengrassV2::ComponentVersion
  Properties:
    InlineRecipe: !Sub
      - "{\n
        \n\"RecipeFormatVersion\": \"2020-01-25\", \n
        \n\"ComponentName\": \"component1\", \n
        \n\"ComponentVersion\": \"${ComponentVersion}\", \n
        \n\"ComponentType\": \"aws.greengrass.generic\", \n
        \n\"ComponentDescription\": \"This\", \n
        \n\"ComponentPublisher\": \"You\", \n
        \n\"Manifests\": [\n
        {\n
          \n\"Platform\": {\n
            \n\"os\": \"darwin\" \n
          }, \n
          \n\"Lifecycle\": {}, \n
          \n\"Artifacts\": [] \n
        }, \n
        {\n
          \n\"Lifecycle\": {}, \n
          \n\"Artifacts\": [] \n
        } \n
      ], \n
      \n\"Lifecycle\": {\n
      \n\"install\": {\n
        \n\"script\": \"yuminstallpython\" \n
      } \n
    } \n
  }"
    - { ComponentVersion: !Ref ComponentVersion }

```

Exemple de modèle de déploiement

Ce qui suit est un fichier YAML définissant un modèle simple pour un déploiement.

```

Parameters:
  ComponentVersion:
    Type: String
  TargetArn:
    Type: String
Resources:
  TestDeployment:

```

```
Type: AWS::GreengrassV2::Deployment
Properties:
  Components:
    component1:
      ComponentVersion: !Ref ComponentVersion
  TargetArn: !Ref TargetArn
  DeploymentName: CloudFormationIntegrationTest
  DeploymentPolicies:
    FailureHandlingPolicy: DO_NOTHING
    ComponentUpdatePolicy:
      TimeoutInSeconds: 5000
      Action: SKIP_NOTIFY_COMPONENTS
    ConfigurationValidationPolicy:
      TimeoutInSeconds: 30000
Outputs:
  TestDeploymentArn:
    Value: !Sub
      - arn:${AWS::Partition}:greengrass:${AWS::Region}:${AWS::AccountId}:deployments:
        ${DeploymentId}
      - DeploymentId: !GetAtt TestDeployment.DeploymentId
```

En savoir plus sur CloudFormation

Pour en savoir plus CloudFormation, consultez les ressources suivantes :

- [AWS CloudFormation](#)
- [AWS CloudFormation Guide de l'utilisateur](#)
- [CloudFormation API Reference](#)
- [Guide de l'utilisateur de l'interface de ligne de commande AWS CloudFormation](#)

Logiciel de AWS IoT Greengrass base open source

Le AWS IoT Greengrass Version 2 Edge Runtime (Nucleus) et les autres composants du logiciel AWS IoT Greengrass Core sont open source. Cela signifie que vous pouvez consulter le code pour résoudre les problèmes liés aux interactions avec vos applications. Vous pouvez également personnaliser et étendre le logiciel AWS IoT Greengrass Core pour répondre à vos besoins spécifiques en matière de logiciels et de matériel.

Pour plus d'informations sur les référentiels open source pour le logiciel AWS IoT Greengrass Core, consultez l'organisation [aws-greengrass](#) sur GitHub. Votre utilisation de logiciels open source est régie par la licence open source du [GitHub référentiel correspondant](#).

Votre utilisation du logiciel AWS IoT Greengrass principal et des composants non soumis à une licence open source est régie par la licence [logicielle AWS Greengrass Core](#).

Historique du document pour le guide du AWS IoT Greengrass V2 développeur

Le tableau suivant décrit la documentation de cette version de AWS IoT Greengrass Version 2.

- Version de l'API : 2020-11-30

Modification	Description	Date
Sortie du composant Greengrass Nucleus Lite v2.3.2	La version 2.3.2 du composant Greengrass nucleus lite est disponible. Cette version met à jour le fichier de version pour un rapport correct des versions de Nucleus.	12 février 2026
Sortie du composant Greengrass Nucleus Lite v2.3.1	La version 2.3.1 du composant Greengrass nucleus lite est disponible. Cette version inclut plusieurs corrections de bogues et améliorations.	11 février 2026
Greengrass CLI v2.16.1 est sortie	Le composant Greengrass CLI v2.16.1 est disponible.	23 décembre 2025
AWS IoT Greengrass Mise à jour logicielle Core v2.16.1	Cette version fournit la version 2.16.1 du composant Greengrass nucleus et met à jour les composants fournis. AWS	23 décembre 2025
Sortie du redirecteur de journal du système v2.1.0	Le redirecteur de journal système v2.1.0 est disponible.	6 novembre 2025
Shadow Manager v2.3.12 est sorti	Shadow Manager v2.3.12 est disponible.	6 novembre 2025

Lancement du gestionnaire de journaux v2.3.11	Le composant Log Manager v2.3.11 est disponible.	6 novembre 2025
Sortie du composant Greengrass Nucleus Lite v2.3.0	La version 2.3.0 du composant Greengrass nucleus lite est disponible. Cette version ajoute la prise en charge de TPM 2.0, d'une gamme plus large de systèmes d'exploitation et d' RestartComponent IPC. Cette version inclut également plusieurs autres corrections de bogues et améliorations.	6 novembre 2025
Greengrass CLI v2.16.0 est sortie	Le composant Greengrass CLI v2.16.0 est disponible.	6 novembre 2025
AWS IoT Greengrass Mise à jour logicielle Core v2.16.0	Cette version fournit la version 2.16.0 du composant Greengrass Nucleus et met à jour les composants fournis. AWS	6 novembre 2025
AWS IoT Greengrass Mise à jour logicielle Core v2.15.1	Cette version fournit la version 2.15.1 du composant Greengrass Nucleus et met à jour les composants fournis. AWS	6 novembre 2025
Sortie du redirecteur de journal du système v2.0.1	Le redirecteur de journal système v2.0.1 est disponible. Cette version met à jour la recette des composants pour prendre en charge correctement les systèmes aarch64 (arm64).	2 octobre 2025

Sortie du composant Greengrass Nucleus Lite v2.2.2	La version 2.2.2 du composant Greengrass nucleus lite est disponible. Cette version inclut plusieurs corrections de bogues et améliorations.	18 août 2025
Sortie du redirecteur de journal du système v2.0.0	Le composant System Log Forwarder v2.0.0 est disponible. Version initiale publiée.	25 juillet 2025
Sortie du composant Greengrass Nucleus Lite v2.2.1	La version 2.2.1 du composant Greengrass nucleus lite est disponible. Cette version corrige un problème selon lequel le noyau ne parvient pas à obtenir les informations d'identification TES.	25 juillet 2025
Sortie du composant Greengrass Nucleus Lite v2.2.0	La version 2.2.0 du composant Greengrass nucleus lite est disponible. Cette version ajoute la prise en charge des artefacts URIs d'image de conteneur.	3 juillet 2025
AWS IoT Greengrass Mise à jour logicielle Core v2.15.0	Cette version fournit la version 2.15.0 du composant Greengrass Nucleus et met à jour les composants fournis. AWS	3 juillet 2025

[Sortie de Secure Tunneling v1.1.3](#)

Secure Tunneling v1.1.3 est disponible. Cette version met à niveau le client de AWS IoT périphérique sous-jacent invoqué par le composant de la version 1.10.0 à la version 1.10.1. Cette version corrige également les problèmes de compatibilité de la bibliothèque GNU C (glibc) dans la version 1.1.2 du composant de tunneling sécurisé.

16 mai 2025

[Sortie du composant Greengrass Nucleus Lite v2.1.0](#)

La version 2.1.0 du composant Greengrass nucleus lite est disponible. Cette version ajoute le support du proxy HTTP. Cette version inclut également plusieurs autres corrections de bogues et améliorations.

5 mai 2025

[Secret Manager v2.2.5 est sorti](#)

Secret Manager v2.2.5 est disponible. Cette version corrige un problème selon lequel aucun secret n'est extrait du cache local AWS Cloud s'il n'est pas présent dans le cache local.

16 avril 2025

[AWS IoT Greengrass Mise à jour logicielle Core v2.14.3](#)

Cette version fournit la version 2.14.3 du composant Greengrass Nucleus et met à jour les composants fournis. AWS

11 avril 2025

[Secret Manager v2.2.4 est sorti](#)

Secret Manager v2.2.4 est disponible. Cette version réduit la fréquence des écritures dans le magasin secret local. Le gestionnaire de secrets écrit désormais dans le magasin local uniquement lorsque les secrets sont mis à jour.

8 avril 2025

[Greengrass CLI v2.14.3 est sortie](#)

Le composant Greengrass CLI v2.14.3 est disponible.

8 avril 2025

[Sortie de la version v1.1.2 de Secure Tunneling](#)

Le tunneling sécurisé v1.1.2 est disponible. Cette version met à niveau le client de AWS IoT périphérique sous-jacent invoqué par le composant de la version 1.9.0 à la version 1.10.0. Le tunneling sécurisé v1.1.2 résout également le problème de transfert de charge utile qui empêche les utilisateurs de transférer des fichiers volumineux des appareils principaux de Greengrass V2 vers le périphérique source via le tunnel sécurisé.

27 mars 2025

[Greengrass CLI v2.14.2 est sortie](#)

Le composant Greengrass CLI v2.14.2 est disponible.

27 mars 2025

[AWS IoT Greengrass Mise à jour logicielle Core v2.14.2](#)

Cette version fournit la version 2.14.2 du composant Greengrass Nucleus et met à jour les composants fournis. AWS

27 mars 2025

[Secret Manager v2.2.3 est sorti](#)

Secret Manager v2.2.3 est disponible. Cette version corrige un problème selon lequel le gestionnaire de secrets efface les secrets persistants localement lorsque le périphérique principal est hors ligne et que le service de sécurité de l'appareil (tel qu'un HSM) n'est pas disponible.

18 mars 2025

[Le gestionnaire de flux v2.2.1 est sorti](#)

Stream Manager v2.2.1 est désormais disponible. Cette version résout un problème en raison duquel le gestionnaire de flux ne parvient pas à exporter les messages vers les destinations Kinesis. Cette version améliore également les performances des exportations du gestionnaire de flux vers les destinations Kinesis.

12 mars 2025

[Le composant d'authentification du périphérique client v2.5.3 est sorti](#)

Le composant d'authentification du périphérique client v2.5.3 est disponible. Cette version résout un problème selon lequel les appareils clients ne peuvent pas se connecter au périphérique principal en raison de certificats clients périmés.

12 mars 2025

[AWS IoT Le plugin de provisionnement de flotte v1.2.2 est sorti](#)

AWS IoT le plugin de provisionnement de flotte v1.2.2 est disponible. Cette version ajoute la prise en charge des chemins personnalisés de certificat de périphérique principal (certificatePath) et de clé privée (privateKeyPath).

12 mars 2025

[Sortie de la version v1.1.1 de Secure Tunneling](#)

Secure Tunneling v1.1.1 est disponible. Cette version ajoute une configuration pour prendre en charge Greengrass nucleus lite.

7 février 2025

[Greengrass CLI v2.14.1 est sortie](#)

Le composant Greengrass CLI v2.14.1 est disponible.

7 février 2025

[AWS IoT Greengrass Mise à jour logicielle Core v2.14.1](#)

Cette version fournit la version 2.14.1 du composant Greengrass Nucleus et met à jour les composants fournis. AWS

7 février 2025

Sortie du composant Greengrass Nucleus Lite v2.0.2	La version 2.0.2 du composant Greengrass nucleus lite est disponible. Cette version inclut des corrections de bogues et des améliorations générales.	6 février 2025
Sortie de la version 2.0.1 du composant Greengrass Nucleus Lite	La version 2.0.1 du composant Greengrass nucleus lite est disponible. Cette version inclut des corrections de bogues et des améliorations générales.	28 janvier 2025
Greengrass CLI v2.14.0 est sortie	Le composant Greengrass CLI v2.14.0 est disponible.	24 décembre 2024
Le gestionnaire de flux v2.2.0 est sorti	Stream Manager v2.2.0 est désormais disponible.	16 décembre 2024
Shadow Manager v2.3.10 est sorti	Shadow Manager v2.3.10 est disponible.	16 décembre 2024
Sortie de la version v1.1.0 de Secure Tunneling	Secure Tunneling v1.1.0 est disponible. Cette version ajoute des supports de recettes pour Greengrass Nucleus Lite.	16 décembre 2024
Nouveau composant émetteur de télémétrie Nucleus	La version 1.0.10 du composant émetteur de télémétrie Nucleus est disponible.	16 décembre 2024
Sortie de l'adaptateur de protocole Modbus-RTU v2.1.10	Le composant adaptateur de protocole Modbus-RTU v2.1.10 est disponible.	16 décembre 2024
Lancement du gestionnaire de journaux v2.3.9	Le composant Log Manager v2.3.9 est disponible.	16 décembre 2024

Sortie de la console de débogage locale v2.4.4	Le composant de console de débogage local v2.4.4 est disponible. Cette version inclut des corrections de bogues et des améliorations générales.	16 décembre 2024
Lambda Manager v2.3.5 est sorti	Le composant Lambda Manager v2.3.5 est disponible.	16 décembre 2024
Sortie du détecteur IP v2.2.1	Le composant de détection IP v2.2.1 est disponible.	16 décembre 2024
Sortie du composant Greengrass Nucleus Lite v2.0.0	La version 2.0.0 du composant Greengrass nucleus lite est disponible. Il s'agit de la version initiale.	16 décembre 2024
Disk Spooler v1.0.5 est sorti	Le composant Disk Spooler v1.0.5 est disponible.	16 décembre 2024
Le composant d'authentification du périphérique client v2.5.2 est sorti	Le composant d'authentification du périphérique client v2.5.2 est disponible.	16 décembre 2024

[AWS IoT Greengrass Mise à jour logicielle Core v2.14.0](#)

16 décembre 2024

Cette version fournit la version 2.14.0 du composant Greengrass Nucleus, ainsi que de nouvelles AWS IoT Greengrass mises à jour de Nucleus Lite. Le AWS IoT Greengrass nucleus lite est un nouveau runtime, disponible pour AWS IoT Greengrass la version 2. Il offre une alternative à l'encombrement mémoire réduit. C'est une bonne option pour les appareils aux ressources limitées. Il implémente un sous-ensemble des fonctionnalités du noyau avec une compatibilité accrue prévue pour les futures versions. Le code source est désormais disponible sur [Github](#). Avec le runtime Nucleus Lite, vous pouvez :

- Déployez des composants sur les appareils principaux de Greengrass. Utilisez le même format de recette, même si certaines fonctionnalités avancées ne sont peut-être pas encore disponibles.
- Les applications déployées en tant que composants Greengrass peuvent utiliser l'appareil SDKs pour accéder aux IPC Greengrass

s pris en charge APIs, tels que l'accès AWS IoT Core MQTT, le pub/sub local et l'accès à la configuration Greengrass. Consultez le tableau de compatibilité pour obtenir la liste des [IPC APIs pris en charge](#).

- Certains composants AWS gérés ont été mis à jour pour le support de Nucleus Lite. Consultez les [composants AWS fournis](#) pour obtenir la liste des composants compatibles existants.

Nouvelles fonctions :

- Utilise moins de mémoire et d'espace disque (moins de 5 Mo de RAM et moins de 5 Mo de stockage).
- Les composants s'intègrent au gestionnaire de services du système hôte (systemd pour les plateformes Linux actuellement prises en charge).

Points à surveiller :

- AWS IoT Greengrass les recettes Nucleus Lite distinguent les majuscules et minuscules. Assurez-vous que le bon boîtier (clés) est

utilisé, comme indiqué dans la référence de la <https://docs.aws.amazon.com/greengrass/v2/developerguide/component-recipe-reference.html> recette.

- Le runtime Nucleus Lite prend en charge les déploiements par groupes d'objets, mais ne prend pas encore en charge le type de cible de déploiement (unique) d'appareils Core. Pour effectuer un déploiement sur un seul appareil Greengrass, utilisez un groupe d'objets contenant uniquement cet appareil.
- Le runtime Nucleus Lite utilise des ressources de mémoire limitées ; les fonctionnalités qui évoluent en fonction de l'utilisation sur le runtime classique peuvent échouer en raison du dépassement des ressources disponibles sur Lite. Cela inclut une limite actuelle sur un maximum de 50 abonnements MQTT à la fois, ainsi que des limites maximales sur la taille des fichiers de recettes et les déploiements. Certaines de ces limites sont configurables au moment de la

compilation si vous compilez vous-même le runtime Lite.

- Le runtime Nucleus Lite n'est pas fourni avec Java. Pour utiliser des composants nécessitant Java, le système doit avoir déjà installé Java, ou un composant peut être utilisé pour installer Java.
- Nous vous recommandons de compiler le runtime Nucleus Lite à partir des sources et d'utiliser votre propre version adaptée à votre système. Pour les systèmes Yocto, une couche est disponible pour intégrer le runtime Nucleus Lite dans l'image de votre système.
- Actuellement, le nucleus lite suppose un système Linux utilisant systemd, ou une image de conteneur utilisant systemd.
- Bien que vous puissiez gérer les conteneurs Docker à l'aide de scripts de recettes, les artefacts des conteneurs gérés par Greengrass ne sont pas encore disponibles.
- Le runtime Nucleus Lite ne prend pas encore en charge

les clés stockées dans un PKCS11 module. Si votre cas d'utilisation nécessite des clés stockées sur un élément sécurisé, le runtime classique peut actuellement prendre en charge ce cas d'utilisation. Pour éviter toute fuite des informations d'identification de votre appareil, assurez-vous que les appareils de production n'utilisent le chiffrement complet du disque.

Parallèlement à l'introduction de Nucleus Lite, nous publions également la version 2.14.0 de Nucleus. Cette mise à jour apporte des améliorations significatives au noyau Greengrass existant.

Principales fonctionnalités et améliorations :

- La nouvelle prise en charge des terminaux à double pile permet la communication IPv6 réseau.
- Résilience améliorée contre les échecs de redémarrage du noyau et la corruption de répertoires.
- Correction de fuites de mémoire lors des fermetures

s PubSub d'abonnements IPC.

Le gestionnaire de flux v2.1.13 est sorti	Stream Manager v2.1.13 est désormais disponible. Cette version ajoute la prise en charge du point de terminaison FIPS pour AWS IoT SiteWise	26 août 2024
Shadow Manager v2.3.9 est sorti	Shadow Manager v2.3.9 est disponible.	26 août 2024
Sortie de l'adaptateur de protocole Modbus-RTU v2.1.9	Le composant adaptateur de protocole Modbus-RTU v2.1.9 est disponible.	26 août 2024
Lancement du gestionnaire de journaux v2.3.8	Le composant Log Manager v2.3.8 est disponible.	26 août 2024
Sortie de la console de débogage locale v2.4.3	Le composant de console de débogage local v2.4.3 est disponible. Cette version inclut des corrections de bogues et des améliorations générales.	26 août 2024
Disk Spooler v1.0.4 est sorti	Le composant Disk Spooler v1.0.4 est disponible.	26 août 2024
AWS IoT Greengrass Mise à jour logicielle Core v2.13.0	Cette version fournit la version 2.13.0 du composant Greengrass nucleus.	26 août 2024
Nouveau composant émetteur de télémétrie Nucleus	La version 1.0.9 du composant émetteur de télémétrie Nucleus est disponible.	23 août 2024
Lambda Manager v2.3.4 est sorti	Le composant Lambda Manager v2.3.4 est disponible.	23 août 2024

Greengrass CLI v2.13.0 est sortie	Le composant Greengrass CLI v2.13.0 est disponible.	23 août 2024
Le composant d'authentification du périphérique client v2.5.1 est sorti	Le composant d'authentification du périphérique client v2.5.0 est disponible. Cette version ajoute la prise en charge du point de terminaison FIPS.	23 août 2024
Validation des recettes	Ajout d'une fonctionnalité de validation de recette qui validera une recette de composant lors de la création d'une version de composant.	15 août 2024
Sortie du détecteur IP v2.2.0	Le composant de détection IP v2.2.0 est disponible. Cette version ajoute la prise en charge de IPv6. Vous pouvez désormais l'utiliser IPv6 pour la messagerie locale.	29 juillet 2024
Shadow Manager v2.3.8 est sorti	Shadow Manager v2.3.8 est disponible. Cette version corrige un problème selon lequel le gestionnaire de fantômes crée une situation de blocage lors de la connexion du client MQTT.	5 juin 2024
Greengrass CLI v2.12.6 est sortie	Le composant Greengrass CLI v2.12.6 est disponible.	24 mai 2024

[AWS IoT Greengrass Mise à jour logicielle Core v2.12.6](#)

Cette version fournit la version 2.12.6 du composant Greengrass Nucleus et met à jour les composants fournis. AWS

24 mai 2024

[AWS IoT Device Tester v4.9.4 avec sortie de GGV2 Q v2.5.4](#)

La version 4.9.4 d'IDT pour AWS IoT Greengrass V2 est disponible. Cette version inclut la suite de qualification AWS IoT Greengrass V2 (GGV2Q) v2.5.4 et prend en charge les versions 2.12.0, 2.11.0, 2.10.0, 2.9.5 du noyau Greengrass.

3 mai 2024

[Sortie de la version 1.0.19 de Secure Tunneling](#)

Secure Tunneling v1.0.19 est disponible. Cette version met à niveau le client de AWS IoT périphérique sous-jacent invoqué par le composant de la version 1.8.0 à la version 1.9.0. Secure Tunneling v1.0.19 augmente la limite de tunnels simultanés à 20 tunnels au niveau des composants. Cette nouvelle version augmente également le délai d'expiration du AWS IoT Greengrass Core IPC de 3 secondes à 10 secondes.

1 mai 2024

[Sortie du connecteur Edge pour le composant Kinesis Video Streams v1.0.5](#)

La version 1.0.5 du connecteur Edge pour le composant Kinesis Video Streams est disponible. Cette version inclut des corrections de bogues et des améliorations générales.

29 avril 2024

Greengrass CLI v2.12.5 est sortie	Le composant Greengrass CLI v2.12.5 est disponible.	25 avril 2024
Le composant d'authentification du périphérique client v2.5.0 est sorti	Le composant d'authentification du périphérique client v2.5.0 est disponible. Cette version ajoute la prise en charge des variables de politique pour les noms d'objets. Cette version autorise également les ressources politiques avec des caractères génériques.	25 avril 2024
AWS IoT Greengrass Mise à jour logicielle Core v2.12.5	Cette version fournit la version 2.12.5 du composant Greengrass Nucleus et AWS met à jour les composants fournis.	25 avril 2024
AWS IoT Device Tester v4.9.3 avec sortie de GGV2 Q v2.5.3	La version 4.9.3 d'IDT pour AWS IoT Greengrass V2 est disponible. Cette version inclut la suite de qualification AWS IoT Greengrass V2 (GGV2Q) v2.5.3 et prend en charge les versions 2.12.0, 2.11.0, 2.10.0, 2.9.5 du noyau Greengrass.	5 avril 2024
Greengrass CLI v2.12.4 est sortie	Le composant Greengrass CLI v2.12.4 est disponible.	2 avril 2024
AWS IoT Greengrass Mise à jour logicielle Core v2.12.4	Cette version fournit la version 2.12.4 du composant Greengrass Nucleus et met à jour les composants fournis. AWS	2 avril 2024

[Shadow Manager v2.3.7 est sorti](#)

Shadow Manager v2.3.7 est disponible. Cette version corrige un problème selon lequel Shadow Manager enregistre régulièrement une `NullPointerException` erreur lors d'une synchronisation du Shadow Manager.

27 mars 2024

[Sortie du broker Moquette MQTT 3.1.1 v2.3.6](#)

Le composant de broker Moquette MQTT 3.1.1 v2.3.6 est disponible. Cette version inclut des corrections de bogues et des améliorations générales.

27 mars 2024

[Sortie de la console de débogage locale v2.4.2](#)

Le composant de console de débogage local v2.4.2 est disponible. Cette version inclut des corrections de bogues et des améliorations générales.

27 mars 2024

[Lambda Manager v2.3.3 est sorti](#)

Le composant Lambda Manager v2.3.3 est disponible. Cette version inclut des corrections de bogues et des améliorations générales.

27 mars 2024

[Sortie du détecteur IP v2.1.9](#)

Le composant de détection IP v2.1.9 est disponible. Cette version ajuste l'étape d'acquisition de l'adresse IP pour envoyer uniquement les journaux au niveau du journal de débogage.

27 mars 2024

[AWS IoT Le plugin de provisionnement de flotte v1.2.1 est sorti](#)

AWS IoT le plugin de provisionnement de flotte v1.2.1 est disponible. Cette version corrige un problème selon lequel le plugin de provisionnement de flotte est hors ligne lors du démarrage du noyau Greengrass. Le plugin de provisionnement de flotte réessaie désormais indéfiniment les appels MQTT connect.

[AWS IoT Greengrass Mise à jour logicielle Core v2.12.3](#)

Cette version fournit la version 2.12.3 du composant Greengrass Nucleus et met à jour les composants fournis. AWS

[Greengrass CLI v2.12.3 est sortie](#)

Le composant Greengrass CLI v2.12.3 est disponible.

[AWS IoT Device Tester v4.9.2 avec sortie de GGV2 Q v2.5.2](#)

La version 4.9.2 d'IDT pour AWS IoT Greengrass V2 est disponible. Cette version inclut la suite de qualification AWS IoT Greengrass V2 (GGV2Q) v2.5.2 et prend en charge les versions 2.12.0, 2.11.0, 2.10.0, 2.9.5 du noyau Greengrass.

[AWS IoT Greengrass Mise à jour logicielle Core v2.12.2](#)

Cette version fournit la version 2.12.2 du composant Greengrass Nucleus et met à jour les composants fournis. AWS

[Shadow Manager v2.3.6 est sorti](#)

Shadow Manager v2.3.6 est disponible. Cette version résout un problème selon lequel les propriétés d'ombre supprimées par le biais de AWS Cloud mises à jour alors que l'appareil est hors ligne continuent d'exister dans l'ombre locale une fois la connectivité rétablie.

14 février 2024

[Lambda Launcher v2.0.13 est sorti](#)

La version 2.0.13 du composant du lanceur Lambda est disponible. Cette version inclut des corrections de bogues et des améliorations générales.

14 février 2024

[Disk Spooler v1.0.3 est sorti](#)

Le composant Disk Spooler v1.0.3 est disponible. Cette version améliore les performances en réutilisant les connexions aux bases de données.

14 février 2024

[Kit de développement Greengrass CLI v1.6.2](#)

La version 1.6.2 de la CLI du kit de développement Greengrass est disponible. Cette version corrige un problème selon lequel Windows gradlew.bat ne fonctionne pas en raison du chemin relatif. Cette version contient également des améliorations supplémentaires.

16 janvier 2024

[Nouveaux événements CloudTrail liés aux données](#)

Vous pouvez désormais enregistrer AWS CloudTrail les événements de données pour obtenir des informations sur les opérations relatives aux ressources, telles que l'obtention d'un composant ou la configuration d'un déploiement. Utilisez ces événements pour avoir un aperçu du fonctionnement de vos appareils Greengrass.

20 décembre 2023

[Le gestionnaire de flux v2.1.12 est sorti](#)

Stream Manager v2.1.12 est désormais disponible. Cette version modifie l'ordre utilisé par Greengrass pour sélectionner un ensemble d'informations d'identification pour les appels de AWS service.

8 décembre 2023

[Sortie du pont MQTT v2.3.1](#)

Le pont MQTT v2.3.1 est disponible. Cette version corrige un problème rare où le client MQTT local entre dans une boucle de déconnexion.

8 décembre 2023

[Disk Spooler v1.0.2 est sorti](#)

Le composant Disk Spooler v1.0.2 est disponible. Cette version corrige un problème selon lequel le champ de format de message MQTT n'est pas conservé dans certains cas.

8 décembre 2023

[Le composant d'authentification du périphérique client v2.4.5 est sorti](#)

Le composant d'authentification du périphérique client v2.4.5 est disponible. Cette version ajoute la prise en charge des caractères génériques à la fin des noms d'objets dans une règle de sélection et résout le problème selon lequel les certificats ne sont pas mis à jour avec de nouvelles informations de connectivité dans certains cas.

8 décembre 2023

[AWS IoT Greengrass Mise à jour logicielle Core v2.12.1](#)

Cette version fournit la version 2.12.1 du composant Greengrass Nucleus et met à jour les composants fournis. AWS

8 décembre 2023

[Kit de développement Greengrass CLI v1.6.1](#)

La version 1.6.1 de la CLI du kit de développement Greengrass est disponible. Cette version contient des corrections de bogues et des améliorations.

6 décembre 2023

[Validation des recettes](#)

Ajout d'une fonctionnalité de validation de recette qui validera une recette de composant lors de la création d'une version de composant.

16 novembre 2023

[Composants pris en charge par l'éditeur](#)

AWS IoT Greengrass propose désormais des composants pris en charge par Publisher . Ces composants sont développés, proposés et entretenus par des fournisseurs tiers.

16 novembre 2023

[Greengrass Testing Framework v1.2.0 est sorti](#)

Greengrass Testing Framework v1.2.0 est disponible.

15 novembre 2023

[Kit de développement Greengrass CLI v1.6.0](#)

La version 1.6.0 de la CLI du kit de développement Greengrass est disponible. Cette version ajoute un contrôle de validation des recettes par rapport au schéma de recette Greengrass lors des commandes `component build` et `component publish`. Cette mise à jour aide les développeurs à identifier les problèmes réalisables dans leurs recettes de composants plus tôt dans le processus de création des composants. Cette version ajoute également une suite de tests de confiance au modèle qui peut être extraite par la `test-e2e init` commande. Cette suite de tests de confiance comprend huit tests génériques qui peuvent être utilisés et étendus pour répondre aux besoins de test des composants de base.

15 novembre 2023

[AWS IoT Device Tester La version 4.9.1 prend en charge la version 2.12.0 de Greengrass Nucleus](#)

La version 4.9.1 d'IDT pour AWS IoT Greengrass V2 prend désormais en charge la version 2.12.0 de Greengrass nucleus.

7 novembre 2023

[AWS IoT Greengrass Mise à jour logicielle Core v2.12.0](#)

Cette version fournit la version 2.12.0 du composant Greengrass Nucleus et met à jour les composants fournis. AWS

7 novembre 2023

[Utiliser un appareil principal Greengrass en VPC](#)

L'utilisation d'un appareil principal Greengrass en VPC est disponible. Cette fonctionnalité vous permet d'effectuer des déploiements en VPC sans accès public à Internet.

3 novembre 2023

[Greengrass CLI v2.12.0 est sortie](#)

Le composant Greengrass CLI v2.12.0 est disponible.

30 octobre 2023

[Le gestionnaire de flux v2.1.10 est sorti](#)

Stream Manager v2.1.10 est désormais disponible. Cette version corrige un problème selon lequel la configuration du proxy HTTPS ne fait pas confiance à la chaîne de certificats Greengrass CA.

26 octobre 2023

[Lambda Launcher v2.0.12 est sorti](#)

La version 2.0.12 du composant du lanceur Lambda est disponible. Cette version corrige un problème selon lequel le lanceur Lambda pouvait générer une erreur si le processus précédent n'était pas correctement arrêté.

26 octobre 2023

[Kit de développement Greengrass CLI v1.5.0](#)

La version 1.5.0 de la CLI du kit de développement Greengrass est disponible. Cette version met à jour les modèles reconnus par l'option de `excludes construction` quand `build_system` c'est le caszip. Cette version reconnaîtra désormais les modèles globulaires qui correspondent aux noms de chemin en fonction de leurs caractères génériques. Cela permet de spécifier de manière personnalisée les répertoires à exclure.

26 octobre 2023

[Shadow Manager v2.3.4 est sorti](#)

Shadow Manager v2.3.4 est disponible. Cette version ajoute la prise en charge des documents à état fictif nuls et vides.

18 octobre 2023

[Lancement du gestionnaire de journaux v2.3.6](#)

Le composant Log Manager v2.3.6 est disponible.

18 octobre 2023

[Sortie de la console de débogage locale v2.4.0](#)

Le composant de console de débogage local v2.4.0 est disponible.

18 octobre 2023

[Lambda Manager v2.3.1 est sorti](#)

Le composant Lambda Manager v2.3.1 est disponible.

18 octobre 2023

[Greengrass CLI v2.11.3 est sortie](#)

Le composant Greengrass CLI v2.11.3 est disponible.

18 octobre 2023

[AWS IoT Greengrass Mise à jour logicielle Core v2.11.3](#)

Cette version fournit la version 2.11.3 du composant Greengrass Nucleus et met à jour les composants fournis. AWS

18 octobre 2023

[Sortie de la version 1.0.17 de Secure Tunneling](#)

Secure Tunneling v1.0.17 est disponible.

4 octobre 2023

[Kit de développement Greengrass CLI v1.4.0](#)

La version 1.4.0 de la CLI du kit de développement Greengrass est disponible. Cette version ajoute une nouvelle config commande qui lance une invite interactive pour modifier les champs d'un fichier de configuration GDK existant. Cette version modifie également les gdk component publish commandes gdk component build et pour vérifier que la taille de la recette est conforme aux exigences de Greengrass ($\leq 16\,000$ octets) avant de continuer.

2 octobre 2023

[Sortie du broker Moquette MQTT 3.1.1 v2.3.5](#)

Le composant de broker Moquette MQTT 3.1.1 v2.3.5 est disponible. Cette version met à jour Moquette vers la version 0.17.

28 septembre 2023

Sortie du pont MQTT v2.3.0	Le pont MQTT v2.3.0 est disponible. Cette version ajoute la prise en charge de MQTT 5 pour le pont entre les sources MQTT AWS IoT Core et les sources MQTT locales.	28 septembre 2023
Lambda Manager v2.3.0 est sorti	Le composant Lambda Manager v2.3.0 est disponible.	15 septembre 2023
Lambda Launcher v2.0.11 est sorti	La version 2.0.11 du composant du lanceur Lambda est disponible. Cette version prend en charge Lambda Manager 2.3.0.	15 septembre 2023
Sortie du broker Moquette MQTT 3.1.1 v2.3.4	Le composant de broker Moquette MQTT 3.1.1 v2.3.4 est disponible.	1er septembre 2023
Cadre de test Greengrass	GTF est un ensemble d'éléments de base destinés à soutenir l' end-to-endautomatisation. Il permet aux clients AWS IoT Greengrass Version 2 internes d'utiliser le même cadre de test que celui utilisé par l'équipe de service à des fins de qualification des modifications logicielles, d'acceptation automatique et d'assurance qualité.	11 août 2023

[AWS IoT Greengrass Mise à jour logicielle Core v2.11.2](#)

Cette version fournit la version 2.11.2 du composant Greengrass Nucleus et met à jour les composants fournis. AWS

9 août 2023

[Kit de développement Greengrass CLI v1.3.0](#)

La version 1.3.0 de la CLI du kit de développement Greengrass est disponible. Cette version ajoute une nouvelle test-e2e commande pour prendre en charge le end-to-end test des composants à l'aide d'Open Test Framework.

21 juillet 2023

[AWS IoT Greengrass Mise à jour logicielle Core v2.11.1](#)

Cette version fournit la version 2.11.1 du composant Greengrass Nucleus et met à jour les composants fournis. AWS

21 juillet 2023

[Disk Spooler v1.0.0 est sorti](#)

Le composant Disk Spooler v1.0.0 est disponible.

28 juin 2023

[AWS IoT Greengrass Mise à jour logicielle Core v2.11.0](#)

Cette version fournit la version 2.11.0 du composant Greengrass Nucleus et met à jour les composants fournis. AWS

28 juin 2023

[AWS IoT Greengrass Mise à jour logicielle Core v2.10.3](#)

Cette version fournit la version 2.10.3 du composant Greengrass Nucleus et met à jour les composants fournis. AWS

21 juin 2023

<u>AWS IoT Greengrass Mise à jour logicielle Core v2.10.2</u>	Cette version fournit la version 2.10.2 du composant Greengrass Nucleus et AWS met à jour les composants fournis.	5 juin 2023
<u>AWS IoT Greengrass Mise à jour logicielle Core v2.10.1</u>	Cette version fournit la version 2.10.1 du composant Greengrass Nucleus et AWS met à jour les composants fournis.	11 mai 2023
<u>AWS IoT Greengrass Mise à jour logicielle Core v2.10.0</u>	Cette version fournit la version 2.10.0 du composant Greengrass Nucleus et met à jour les composants fournis. AWS	9 mai 2023
<u>SageMaker AI Edge Manager n'est plus disponible</u>	Le composant Amazon SageMaker AI Edge Manager ne sera plus disponible le 26 avril 2024.	28 avril 2023
<u>AWS IoT Greengrass Mise à jour logicielle Core v2.9.6</u>	Cette version fournit la version 2.9.6 du composant Greengrass Nucleus et met à jour les composants fournis. AWS	20 avril 2023
<u>Lancement du gestionnaire de journaux v2.3.2</u>	Le composant Log Manager v2.3.2 est disponible.	19 avril 2023

[Le gestionnaire de flux v2.1.4 est sorti](#)

Stream Manager v2.1.4 est désormais disponible. Cette version corrige un problème selon lequel les entrées pour le même actif immobilier avec le même horodatage au sein d'un même lot sont renvoyées par l' SiteWise API, ce qui oblige le gestionnaire `ConflictingOperationException` de flux à réessayer en permanence. Cette version met également à jour le délai de connexion par défaut de 3 secondes à 1 minute.

13 avril 2023

[Kit de développement Greengrass CLI v1.2.3](#)

La version 1.2.3 de la CLI du kit de développement Greengrass est disponible. Cette version contient des corrections de bogues.

13 avril 2023

[Le composant d'authentification du périphérique client v2.4.0 est sorti](#)

Le composant d'authentification du périphérique client v2.4.0 est disponible. Cette version ajoute la prise en charge de l'authentification des appareils clients pour émettre des métriques opérationnelles qui peuvent être affichées sur le tableau de bord des appareils clients Greengrass.

10 avril 2023

Kit de développement Greengrass CLI v1.2.2	La version 1.2.2 de la CLI du kit de développement Greengrass est disponible. Cette version contient des améliorations et des corrections de bugs.	7 avril 2023
AWS IoT Greengrass Mise à jour logicielle Core v2.9.5	Cette version fournit la version 2.9.5 du composant Greengrass Nucleus et AWS met à jour les composants fournis.	30 mars 2023
Le gestionnaire de flux v2.1.3 est sorti	Stream Manager v2.1.3 est désormais disponible. Cette version corrige un problème de démarrage sur le système d'exploitation Windows lors de l'exécution en tant qu'utilisateur du SYSTÈME.	7 mars 2023
Sortie de l'adaptateur de protocole Modbus-RTU v2.1.5	Le composant adaptateur de protocole Modbus-RTU v2.1.5 est disponible. Cette version corrige un problème lié au ReadDiscreteInput fonctionnement.	7 mars 2023

[Sortie du composant d'authentification du périphérique client v2.3.2](#)

Le composant d'authentification du périphérique client v2.3.2 est disponible. Cette version ajoute la prise en charge de la mise en cache des informations de nom d'hôte afin que le composant génère correctement les sujets de certificat lors du redémarrage en mode hors connexion.

7 mars 2023

[AWS IoT Device Tester La version 4.7.0 prend en charge la version 2.9.4 du noyau Greengrass](#)

La version 4.7.0 d'IDT pour AWS IoT Greengrass V2 prend désormais en charge la version 2.9.4 de Greengrass nucleus.

2 mars 2023

[L'interface de ligne de commande Greengrass v1.2.0 est sortie](#)

L'interface de ligne de commande Greengrass v1.2.0 est disponible.

28 février 2023

[AWS IoT Greengrass Mise à jour logicielle Core v2.9.4](#)

Cette version fournit la version 2.9.4 du composant Greengrass Nucleus et AWS met à jour les composants fournis.

24 février 2023

[Shadow Manager v2.3.1 est sorti](#)

Shadow Manager v2.3.1 est disponible. Cette version corrige un problème susceptible d'empêcher la synchronisation des mises à jour de Cloud Shadow. Cette version résout également un problème selon lequel les modifications apportées à la configuration de synchronisation des ombres nommées ne s'appliquent qu'à une seule ombre nommée.

21 février 2023

[AWS IoT Device Tester La version 4.7.0 prend en charge la version 2.9.3 de Greengrass Nucleus](#)

La version 4.7.0 d'IDT pour AWS IoT Greengrass V2 prend désormais en charge la version 2.9.3 de Greengrass nucleus.

9 février 2023

[Meilleures pratiques IAM mises à jour](#)

Mise à jour du guide s'aligner sur les bonnes pratiques IAM. Pour plus d'informations, consultez [Bonnes pratiques de sécurité dans IAM](#).

3 février 2023

[AWS IoT Greengrass Mise à jour logicielle Core v2.9.3](#)

Cette version fournit la version 2.9.3 du composant Greengrass Nucleus et AWS met à jour les composants fournis.

1er février 2023

[Lancement du gestionnaire de journaux v2.3.1](#)

Le gestionnaire de journaux v2.3.1 est disponible.

27 janvier 2023

[AWS IoT Device Tester La version 4.7.0 prend en charge la version 2.9.2 de Greengrass Nucleus](#)

La version 4.7.0 d'IDT pour AWS IoT Greengrass V2 prend désormais en charge la version 2.9.2 de Greengrass nucleus.

3 janvier 2023

[Shadow Manager v2.3.0 est sorti](#)

Shadow Manager v2.3.0 est disponible. Cette version corrige un problème susceptible d'empêcher les ombres de se synchroniser lorsqu'un appareil stocke la clé privée de l'appareil Greengrass dans un module de sécurité matériel.

29 décembre 2022

[AWS IoT Le plugin de provisionnement de flotte v1.2.0 est sorti](#)

AWS IoT le plugin de provisionnement de flotte v1.2.0 est disponible. Cette version ajoute la prise en charge du provisionnement des appareils via une demande de signature de certificat avec un chemin de clé privée configurable.

22 décembre 2022

[AWS IoT Greengrass Mise à jour logicielle Core v2.9.2](#)

Cette version fournit la version 2.9.2 du composant Greengrass Nucleus et AWS met à jour les composants fournis.

22 décembre 2022

[AWS IoT Device Tester v4.7.0 avec sortie de GGV2 Q v2.5.0](#)

La version 4.7.0 d'IDT pour AWS IoT Greengrass V2 est disponible. Cette version inclut la suite de qualification AWS IoT Greengrass V2 (GGV2Q) v2.5.0 et prend en charge les versions 2.9.1, 2.9.0, 2.8.1, 2.8.0, 2.7.0 et 2.6.0 du noyau Greengrass.

13 décembre 2022

[Shadow Manager v2.2.4 est sorti](#)

Résout un problème en raison duquel la validation de la taille de l'ombre n'était pas cohérente avec celle du cloud lors de la mise à jour du document fantôme local. Cela résout également un problème selon lequel le gestionnaire fantôme arrête d'écouter les mises à jour de configuration si un déploiement effectue une opération RESET sur les nœuds de configuration.

8 décembre 2022

[Lancement du gestionnaire de journaux v2.3.0](#)

Le composant Log Manager v2.3.0 est disponible.

18 novembre 2022

[AWS IoT Device Tester La version 4.5.11 prend en charge la version 2.9.1 du noyau Greengrass](#)

La version 4.5.11 d'IDT pour AWS IoT Greengrass V2 prend désormais en charge la version 2.9.1 du noyau Greengrass.

18 novembre 2022

[AWS IoT Greengrass Mise à jour logicielle Core v2.9.1](#)

Cette version fournit la version 2.9.1 du composant Greengrass Nucleus et AWS met à jour les composants fournis.

18 novembre 2022

[AWS IoT Device Tester La version 4.5.11 prend en charge la version 2.9.0 de Greengrass Nucleus](#)

La version 4.5.11 d'IDT pour AWS IoT Greengrass V2 prend désormais en charge la version 2.9.0 de Greengrass nucleus.

17 novembre 2022

[Le gestionnaire de flux v2.1.2 est sorti](#)

Stream Manager v2.1.2 est désormais disponible. Cette version résout un problème sur les systèmes d'exploitation Windows qui utilisent une langue autre que l'anglais.

15 novembre 2022

[AWS IoT Greengrass Mise à jour logicielle Core v2.9.0](#)

Cette version fournit la version 2.9.0 du composant Greengrass Nucleus et AWS met à jour les composants fournis.

15 novembre 2022

[AWS IoT Device Tester La version 4.5.11 prend en charge la version 2.8.1 de Greengrass Nucleus](#)

La version 4.5.11 d'IDT pour AWS IoT Greengrass V2 prend désormais en charge la version 2.8.1 de Greengrass nucleus.

19 octobre 2022

[AWS IoT Device Tester
v4.5.11 avec GGV2 sortie de
Q v2.4.1](#)

La version 4.5.11 d'IDT pour AWS IoT Greengrass V2 est disponible. Cette version inclut la suite de qualification AWS IoT Greengrass V2 (GGV2Q) v2.4.1 et prend en charge les versions 2.8.0, 2.7.0 et 2.6.0 du noyau Greengrass.

13 octobre 2022

[AWS IoT Greengrass Mise à
jour logicielle Core v2.8.1](#)

Cette version fournit la version 2.8.1 du composant Greengrass Nucleus et AWS met à jour les composants fournis.

13 octobre 2022

[AWS IoT Greengrass Mise à
jour logicielle Core v2.8.0](#)

Cette version fournit la version 2.8.0 du composant Greengrass Nucleus et AWS met à jour les composants fournis.

7 octobre 2022

[CloudFormation Support
supplémentaire pour les
déploiements](#)

CloudFormation prend désormais en charge les AWS IoT Greengrass déploiements en tant que ressource.

6 octobre 2022

[SageMaker AI Edge Manager v1.3.0 est sorti](#)

Le composant Amazon SageMaker AI Edge Manager v1.3.0 est disponible. Cette version ajoute la prise en charge de ce composant afin de définir la taille du disque pour le cache du modèle TensorRT et améliore la simultanéité des prédictions afin de mieux utiliser les moteurs d'accélérateur de périphériques tels que. GPUs

1er septembre 2022

[Utiliser le client de communication interprocessus \(IPC\) V2](#)

Ajout d'informations sur le client IPC V2, qui réduisent la quantité de code à écrire pour utiliser les opérations IPC et permettent d'éviter les erreurs courantes susceptibles de se produire avec le client IPC V1.

12 août 2022

[AWS IoT Device Tester v4.5.8 avec sortie de GGV2 Q v2.4.0](#)

La version 4.5.8 d'IDT pour AWS IoT Greengrass V2 est disponible. Cette version inclut la suite de qualification AWS IoT Greengrass V2 (GGV2Q) v2.4.0 et prend en charge les versions 2.7.0, 2.6.0 et 2.5.6 du noyau Greengrass.

12 août 2022

[SageMaker AI Edge Manager v1.2.0 est sorti](#)

Le composant Amazon SageMaker AI Edge Manager v1.2.0 est disponible. Cette version ajoute la prise en charge de ce composant afin de récupérer automatiquement les modèles compilés par SageMaker AI Neo que vous téléchargez sur Amazon S3, afin que vous puissiez déployer de nouveaux modèles sans avoir à créer de AWS IoT Greengrass déploiement.

3 août 2022

[AWS IoT Device Tester La version 4.5.3 prend en charge la version 2.7.0 de Greengrass Nucleus](#)

La version 4.5.3 d'IDT pour AWS IoT Greengrass V2 prend désormais en charge la version 2.7.0 de Greengrass nucleus.

1er août 2022

[Le gestionnaire de flux v2.1.0 est sorti](#)

Stream Manager v2.1.0 est désormais disponible. Cette version inclut la prise en charge de l'envoi de mesures de télémétrie à Amazon. EventBridge

28 juillet 2022

[AWS IoT Greengrass Mise à jour logicielle Core v2.7.0](#)

Cette version fournit la version 2.7.0 du composant Greengrass Nucleus et AWS met à jour les composants fournis. Il inclut une assistance vous permettant d'envoyer des métriques de télémétrie à Amazon. EventBridge

28 juillet 2022

[IoT SiteWise Publisher v2.2.0 est sorti](#)

Le composant IoT SiteWise Publisher v2.2.0 est disponible. Cette version met à jour le composant pour compresser les données avant de les envoyer au AWS IoT SiteWise service, ce qui réduit l'utilisation de la bande passante jusqu'à 75 %.

19 juillet 2022

[Tutoriel : Développement d'un composant qui interagit avec les ombres des appareils clients](#)

Ajout d'un nouveau module au [didacticiel : Interagissez avec des appareils IoT locaux via MQTT](#), que vous pouvez suivre pour apprendre à développer un composant qui interagit avec les ombres des appareils clients.

18 juillet 2022

[Choisissez un courtier MQTT local](#)

Ajout d'informations sur la façon de choisir un courtier MQTT local où les appareils clients se connectent à un périphérique principal.

18 juillet 2022

[AWS IoT Device Tester La version 4.5.3 prend en charge la version 2.6.0 de Greengrass Nucleus](#)

La version 4.5.3 d'IDT pour AWS IoT Greengrass V2 prend désormais en charge la version 2.6.0 de Greengrass nucleus.

29 juin 2022

[AWS IoT Greengrass Mise à jour logicielle Core v2.6.0](#)

27 juin 2022

Cette version fournit la version 2.6.0 du composant Greengrass Nucleus et AWS met à jour les composants fournis. Il inclut la prise en charge des ombres des appareils clients et un broker MQTT 5 local pour les appareils clients. Il inclut également la prise en charge des caractères génériques dans les publish/subscribe rubriques locales, des variables de recette dans les configurations des composants et des caractères génériques dans les politiques d'autorisation IPC. Ces fonctionnalités vous permettent de développer et de configurer plus facilement des composants que vous déployez sur des flottes d'appareils principaux. Cette version inclut également la prise en charge des composants utilisant des opérations IPC qui gèrent les déploiements locaux et les composants sur un périphérique principal.

[Mises à jour des composants des appareils clients](#)

[L'authentification du périphérique client v2.1.0, le courtier MQTT \(Moquette\) v2.1.0, le pont MQTT v2.1.1 et le détecteur IP v2.1.2 sont disponibles.](#) Cette version améliore la rotation des certificats, améliore les performances du broker MQTT et résout les problèmes liés à la manière dont ces composants gèrent les mises à jour de réinitialisation de configuration.

14 juin 2022

[AWS IoT Device Tester La version 4.5.3 prend en charge la version 2.5.6 de Greengrass Nucleus](#)

La version 4.5.3 d'IDT pour AWS IoT Greengrass V2 prend désormais en charge la version 2.5.6 de Greengrass nucleus.

1er juin 2022

[AWS IoT Greengrass Mise à jour logicielle Core v2.5.6](#)

Cette version fournit la version 2.5.6 du composant Greengrass Nucleus et AWS met à jour les composants fournis. Il inclut la prise en charge des modules de sécurité matériels dotés de clés ECC. Il inclut également d'autres corrections de bogues et améliorations.

31 mai 2022

[AWS IoT Le plugin de provisionnement de flotte v1.1.0 est sorti](#)

AWS IoT le plugin de provisionnement de flotte v1.1.0 est disponible. Cette version ajoute la prise en charge de formats de chemin de fichier supplémentaires lorsque vous configurez le plug-in sur des appareils Windows.

[Nouveaux environnements d'exécution Lambda publiés](#)

Ajout du support pour les nouveaux environnements d'exécution Lambda : Python 3.9, Java 11 et NodeJS 14.

[Développer un composant Greengrass qui reporte les mises à jour des composants](#)

Ajout d'un didacticiel que vous pouvez suivre pour apprendre à développer un composant Greengrass qui diffère les mises à jour des composants des déploiements. Vous souhaitez peut-être retarder une mise à jour lorsque le niveau de batterie d'un appareil est faible ou lorsqu'il exécute un processus qui ne peut pas être interrompu, par exemple.

[CloudWatch Metrics v3.1.0
et AWS IoT Device Defender
v3.1.0 ont été publiées](#)

CloudWatch le composant métriques v3.1.0 et le AWS IoT Device Defender composant v3.1.0 sont disponibles. Ces versions ajoutent la prise en charge des configurations de proxy réseau HTTPS. Pour plus d'informations, voir [Se connecter sur le port 443 ou via un proxy réseau](#) et [Activer le périphérique principal pour qu'il fasse confiance à un proxy HTTPS](#).

27 avril 2022

[Migrer depuis AWS IoT
Greengrass Version 1](#)

Ajout d'un guide que vous pouvez suivre pour migrer de AWS IoT Greengrass V1 vers AWS IoT Greengrass V2.

26 avril 2022

[AWS IoT Device Tester v4.5.3 avec GGV2 Q v2.3.1 mis à jour et IDT v4.5.1 avec Q v2.3.0 ajouté aux versions prises en charge GGV2](#)

La version 4.5.3 d'IDT pour AWS IoT Greengrass V2 avec suite de qualification AWS IoT Greengrass V2 (GGV2Q) v2.3.1 a été mise à jour pour inclure le support des versions 2.5.5, 2.5.4 et 2.5.3 du noyau Greengrass. Cette mise à jour inclut également IDT 4.5.1 avec suite de qualification AWS IoT Greengrass V2 (GGV2Q) v2.3.0 en tant que version prise en charge. IDT 4.5.1 avec suite de qualification AWS IoT Greengrass V2 (GGV2Q) v2.3.0 prend en charge la version 2.5.3 du noyau Greengrass.

25 avril 2022

[Sortie de l'adaptateur de protocole Modbus-RTU v2.1.0](#)

Le composant adaptateur de protocole Modbus-RTU v2.1.0 est disponible. Cette version ajoute de nouveaux paramètres que vous pouvez spécifier pour configurer la communication série avec les appareils Modbus RTU.

20 avril 2022

[CloudWatch metrics v2.1.0, Firehose v2.1.0 et Amazon SNS v2.1.0 sont disponibles](#)

CloudWatch le composant metrics v2.1.0, le composant Firehose v2.1.0 et le composant Amazon SNS v2.1.0 sont disponibles. Ces versions ajoutent la prise en charge des configurations de proxy réseau HTTPS. Pour plus d'informations, voir [Se connecter sur le port 443 ou via un proxy réseau et Activer le périphérique principal pour qu'il fasse confiance à un proxy HTTPS.](#)

19 avril 2022

[AWS IoT Device Tester v4.5.3 avec sortie de GGV2 Q v2.3.1](#)

La version 4.5.3 d'IDT pour AWS IoT Greengrass V2 est disponible. Cette version inclut la suite de qualification AWS IoT Greengrass V2 (GGV2Q) v2.3.1 et prend en charge la version 2.5.5 du noyau Greengrass.

15 avril 2022

[AWS IoT Greengrass Mise à jour logicielle Core v2.5.5](#)

Cette version fournit la version 2.5.5 du composant Greengrass Nucleus et AWS met à jour les composants fournis. Il ajoute la prise en charge des appareils Windows qui utilisent une langue d'affichage autre que l'anglais. Cela résout également un problème selon lequel le périphérique principal ne signalait pas son état au service AWS IoT Greengrass cloud après le provisionnement dans certains scénarios.

6 avril 2022

[AWS IoT Greengrass Mise à jour logicielle Core v2.5.4](#)

Cette version fournit la version 2.5.4 du composant Greengrass Nucleus et AWS met à jour les composants fournis. Il inclut des corrections de bogues et des améliorations.

23 mars 2022

[Télécharger AWS IoT Device Tester par programmation](#)

Ajout d'informations sur le téléchargement d'IDT par AWS IoT Greengrass V2 programmation.

15 mars 2022

[Kit de développement Greengrass CLI v1.1.0](#)

La version 1.1.0 de la CLI du kit de développement Greengrass est disponible. Cette version ajoute de nouveaux arguments aux commandes `component publish` et `component init`. Cette version met également à jour la commande `component publish` pour créer le composant s'il n'est pas créé.

24 février 2022

[Shadow Manager v2.1.0 est sorti](#)

Le composant Shadow Manager v2.1.0 est disponible. Cette version ajoute la possibilité de configurer l'intervalle avec AWS IoT Core auquel le composant synchronise les ombres. Par exemple, vous pouvez définir un intervalle plus long pour réduire l'utilisation de la bande passante et les frais.

3 février 2022

[Images Dockerfile et Docker pour AWS IoT Greengrass le logiciel Core v2.5.3](#)

Le Dockerfile et l'image Docker pour le logiciel AWS IoT Greengrass Core v2.5.3 sont désormais disponibles.

12 janvier 2022

[AWS IoT Device Tester v4.5.1
avec sortie de GGV2 Q v2.3.0](#)

La version 4.5.1 d'IDT pour AWS IoT Greengrass V2 est disponible. Cette version inclut la suite de qualification AWS IoT Greengrass V2 (GGV2Q) v2.3.0 et prend en charge la validation et la qualification des appareils Linux qui utilisent un module de sécurité matériel (HSM) pour stocker la clé privée et le certificat utilisés par le logiciel Core. AWS IoT Greengrass

11 janvier 2022

[AWS IoT Greengrass Mise à
jour logicielle Core v2.5.3](#)

Cette version fournit la version 2.5.3 du composant Greengrass Nucleus et AWS met à jour les composants fournis. Il inclut une assistance vous permettant de configurer le logiciel AWS IoT Greengrass principal pour utiliser une clé privée et un certificat que vous stockez en toute sécurité dans un module de sécurité matériel (HSM).

6 janvier 2022

[Images Dockerfile et Docker
pour AWS IoT Greengrass le
logiciel Core v2.5.2](#)

Le Dockerfile et l'image Docker pour le logiciel AWS IoT Greengrass Core v2.5.2 sont désormais disponibles.

20 décembre 2021

[AWS IoT Device Tester v4.4.1
avec sortie de GGV2 Q v2.2.1](#)

La version 4.4.1 d'IDT pour AWS IoT Greengrass V2 est disponible. Cette version inclut la suite de qualification AWS IoT Greengrass V2 (GGV2Q) v2.2.1 et prend en charge la version 2.5.2 de Greengrass nucleus pour la qualification des appareils.

12 décembre 2021

[Effectuez des inférences
basées sur le machine
learning à l'aide d'Amazon
Lookout for Vision](#)

Ajout d'informations sur la manière d'effectuer des inférences basées sur le machine learning à l'aide de Lookout for Vision sur les appareils principaux de Greengrass. Lookout for Vision utilise la vision par ordinateur pour détecter les défauts visuels des produits industriels.

8 décembre 2021

[AWS IoT Device Tester v4.4.1
avec sortie de GGV2 Q v2.2.0](#)

La version 4.4.1 d'IDT pour AWS IoT Greengrass V2 est disponible. Cette version inclut la suite de qualification AWS IoT Greengrass V2 (GGV2Q) v2.2.0 et prend en charge la version 2.5.2 de Greengrass nucleus pour la qualification des appareils.

6 décembre 2021

[AWS IoT Greengrass Mise à jour logicielle Core v2.5.2](#)

Cette version fournit la version 2.5.2 du composant Greengrass Nucleus et AWS met à jour les composants fournis. Il corrige un problème avec le service Windows qui se produit après les mises à jour du noyau Greengrass. Il inclut également la prise en charge du AWS IoT Device Defender composant sur les appareils Windows.

3 décembre 2021

[Nouveau connecteur Edge pour le composant Kinesis Video Streams](#)

La version 1.0.0 du connecteur Edge pour le composant Kinesis Video Streams est disponible. Ce AWS service permet de lire les flux vidéo des caméras locales et de les publier sur Kinesis Video Streams. Ce composant s'intègre à AWS IoT TwinMaker, ce qui vous permet de visualiser et de gérer les flux vidéo et autres données dans les tableaux de bord Grafana.

30 novembre 2021

[Gérez les appareils principaux de Greengrass avec AWS Systems Manager](#)

Ajout d'informations sur la façon de gérer les appareils principaux de Greengrass avec. AWS Systems Manager Systems Manager est un AWS service qui vous permet de visualiser les données opérationnelles, d'automatiser les tâches opérationnelles et de garantir la sécurité et la conformité.

29 novembre 2021

[Kit de développement Greengrass CLI](#)

Ajout d'informations sur l'interface de ligne de commande du kit de AWS IoT Greengrass développement (GDK CLI), un outil que vous pouvez télécharger sur votre ordinateur de développement local pour vous aider à développer des composants Greengrass personnalisés. Vous pouvez utiliser la CLI GDK pour créer, créer et publier des composants personnalisés.

29 novembre 2021

[Composants Greengrass fournis par la communauté](#)

Ajout d'informations sur le catalogue de logiciels Greengrass, qui est un index des composants de Greengrass développés par la communauté Greengrass. À partir de ce catalogue, vous pouvez télécharger, modifier et déployer des composants pour créer vos applications Greengrass.

29 novembre 2021

[AWS IoT Greengrass Mise à jour logicielle Core v2.5.1](#)

Cette version fournit la version 2.5.1 du composant Greengrass Nucleus et AWS met à jour les composants fournis. Il inclut le support de Java 32 bits sur les appareils Windows. Il résout également les problèmes liés au nouveau comportement de suppression des groupes d'objets et au chargement des variables d'environnement système sur les appareils Windows.

23 novembre 2021

[AWS IoT Device Tester v4.4.0 avec sortie de GGV2 Q v2.1.0](#)

La version 4.4.0 d'IDT pour AWS IoT Greengrass V2 est disponible. Cette version inclut la suite de qualification AWS IoT Greengrass V2 (GGV2Q) v2.1.0 et prend en charge la qualification des appareils Greengrass basés sur Windows exécutant Greengrass nucleus version 2.5.0.

19 novembre 2021

[AWS IoT Greengrass Mise à jour logicielle Core v2.5.0](#)

Cette version fournit la version 2.5.0 du composant Greengrass Nucleus et AWS met à jour les composants fournis. Il inclut la prise en charge de l'exécution du logiciel AWS IoT Greengrass Core sur des appareils Windows. Cela modifie également le comportement de suppression des groupes d'objets et ajoute la prise en charge des proxys HTTPS.

12 novembre 2021

[SageMaker AI Edge Manager v1.1.0 est sorti](#)

Le composant Amazon SageMaker AI Edge Manager v1.1.0 est disponible. Cette version ajoute la prise en charge des appareils principaux de Greengrass exécutant Amazon Linux 2 et ajoute un nouveau paramètre de configuration pour spécifier l'emplacement du dossier de données de capture sur votre appareil.

03 novembre 2021

[Mise à jour de la prévention du problème de l'adjoint confus entre services](#)

AWS IoT Greengrass V2 prend en charge l'utilisation 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

[Mises à jour des composants des appareils clients](#)

[L'authentification du périphérique client v2.0.3](#), [le détecteur IP v2.1.0](#), [le pont MQTT v2.1.0](#) et [le courtier MQTT \(Moquette\) v2.0.2](#) sont disponibles. Cette version ajoute une prise en charge complète des ports de broker MQTT autres que ceux par défaut et inclut d'autres corrections de bogues et améliorations.

28 octobre 2021

[Shadow Manager v2.0.4 est sorti](#)

Le composant Shadow Manager v2.0.4 est disponible. Cette version corrige un problème en raison duquel le gestionnaire des ombres supprimait les versions nouvellement créées de toutes les ombres précédemment supprimées. À partir de cette version, l'opération `DeleteThingShadow` IPC incrémente la version fantôme.

20 octobre 2021

[Lancement du gestionnaire de journaux v2.2.0](#)

Le composant Log Manager v2.2.0 est disponible. Le gestionnaire de journaux prend désormais en charge l'utilisation d'une carte de configuration pour fournir les configurations des journaux des composants.

20 octobre 2021

[Lambda Manager v2.1.4 est sorti](#)

Le composant Lambda Manager v2.1.4 est disponible. Cette version corrige un problème selon lequel les fonctions Lambda utilisant les environnements d'exécution NodeJS ne traitaient qu'un seul message.

20 octobre 2021

Utiliser la communication interprocessus, les AWS informations d'identification et le gestionnaire de flux dans les composants du conteneur Docker	Ajout d'informations sur l'utilisation de la communication interprocessus (IPC), des AWS informations d'identification et du gestionnaire de flux dans vos composants de conteneur Docker personnalisés.	19 octobre 2021
Nouveau composant émetteur de télémétrie Nucleus	La version 1.0.0 du composant émetteur de télémétrie Nucleus est disponible. Ce composant AWS fournit la collecte des données de télémétrie relatives à l'état du système et les publie en permanence sur un sujet local et un sujet MQTT. AWS IoT Core	30 septembre 2021
Autoriser le trafic des appareils via un proxy ou un pare-feu	Ajout d'informations sur les points de terminaison et les ports utilisés par l'appareil principal de Greengrass, afin que vous puissiez restreindre le trafic par mesure de sécurité.	16 septembre 2021
AWS IoT Device Tester v4.2.0 avec sortie de GGV2 Q v2.0.1	La version 4.2.0 d'IDT pour AWS IoT Greengrass V2 a été mise à jour avec la suite de qualification AWS IoT Greengrass V2 (GGV2Q) v2.0.1. Cette version prend en charge la version 2.4.0 de Greengrass Nucleus pour la qualification des appareils.	31 août 2021

[Composants du programme d'installation d'apprentissage automatique mis à jour](#)

Le composant d'installation DLR v1.6.5 et le composant d'installation TensorFlow Lite v2.5.4 sont disponibles. Ces versions de composants incluent le nouveau paramètre `UseInstaller` de configuration qui permet de désactiver le script d'installation par défaut.

30 août 2021

[Support Linux intégré pour AWS IoT Greengrass](#)

La BitBake recette AWS IoT Greengrass V2 est disponible dans le meta-aws projet sur GitHub. Vous pouvez utiliser cette recette pour créer un système d'exploitation personnalisé basé sur Linux à l'aide du projet Yocto.

20 août 2021

[Intégrité du code](#)

Ajout d'informations sur la manière de AWS IoT Greengrass V2 vérifier l'intégrité des logiciels que les appareils principaux de Greengrass téléchargent depuis le AWS Cloud

19 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

Le gestionnaire de flux v2.0.12 est sorti	Stream Manager v2.0.12 est désormais disponible. Cette version corrige un problème qui empêchait les mises à niveau de la version 2.0.7 du composant du gestionnaire de flux vers une version comprise entre v2.0.8 et v2.0.11.	10 août 2021
Images Dockerfile et Docker pour AWS IoT Greengrass le logiciel Core v2.4.0	Le Dockerfile et l'image Docker pour le logiciel AWS IoT Greengrass Core v2.4.0 sont désormais disponibles.	9 août 2021
AWS IoT Greengrass Mise à jour logicielle Core v2.4.0	Cette version fournit la version 2.4.0 du composant Greengrass Nucleus et AWS met à jour les composants fournis. Il inclut la prise en charge des limites de ressources du système des composants, des opérations IPC pour suspendre et reprendre les composants, et des plug-ins de provisionnement.	3 août 2021
Nouveaux AWS IoT SiteWise composants	Ajout des composants suivants AWS fournis pour AWS IoT SiteWise : le collecteur IoT SiteWise OPC-UA, l'éditeur IoT SiteWise et le processeur IoT SiteWise	29 juillet 2021

[AWS IoT Device Tester v4.2.0 avec sortie de GGV2 Q v2.0.0](#)

La version 4.2.0 d'IDT pour AWS IoT Greengrass V2 est disponible. Cette version inclut la suite de qualification AWS IoT Greengrass V2 (GGV2Q) v2.0.0 et inclut la prise en charge des tests de qualification facultatifs pour les composants Docker, l'apprentissage automatique et le gestionnaire de flux.

14 juillet 2021

[AWS IoT Greengrass Bibliothèque que IPC de base disponible Kit SDK des appareils AWS IoT pour C ++ v2](#)

La version 1.13.0 de Kit SDK des appareils AWS IoT for C ++ v2 prend en charge AWS IoT Greengrass Core IPC, ce qui vous permet de développer des composants en C++ qui interagissent avec le AWS IoT Greengrass logiciel Core.

14 juillet 2021

[SageMaker Le composant AI Edge Manager v1.0.2 est sorti](#)

Le composant Amazon SageMaker AI Edge Manager v1.0.2 est disponible. Cette version met à jour le script d'installation dans le cycle de vie des composants. Python 3.6 ou version ultérieure doit désormais être installé sur vos appareils principaux, y compris pip pour votre version de Python, avant de déployer ce composant.

12 Juillet 2021

<u>Support mis à jour AWS IoT Device Tester pour la AWS IoT Greengrass version 2</u>	La version 4.1.0 d'IDT pour AWS IoT Greengrass V2 prend désormais en charge l'utilisation de Greengrass nucleus version 2.3.0 pour la qualification des appareils.	8 juillet 2021
<u>Images Dockerfile et Docker pour AWS IoT Greengrass le logiciel Core v2.3.0</u>	Le Dockerfile et l'image Docker pour le logiciel AWS IoT Greengrass Core v2.3.0 sont désormais disponibles.	7 juillet 2021
<u>AWS politiques gérées</u>	Ajout d'informations sur les politiques AWS gérées pour AWS IoT Greengrass.	2 juillet 2021
<u>Nouvelles options JVM recommandées</u>	Ajout d'informations sur les options JVM recommandées pour contrôler l'allocation de mémoire pour le logiciel AWS IoT Greengrass Core.	30 Juin 2021
<u>AWS IoT Greengrass Mise à jour logicielle Core v2.3.0</u>	Cette version fournit la version 2.3.0 du composant Greengrass Nucleus et AWS met à jour les composants fournis. Il inclut la prise en charge des documents de configuration de composant s volumineux dans les déploiements.	29 juin 2021
<u>Images Dockerfile et Docker pour AWS IoT Greengrass le logiciel Core v2.2.0</u>	Le Dockerfile et l'image Docker pour le logiciel AWS IoT Greengrass Core v2.2.0 sont désormais disponibles.	28 juin 2021

[AWS IoT Device Tester v4.1.0 avec sortie de GGV2 Q v1.1.1](#)

La version 4.1.0 d'IDT pour AWS IoT Greengrass V2 est disponible. Cette version inclut la suite de qualification AWS IoT Greengrass V2 (GGV2Q) v1.1.1 et prend en charge l'utilisation de Greengrass nucleus v2.2.0, v2.1.0 et v2.0.5 pour la qualification des appareils.

18 juin 2021

[AWS IoT Greengrass Mise à jour logicielle Core v2.2.0](#)

Cette version fournit la version 2.2.0 du composant Greengrass Nucleus et AWS met à jour les composants fournis. Il inclut des composants que vous pouvez déployer pour prendre en charge les appareils clients et ajouter le service parallèle local.

18 juin 2021

[Lambda Launcher v2.0.6 est sorti](#)

La version 2.0.6 du composant du lanceur Lambda est disponible. Cette version inclut des améliorations de performances et des corrections de bogues.

13 juin 2021

[Nouveau composant SageMaker AI Edge Manager publié](#)

La version 1.0.0 du composant Amazon SageMaker AI Edge Manager est disponible pour AWS IoT Greengrass. Ce composant installe le binaire de l'agent SageMaker AI Edge Manager sur les appareils principaux de Greengrass.

10 juin 2021

[Types de composants](#)

Des informations sur les types de composants ont été ajoutées dans AWS IoT Greengrass. Le type de composant indique comment le logiciel AWS IoT Greengrass Core exécute un composant.

3 juin 2021

[AWS IoT Device Tester v4.0.2 avec sortie de GGV2 Q v1.1.0](#)

La version 4.0.2 d'IDT pour AWS IoT Greengrass V2 est disponible. Cette version inclut la suite de qualification AWS IoT Greengrass V2 (GGV2Q) v1.1.0 et prend en charge l'utilisation de Greengrass nucleus v2.1.0 avec Greengrass CLI v2.1.0 pour la qualification des appareils. Cela inclut également de nouveaux groupes de test obligatoires pour MQTT et Lambda, ainsi que d'autres corrections de bogues et améliorations mineures.

5 mai 2021

[Images Dockerfile et Docker pour AWS IoT Greengrass le logiciel Core v2.1.0](#)

Le Dockerfile et l'image Docker pour le logiciel AWS IoT Greengrass Core v2.1.0 sont désormais disponibles. L'image Docker vous permet d'exécuter le logiciel AWS IoT Greengrass principal dans un conteneur Docker qui utilise Amazon Linux 2 comme système d'exploitation de base.

27 avril 2021

[AWS IoT Greengrass Mise à jour logicielle Core v2.1.0](#)

Cette version fournit la version 2.1.0 du composant Greengrass Nucleus et AWS met à jour les composants fournis. Il inclut un nouveau composant que vous pouvez utiliser pour télécharger des images Docker à partir de référentiels Amazon ECR privés, ainsi que de nouveaux exemples de composants pour effectuer des inférences d'apprentissage automatique à l'aide de Lite. TensorFlow

26 avril 2021

[Exemple de composant utilisant Secrets Manager](#)

Ajout d'un exemple de composant qui imprime la valeur d'un AWS Secrets Manager secret que vous déployez sur un périphérique principal.

08 avril 2021

[AWS IoT Politique minimale pour les appareils principaux de Greengrass](#)

Ajout d'informations sur l'ensemble minimal d'autorisations requises pour prendre en charge les fonctionnalités de base de Greengrass sur un appareil principal.

2 avril 2021

[Abonnez-vous aux diffusions d'événements IPC](#)

Ajout d'informations sur l'utilisation des opérations de communication interprocessus (IPC) pour s'abonner à des flux d'événements sur un appareil principal Greengrass.

1 avril 2021

[Support mis à jour AWS IoT Device Tester pour AWS IoT Greengrass](#)

La version 4.0.1 d'IDT pour AWS IoT Greengrass V2 prend désormais en charge l'utilisation de Greengrass nucleus version 2.0.5 avec Greengrass CLI version 2.0.5 pour la qualification des appareils.

17 mars 2021

[Créez des composants personnalisés qui utilisent le gestionnaire de flux](#)

Ajout d'informations sur la façon de configurer des recettes de composants et des artefacts pour développer des applications qui gèrent les flux de données.

9 mars 2021

[AWS IoT Greengrass Mise à jour logicielle Core v2.0.5](#)

Cette version fournit la version 2.0.5 du composant Greengrass Nucleus et AWS met à jour les composants fournis. Il résout un problème lié à la prise en charge des proxys réseau et un problème lié au point de terminaison du plan de données Greengrass dans les régions de AWS Chine.

9 mars 2021

[Référence à la variable d'environnement du composant](#)

Ajout d'informations sur les variables d'environnement que le logiciel AWS IoT Greengrass Core définit pour les composants. Vous pouvez utiliser ces variables d'environnement pour obtenir le nom de l'objet et la Région AWS version du noyau de Greengrass.

23 février 2021

[Installation manuelle](#)

Ajout d'informations sur la façon de créer les AWS ressources requises manuellement ou de les installer derrière un pare-feu ou un proxy réseau. En utilisant une installation manuelle, vous n'avez pas besoin d'autoriser le programme d'installation à créer des ressources dans votre ordinateur Compte AWS, car vous créez les ressources requises AWS IoT et les ressources IAM. Vous pouvez également configurer votre appareil pour qu'il se connecte sur le port 443 ou via un proxy réseau.

17 février 2021

[AWS IoT Greengrass Mise à jour de la bibliothèque IPC de base Kit SDK des appareils AWS IoT pour Python v2](#)

La version 1.5.4 de Kit SDK des appareils AWS IoT for Python v2 simplifie les étapes requises pour se connecter au service AWS IoT Greengrass Core IPC.

11 février 2021

[Support mis à jour AWS IoT Device Tester pour AWS IoT Greengrass](#)

La version 4.0.1 d'IDT pour AWS IoT Greengrass V2 prend désormais en charge l'utilisation de Greengrass nucleus version 2.0.4 avec Greengrass CLI version 2.0.4 pour la qualification des appareils.

5 février 2021

[Nouveau tutoriel pour importer des fonctions Lambda](#)

Ajout d'un nouveau didacticiel basé sur une console pour importer une fonction Lambda en tant que composant exécuté sur le périphérique principal de Greengrass.

5 février 2021

[AWS IoT Greengrass Mise à jour logicielle Core v2.0.4](#)

Cette version fournit la version 2.0.4 du composant Greengrass nucleus. Il inclut le nouveau greengrassDataPlanePort paramètre permettant de configurer la communication HTTPS sur le port 443 et corrige les bogues. La politique IAM minimale nécessite désormais le iam:GetPolicy et sts:GetCallerIdentity quand le programme d'installation du logiciel AWS IoT Greengrass Core est exécuté avec --provision true.

4 février 2021

[Nouveau composant de tunneling sécurisé publié](#)

La version 1.0.0 du composant de tunneling sécurisé est disponible pour. AWS IoT Greengrass Ce composant AWS fourni utilise un tunneling AWS IoT sécurisé pour établir une communication bidirectionnelle sécurisée avec un appareil central Greengrass qui se trouve derrière des pare-feux restreints.

21 janvier 2021

[AWS IoT Device Tester pour la AWS IoT Greengrass version 4.0.1 publiée](#)

La version 4.0.1 d'IDT pour AWS IoT Greengrass V2 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.

22 décembre 2020

[Publication initiale de AWS IoT Greengrass Version 2](#)

AWS IoT Greengrass V2 est une nouvelle version majeure de AWS IoT Greengrass. Cette version ajoute plusieurs fonctionnalités telles que des composants logiciels modulaires et des déploiements continus. Ces fonctionnalités vous permettent de développer et de gérer plus facilement des applications de pointe.

15 décembre 2020

AWS Glossaire

Pour la AWS terminologie la plus récente, consultez le [AWS glossaire](#) dans la Glossaire AWS référence.

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.