



Guide du développeur

Intégrations gérées pour AWS IoT Device Management



Intégrations gérées pour AWS IoT Device Management: Guide du développeur

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

À quoi servent les intégrations gérées AWS IoT Device Management	1
Régions prises en charge	1
Utilisez-vous les intégrations gérées pour la première fois ?	1
Présentation des intégrations gérées	1
Terminologie des intégrations gérées	2
Terminologie générale des intégrations gérées	2
Cloud-to-cloud terminologie	2
Terminologie des modèles de données	3
Configurer des intégrations gérées	4
Inscrivez-vous pour un Compte AWS	4
Création d'un utilisateur doté d'un accès administratif	4
Mise en route	7
Types d'appareils	7
Configuration de la clé de chiffrement	8
Techniques d'intégration	9
Intégration des appareils connectés directement	9
Intégration au hub	9
Intégration des appareils connectés au Hub	9
Cloud-to-cloud intégration des appareils	9
Mise en service des appareils	10
Gérez le cycle de vie et les profils des appareils	12
Appareil	12
Profil de l'appareil	13
Modèles de données	14
Modèle de données d'intégrations gérées	14
AWS mise en œuvre du modèle de données Matter	16
Schémas de modèles de données	17
Schéma de capacité	18
Schéma de définition de type	19
Schéma pour les définitions des capacités	19
Schéma pour les définitions de type	37
Création et utilisation de définitions de type dans les documents de schéma de capacités	42
Commandes et événements de l'appareil	56
Commandes de l'appareil	56

Événements liés à l'appareil	59
Balisage des ressources	60
Principes de base des étiquettes	60
Limites et restrictions liées aux balises	61
Marquer avec les politiques IAM	61
Notifications d'intégrations gérées	65
Configurer Amazon Kinesis pour les notifications	65
Étape 1 : créer un flux de données Amazon Kinesis	65
Étape 2 : créer une politique d'autorisation	66
Étape 3 : Accédez au tableau de bord IAM et sélectionnez Rôles	66
Étape 4 : Utiliser une politique de confiance personnalisée	66
Étape 5 : appliquez votre politique d'autorisation	67
Étape 6 : Entrez un nom de rôle	68
Configurer les notifications d'intégrations gérées	68
Étape 1 : autoriser l'utilisateur à appeler l' CreateDestination API	68
Étape 2 : appeler l' CreateDestination API	69
Étape 3 : appeler l' CreateNotificationConfiguration API	69
Types d'événements surveillés avec des intégrations gérées	70
Cloud-to-Cloud connecteurs (C2C)	75
Qu'est-ce qu'un connecteur cloud-to-cloud (C2C) ?	75
Catalogue de connecteurs	75
AWS Lambda fonctionne comme des connecteurs C2C	76
Flux de travail du connecteur d'intégrations gérées	76
Directives d'utilisation d'un connecteur C2C (cloud-to-cloud)	77
Créez un connecteur C2C (Cloud-to-Cloud)	77
Conditions préalables	77
Exigences relatives au connecteur C2C	78
OAuth 2.0 exigences relatives à la liaison de comptes	80
Mettre en œuvre les opérations d'interface du connecteur C2C	86
Appelez votre connecteur C2C	108
Ajoutez des autorisations à votre rôle IAM	109
Testez manuellement votre connecteur C2C	110
Utiliser un connecteur C2C (Cloud-to-Cloud)	110
Hub SDK	122
Architecture du SDK Hub	122
Intégration des appareils	122

Composants d'intégration des appareils	122
Flux d'intégration des appareils	124
Contrôle de l'appareil	125
Flux de contrôle des appareils	126
Composants du SDK	126
Installation et validation du SDK Hub pour les intégrations gérées	127
Installez le SDK à l'aide de AWS IoT Greengrass	127
Déployez le SDK du Hub à l'aide d'un script	130
Déployer le SDK Hub avec systemd	133
Embarquez vos hubs	136
Sous-système d'intégration du hub	137
Configuration pour l'intégration	138
Intégrez les appareils et faites-les fonctionner dans le hub	147
Configuration simple pour embarquer et faire fonctionner les appareils	147
Configuration guidée par l'utilisateur pour embarquer et faire fonctionner les appareils	154
Gestionnaire de certificats personnalisé	163
Définition et composants de l'API	163
Exemple de construction	165
Utilisation	169
Plug-in de protocole personnalisé	170
Client du SDK Hub	171
Obtenez votre SDK Hub d'intégrations gérées	171
À propos de la boîte à outils du SDK Hub	171
Créez votre application personnalisée avec le client Hub SDK	172
Exécution de votre application personnalisée	174
API du client du SDK Hub	175
Types de données	180
Contrôle du hub	182
Prérequis	182
Composants du SDK pour appareils finaux	182
Intégrer au SDK pour appareils finaux	183
Exemple : créer un contrôle du hub	186
Exemples pris en charge	186
Plateformes prises en charge	187
Activer CloudWatch les journaux	187
Prérequis	187

Configuration des configurations du journal du SDK Hub	188
Types d'appareils Zigbee et Z-Wave pris en charge	190
Exécutez des intégrations gérées sur Raspberry Pi	192
Flash du microprogramme Sonoff Zigbee	193
Image du SDK Hub d'intégrations gérées sur Raspberry Pi	194
Intégrations gérées Hub SDK, conteneur Docker sur Raspberry Pi	198
Application de démonstration d'intégrations gérées	203
Hub d'intégrations gérées hors bord	205
Présentation du processus de déconnexion du SDK Hub	205
Prérequis	206
Processus de déconnexion du SDK Hub	206
Après avoir déintégré le SDK Hub	210
Intergiciel spécifique au protocole	211
Architecture intergicelle	212
End-to-end exemple de flux de commande intergiciel	212
Organisation du code Middleware	213
Intégrer le middleware au SDK	219
Kit SDK des appareils finaux	222
Qu'est-ce que le SDK pour appareils finaux ?	222
Architecture et composants	223
Bénéficiaire	224
Flux de travail Provisionee	224
Définir les variables d'environnement	225
Enregistrer un point de terminaison personnalisé	225
Création d'un profil de provisionnement	226
Création d'un objet géré	226
Approvisionnement Wi-Fi des utilisateurs du SDK	227
Approvisionnement de la flotte par sinistre	227
Fonctionnalités de l'objet géré	228
Mises à jour OTA	228
Présentation de l'architecture OTA	228
Conditions préalables	228
Tâches de mise en œuvre Over-the-Air (OTA)	229
Configuration des configurations de tâches OTA	232
Appliquer les paramètres de configuration aux tâches OTA	233
Surveillez les notifications OTA	233

Traiter les documents de travail	235
Implémenter un agent OTA	235
Générateur de code de modèle de données	236
Processus de génération de code	237
Configuration de l'environnement	240
Générer du code pour les appareils	241
Fonction C de faible niveau APIs	243
OnOff API de cluster	244
Interactions entre le service et l'appareil	246
Gestion des commandes à distance	246
Gestion des événements non sollicités	247
Commencez avec le SDK pour appareils finaux	248
Portez le SDK de l'appareil final	260
Référence technique	264
Sécurité	267
Protection des données	268
Chiffrement des données au repos pour les intégrations gérées	269
Gestion des identités et des accès	275
Public ciblé	276
Authentification par des identités	276
Gestion de l'accès à l'aide de politiques	278
AWS politiques gérées	280
Comment fonctionnent les intégrations gérées avec IAM	284
Exemples de politiques basées sur l'identité	290
Résolution des problèmes	293
Utilisation des rôles liés à un service	295
Utilisation à des AWS Secrets Manager fins de protection des données pour les flux de travail	
C2C	299
Comment les intégrations gérées utilisent les secrets	299
Comment créer un secret	300
Accordez l'accès aux intégrations gérées AWS IoT Device Management afin de récupérer le secret	300
Validation de conformité	301
Utilisez des intégrations gérées avec les points de terminaison VPC d'interface	302
Considérations relatives aux points de terminaison VPC	302
Création de points de terminaison de VPC	304

Test des points de terminaison VPC	305
Contrôle d'accès	306
Tarification	308
Limitations	308
Connectez-vous à des intégrations gérées pour les points de terminaison AWS IoT Device Management FIPS	309
Points de terminaison du plan de contrôle	309
Surveillance	310
CloudTrail journaux	310
Événements de gestion dans CloudTrail	312
Exemples d'événements	313
Historique de la documentation	317
.....	cccxviii

À quoi servent les intégrations gérées ? AWS IoT Device Management

Les intégrations gérées AWS IoT Device Management aident les fournisseurs de solutions IoT à unifier le contrôle et la gestion des appareils IoT de centaines de fabricants. Vous pouvez utiliser des intégrations gérées pour automatiser les flux de travail de configuration des appareils et garantir l'interopérabilité entre de nombreux appareils, quel que soit le fournisseur de l'appareil ou le protocole de connectivité. Avec les intégrations gérées, vous pouvez utiliser une interface utilisateur unique et un ensemble de APIs paramètres pour contrôler, gérer et faire fonctionner une gamme d'appareils.

Rubriques

- [Régions prises en charge](#)
- [Utilisez-vous les intégrations gérées pour la première fois ?](#)
- [Présentation des intégrations gérées](#)
- [Terminologie des intégrations gérées](#)

Régions prises en charge

Les intégrations gérées pour AWS IoT Device Management sont prises en charge dans les régions suivantes :

- Canada (Centre)
- Europe (Irlande)

Utilisez-vous les intégrations gérées pour la première fois ?

Si vous utilisez des intégrations gérées pour la première fois, nous vous recommandons de commencer par lire les sections suivantes :

- [Configurer des intégrations gérées](#)
- [Commencez avec les intégrations gérées pour AWS IoT Device Management](#)

Présentation des intégrations gérées

L'image suivante fournit une vue d'ensemble détaillée des intégrations gérées

Terminologie des intégrations gérées

Au sein des intégrations gérées, il existe de nombreux concepts et termes essentiels à comprendre pour gérer la mise en œuvre de vos propres appareils. Les sections suivantes décrivent ces concepts et termes clés afin de mieux comprendre les intégrations gérées.

Terminologie générale des intégrations gérées

Un concept important à comprendre pour les intégrations gérées est celui d'un objet géré par rapport à un AWS IoT Core objet quelconque.

- **AWS IoT Core chose** : Un AWS IoT Core objet est une AWS IoT Core construction qui fournit une représentation numérique. Les développeurs sont tenus de gérer les politiques, le stockage des données, les règles, les actions, les sujets MQTT et la transmission de l'état de l'appareil au stockage des données. Pour plus d'informations sur ce qu'est un AWS IoT Core objet, consultez [la section Gestion des appareils avec AWS IoT](#).
- **Intégrations gérées Managed Thing** : avec un objet géré, nous fournissons une abstraction pour simplifier les interactions entre les appareils et n'obligeons pas le développeur à créer des éléments tels que des règles, des actions, des sujets MQTT et des politiques.

Cloud-to-cloud terminologie

Les appareils physiques qui s'intègrent aux intégrations gérées peuvent provenir d'un fournisseur de cloud tiers. Pour intégrer ces appareils aux intégrations gérées et communiquer avec le fournisseur de cloud tiers, la terminologie suivante couvre certains des concepts clés qui sous-tendent ces flux de travail :

- **Cloud-to-cloud Connecteur (C2C)** : un connecteur C2C établit une connexion entre les intégrations gérées et le fournisseur de cloud tiers.
- **Fournisseur de cloud tiers** : pour les appareils fabriqués et gérés en dehors des intégrations gérées, un fournisseur de cloud tiers permet à l'utilisateur final de contrôler ces appareils et les

intégrations gérées communiquent avec le fournisseur de cloud tiers pour divers flux de travail tels que les commandes des appareils.

Terminologie des modèles de données

Les intégrations gérées utilisent des modèles de données pour organiser les données et end-to-end la communication entre vos appareils. La terminologie suivante couvre certains des concepts clés permettant de comprendre ces deux modèles de données :

- **Appareil** : entité représentant un appareil physique (tel qu'une sonnette vidéo) dont plusieurs nœuds fonctionnent ensemble pour fournir un ensemble complet de fonctionnalités.
- **Point de terminaison** : un terminal encapsule une fonctionnalité autonome (sonnerie, détection de mouvement, éclairage d'une sonnette vidéo).
- **Capacité** : entité représentant les composants nécessaires pour rendre une fonctionnalité disponible sur un terminal (bouton ou voyant et carillon dans la fonction de sonnette d'une sonnette vidéo).
- **Action** : entité représentant une interaction avec une fonctionnalité d'un appareil (faire sonner la cloche ou voir qui est à la porte).
- **Événement** : entité représentant un événement issu d'une fonctionnalité d'un appareil. Un appareil peut envoyer un événement pour signaler un événement (incident/alarm, an activity from a sensor etc. (e.g. there is knock/ringsur la porte).
- **Propriété** : entité représentant un attribut particulier dans l'état de l'appareil (la cloche sonne, la lumière du porche est allumée, la caméra enregistre).
- **Modèle de données** : La couche de données correspond aux éléments de données et de verbes qui contribuent au bon fonctionnement de l'application. L'application fonctionne sur ces structures de données lorsqu'il existe une intention d'interagir avec l'appareil. Pour plus d'informations, consultez [connectedhomeip](https://github.com/aws-samples/aws-iot-device-management-quickstart) sur le site Web. GitHub
- **Schéma** : un schéma est une représentation du modèle de données au format JSON.

Configurer des intégrations gérées

Les sections suivantes vous guident dans la configuration initiale de l'utilisation des intégrations gérées pour AWS IoT Device Management.

Rubriques

- [Inscrivez-vous pour un Compte AWS](#)
- [Création d'un utilisateur doté d'un accès administratif](#)

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 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](#).

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

Après vous être inscrit à un Compte AWS, sécurisez Utilisateur racine d'un compte AWS AWS IAM Identity Center, activez et créez un utilisateur administratif 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 .

Commencez avec les intégrations gérées pour AWS IoT Device Management

Les sections suivantes décrivent les étapes à suivre pour commencer à utiliser les intégrations gérées.

Rubriques

- [Types d'appareils](#)
- [Configuration de la clé de chiffrement](#)
- [Techniques d'intégration](#)

Types d'appareils

Les intégrations gérées gèrent de nombreux types d'appareils. Chaque appareil appartient à l'une des trois catégories suivantes :

- Appareils connectés directement : ce type d'appareil se connecte directement à un point de terminaison d'intégrations géré. Généralement, ces appareils sont conçus et gérés par des fabricants d'appareils qui incluent le SDK pour appareils finaux d'intégrations gérées pour la connectivité directe.
- Appareils connectés au hub : ces appareils se connectent aux intégrations gérées via un hub exécutant le SDK du hub d'intégrations gérées, qui gère les fonctions de découverte, d'intégration et de contrôle des appareils. Les utilisateurs finaux peuvent intégrer ces appareils en appuyant sur un bouton ou en scannant un code-barres.

Les deux flux de travail suivants sont pris en charge pour l'intégration d'un appareil connecté au hub :

- Appuyer sur un bouton à l'initiative de l'utilisateur final pour démarrer la découverte de l'appareil
- Numérisation basée sur des codes-barres pour effectuer l'association des appareils
- Cloud-to-cloud Appareils (C2C) : il s'agit d'appareils conçus et gérés par des fournisseurs qui gèrent leur propre infrastructure cloud et des applications mobiles de marque pour le contrôle des appareils. Les clients des intégrations gérées peuvent accéder à un catalogue de connecteurs C2C prédéfinis ou créer les leurs, afin de développer des solutions IoT compatibles avec plusieurs clouds de fournisseurs tiers via une interface unifiée.

Lorsque l'utilisateur final allume un appareil C2C pour la première fois, celui-ci doit être approvisionné auprès de son fournisseur de cloud tiers respectif pour les intégrations gérées afin d'obtenir les fonctionnalités et les métadonnées de son appareil. Une fois ce flux de provisionnement terminé, les intégrations gérées peuvent communiquer avec le périphérique cloud et le fournisseur de cloud tiers pour le compte de l'utilisateur final.

Note

Un hub n'est pas un type d'appareil spécifique comme indiqué ci-dessus. Son objectif est de jouer le rôle de contrôleur des appareils domestiques intelligents et de faciliter la connexion entre les intégrations gérées et les fournisseurs de cloud tiers. Il peut jouer le rôle à la fois en tant que type d'appareil comme indiqué ci-dessus et en tant que hub.

Configuration de la clé de chiffrement

La sécurité est d'une importance capitale pour les données acheminées entre l'utilisateur final, les intégrations gérées et les clouds tiers. L'une des méthodes que nous prenons en charge pour protéger les données de votre appareil est end-to-end le chiffrement utilisant une clé de chiffrement sécurisée pour acheminer vos données.

En tant que client d'intégrations gérées, vous disposez des deux options suivantes pour utiliser les clés de chiffrement :

- Utilisez la clé de chiffrement gérée par défaut par les intégrations gérées.
- Fournissez un AWS KMS key que vous avez créé.

Pour plus d'informations sur le service AWS KMS, consultez [Service de gestion des clés \(KMS\)](#)

L'appel de l'[PutDefaultEncryptionConfiguration](#) API dans le guide de référence de l'API des intégrations gérées vous permet de mettre à jour l'option de clé de chiffrement que vous souhaitez utiliser. Par défaut, les intégrations gérées utilisent la clé de chiffrement gérée par défaut des intégrations gérées. Vous pouvez mettre à jour la configuration de votre clé de chiffrement à tout moment à l'aide de l'[PutDefaultEncryptionConfiguration](#) API.

En outre, l'appel de la commande [GetDefaultEncryptionConfiguration](#) API renvoie des informations sur la configuration de chiffrement du AWS compte dans la région par défaut ou spécifiée.

Techniques d'intégration

Vous trouverez ci-dessous la liste des types d'intégration :

Intégration des appareils connectés directement

Consultez [Bénéficiaire](#) les étapes à suivre pour intégrer un appareil connecté directement.

Intégration au hub

Consultez [Intégrez vos hubs aux intégrations gérées](#) les étapes à suivre pour intégrer le hub.

Intégration des appareils connectés au Hub

Consultez [Intégrez les appareils et faites-les fonctionner dans le hub](#) les étapes à suivre pour intégrer un appareil connecté au hub.

Cloud-to-cloud intégration des appareils

Consultez [Utiliser un connecteur C2C \(Cloud-to-Cloud\)](#) les étapes à suivre pour intégrer un appareil cloud d'un fournisseur de cloud tiers à des intégrations gérées.

Mise en service des appareils

Le provisionnement des appareils facilite le processus d'intégration des appareils, supervise l'ensemble du cycle de vie des appareils et établit un référentiel centralisé pour les informations sur les appareils, accessible aux autres aspects des intégrations gérées. Les intégrations gérées fournissent une interface unifiée pour gérer différents types d'appareils, en prenant en charge les appareils des clients propriétaires directement connectés via un kit de développement logiciel (SDK) ou des appareils commercial-off-the-shelf (COTS) reliés indirectement via un périphérique hub.

Dans les intégrations gérées, chaque appareil, quel que soit son type, possède un identifiant unique mondial appelé `managedThingId`. Cet identifiant est utilisé lors de l'intégration et de la gestion de l'appareil tout au long de son cycle de vie. Il est entièrement géré par des intégrations gérées et est propre à cet appareil spécifique pour toutes les intégrations gérées. Régions AWS Lorsque un appareil est initialement ajouté aux intégrations gérées, cet identifiant est créé et attaché à l'objet géré dans les intégrations gérées. Un objet géré est une représentation numérique du périphérique physique au sein d'intégrations gérées afin de refléter toutes les métadonnées de l'appareil physique. Pour les appareils tiers, ils peuvent avoir leur propre identifiant unique distinct spécifique à leur cloud tiers, en plus de celui `managedThingId` stocké dans les intégrations gérées représentant le périphérique physique.

Les appareils en cours de provisionnement peuvent avoir des statuts différents en fonction de l'étape du flux d'intégration dans laquelle ils se trouvent. La liste suivante décrit chaque statut de provisionnement :

- **ACTIVÉ** : L'appareil a été trouvé et la commande et le contrôle sont disponibles.
- **DÉCOUVERT** : L'appareil a été trouvé mais la commande et le contrôle ne sont pas encore disponibles.
- **NON ASSOCIÉ** : L'objet géré a été créé mais nécessite d'autres actions pour être découvert. Il n'est pas accessible depuis les contrôleurs AWS Cloud d'intégrations AWS IoT gérées (hubs)
- **PRE_ASSOCIATED** : L'objet géré a été créé et est prêt à être découvert automatiquement une fois allumé ou connecté. Il n'est pas accessible depuis les contrôleurs AWS Cloud d'intégrations AWS IoT gérées (hubs).
- **DELETE_IN_PROGRESS** : le processus de suppression asynchrone a démarré.
- **SUPPRIMÉ** : L'appareil a été supprimé du AWS Cloud.
- **ISOLÉ** : objet géré précédemment découvert ou activé qui n'est plus accessible. Par exemple, un appareil pour un cloud tiers dont les associations de connecteurs ont toutes été supprimées.

Le flux d'intégration suivant est destiné à approvisionner votre hub avec des intégrations gérées :

[Intégrez vos hubs aux intégrations gérées](#): configurez le fournisseur principal et les plugins spécifiques au protocole qui fonctionnent ensemble pour gérer l'authentification, la communication et la configuration des appareils.

Les flux d'intégration suivants sont fournis pour approvisionner les appareils connectés à votre hub avec des intégrations gérées :

- [Configuration simple \(SS\)](#): L'utilisateur final allume l'appareil IoT et scanne son code QR à l'aide de l'application du fabricant de l'appareil. L'appareil est ensuite inscrit dans le cloud d'intégrations gérées et se connecte au hub IoT.
- [Configuration sans contact \(ZTS\)](#): L'appareil est pré-associé en amont de la chaîne d'approvisionnement. Par exemple, au lieu que les utilisateurs finaux scannent le code QR de l'appareil, cette étape est effectuée plus tôt pour pré-associer l'appareil aux comptes clients.
- [Configuration guidée par l'utilisateur \(UGS\)](#): L'utilisateur final allume l'appareil et suit des étapes interactives pour l'intégrer aux intégrations gérées. Cela peut inclure d'appuyer sur un bouton du hub IoT, d'utiliser une application du fabricant de l'appareil ou d'appuyer sur les boutons du hub et de l'appareil. Vous pouvez utiliser cette méthode en cas d'échec de la configuration simple.

 Note

Le flux de travail de provisionnement des appareils dans les intégrations gérées est indépendant des exigences d'intégration d'un appareil. Les intégrations gérées fournissent une interface utilisateur rationalisée pour l'intégration et la gestion d'un appareil, quel que soit le type d'appareil ou le protocole de l'appareil.

Cycle de vie des appareils et des profils d'appareils

La gestion du cycle de vie de vos appareils et de leurs profils garantit la sécurité et le fonctionnement efficace de votre parc d'appareils.

Rubriques

- [Appareil](#)
- [Profil de l'appareil](#)

Appareil

Lors de l'intégration initiale, les intégrations gérées créent un jumeau numérique de votre appareil physique appelé Managed Thing. The Managed Thing possède un `managedThingID` identifiant unique global permettant d'identifier l'appareil dans les intégrations gérées dans toutes les régions. L'appareil s'associe au hub local lors du provisionnement pour une communication en temps réel avec des intégrations gérées ou un cloud tiers pour des appareils tiers. Un appareil est également associé à un propriétaire tel qu'identifié par le `owner` paramètre public APIs pour un objet géré tel que `GetManagedThing`. L'appareil est lié au profil d'appareil correspondant en fonction du type d'appareil.

Note

Un appareil physique peut contenir plusieurs enregistrements s'il est approvisionné plusieurs fois auprès de différents clients.

Le cycle de vie de l'appareil commence par la création de l'objet géré dans les intégrations gérées à l'aide de `CreateManagedThingAPI` et se termine lorsque le client supprime l'objet géré à l'aide de `DeleteManagedThingAPI`. Le cycle de vie d'un appareil est géré par le public suivant APIs :

- `CreateManagedThing`
- `ListManagedThings`
- `GetManagedThing`
- `UpdateManagedThing`
- `DeleteManagedThing`

Profil de l'appareil

Un profil d'appareil représente un type spécifique d'appareil tel qu'une ampoule ou une sonnette. Il est associé à un fabricant et contient les fonctionnalités de l'appareil. Le profil de l'appareil stocke le matériel d'authentification nécessaire pour les demandes de configuration de connectivité de l'appareil avec des intégrations gérées. Le matériel d'authentification utilisé est le code-barres de l'appareil.

Au cours du processus de fabrication de l'appareil, le fabricant peut enregistrer les profils de ses appareils avec des intégrations gérées. Cela permet au fabricant d'obtenir le matériel nécessaire pour les appareils à partir d'intégrations gérées lors des flux de travail d'intégration et de provisionnement. Les métadonnées du profil de l'appareil sont stockées sur le périphérique physique ou imprimées sur l'étiquette de l'appareil. Le cycle de vie du profil de l'appareil prend fin lorsque le fabricant le supprime dans les intégrations gérées.

Modèles de données

Un modèle de données représente la hiérarchie organisationnelle de la manière dont les données sont organisées au sein d'un système. De plus, il prend en charge end-to-end la communication sur l'ensemble de la mise en œuvre de votre appareil. Pour les intégrations gérées, deux modèles de données sont utilisés. Le modèle de données d'intégrations gérées et la AWS mise en œuvre du modèle de données Matter. Ils présentent des similitudes, mais présentent également des différences subtiles qui sont décrites dans les rubriques suivantes.

Pour les appareils tiers, les deux modèles de données sont utilisés pour la communication entre l'utilisateur final, les intégrations gérées et le fournisseur de cloud tiers. Pour traduire les messages tels que les commandes et les événements des appareils à partir des deux modèles de données, la fonctionnalité Cloud-to-Cloud Connector est exploitée.

Rubriques

- [Modèle de données d'intégrations gérées](#)
- [AWS mise en œuvre du modèle de données Matter](#)
- [Schémas de modèles de données](#)

Modèle de données d'intégrations gérées

Le modèle de données des intégrations gérées gère toutes les communications entre l'utilisateur final et les intégrations gérées.

Hiérarchie des appareils

Les éléments de capability données endpoint et sont utilisés pour décrire un appareil dans le modèle de données d'intégrations gérées.

endpoint

Le endpoint représente les interfaces logiques ou les services proposés par la fonctionnalité.

```
{
  "endpointId": { "type":"string" },
  "capabilities": Capability[]
}
```

Capability

Le capability représente les capacités de l'appareil.

```
{
  "$id": "string",           // Schema identifier (e.g. /schema-versions/
  capability/matter.OnOff@1.4)
  "name": "string",          // Human readable name
  "version": "string",        // e.g. 1.0
  "properties": Property[],
  "actions": Action[],
  "events": Event[]
}
```

Pour l'élément de capability données, trois éléments le composent : propertyaction, etevent. Ils peuvent être utilisés pour interagir avec l'appareil et le surveiller.

- Propriété : états enregistrés par l'appareil, tels que l'attribut de niveau de luminosité actuel d'une lampe à intensité variable.

```
• {
  "name":                // Property Name is outside of Property Entity
  "value": Value,         // value represented in any type e.g. 4, "A", []
  "lastChangedAt": Timestamp // ISO 8601 Timestamp upto milliseconds yyyy-MM-
  ddTHH:mm:ss.sssssZ
  "mutable": boolean,
  "retrievable": boolean,
  "reportable": boolean
}
```

- Action : tâches qui peuvent être effectuées, telles que le verrouillage d'une porte sur une serrure. Les actions peuvent générer des réponses et des résultats.

```
• {
  "name": { "$ref": "/schema-versions/definition/aws.name@1.0" }, //required
  "parameters": Map<String name, JSONNode value>,
  "responseCode": HTTPResponseCode,
  "errors": {
    "code": "string",
    "message": "string"
  }
}
```

- Événement : Il s'agit essentiellement d'un enregistrement des transitions entre États passées. Tout en `property` représentant les états actuels, les événements sont un journal du passé et incluent un compteur croissant de façon monotone, un horodatage et une priorité. Ils permettent de capturer les transitions d'état, ainsi que de modéliser les données, ce qui n'est pas facile à réaliser avec `property`.

```
{
  "name": { "$ref": "/schema-versions/definition/aws.name@1.0" },      //
  required
  "parameters": Map<String name, JSONNode value>
}
```

AWS mise en œuvre du modèle de données Matter

La AWS mise en œuvre du modèle Matter Data gère toutes les communications entre les intégrations gérées et les fournisseurs de cloud tiers.

Pour plus d'informations, consultez [Matter Data Model : Developer Resources](#).

Hiérarchie des appareils

Deux éléments de données sont utilisés pour décrire un appareil : `endpoint` et `cluster`.

endpoint

Le `endpoint` représente les interfaces logiques ou les services proposés par la fonctionnalité.

```
{
  "id": { "type": "string"},
  "clusters": Cluster[]
}
```

cluster

Le `cluster` représente les capacités de l'appareil.

```
{
  "id": "hexadecimalString",
  "revision": "string"          // optional
  "attributes": AttributeMap<String attributeId, JSONNode>,
  "commands": CommandMap<String commandId, JSONNode>,
}
```



```
"events": EventMap<String eventId, JsonNode>
}
```

Pour l'élément de `cluster` données, trois éléments le composent : `attributecommand`, `etevent`. Ils peuvent être utilisés pour interagir avec l'appareil et le surveiller.

- **Attribut** : états conservés par l'appareil, tels que l'attribut de niveau de luminosité actuel d'une lampe à intensité variable.

```
{
  "id" (hexadecimalString): (JsonNode) value
}
```

- **Commande** : tâches qui peuvent être effectuées, telles que le verrouillage d'une porte sur une serrure. Les commandes peuvent générer des réponses et des résultats.

```
{
  "id": {
    "fieldId": "fieldValue",
    ...
    "responseCode": HTTPResponseCode,
    "errors": {
      "code": "string",
      "message": "string"
    }
  }
}
```

- **Événement** : Il s'agit essentiellement d'un enregistrement des transitions entre États passées. Tout en `attributes` représentant les états actuels, les événements sont un journal du passé et incluent un compteur croissant de façon monotone, un horodatage et une priorité. Ils permettent de capturer les transitions d'état, ainsi que de modéliser les données, ce qui n'est pas facile à réaliser avec `attributes`.

```
{
  "id": {
    "fieldId": "fieldValue",
    ...
  }
}
```

Schémas de modèles de données

Les intégrations gérées prennent en charge deux types de schéma : la capacité et la définition du type. Si vous créez un modèle de données personnalisé, vous utilisez un document de schéma

JSON pour définir l'un ou l'autre type de schéma. Chaque document de schéma est limité à 50 000 caractères.

Schéma de capacité

Une capacité est un élément fondamental qui représente des fonctionnalités spécifiques au sein d'un point de terminaison. Grâce aux fonctionnalités, vous pouvez modéliser les états et les comportements des appareils à l'aide de propriétés, d'actions et d'événements. Les propriétés vous permettent de modéliser les attributs d'état de l'appareil de manière flexible avec n'importe quel type de données déclaratif. Les actions et les événements modélisent le comportement de l'appareil, notamment les commandes qu'il peut exécuter et les signaux qu'il peut signaler.

Ce qui suit montre la structure de haut niveau d'un schéma de capacités.

```
Capability
|
|-- Action
|-- Event
|-- Property
```

Action

Entité représentant une interaction avec une capacité d'un dispositif. Par exemple, faites sonner la cloche ou regardez qui est à la porte.

Événement

Entité représentant un événement issu d'une capacité d'un appareil. Un appareil peut envoyer un événement pour signaler un incident, une alarme ou une activité provenant d'un capteur, comme un coup à la porte.

Propriété

Entité représentant un attribut particulier dans l'état de l'appareil. Par exemple, une cloche sonne ou la lumière du porche est allumée

Chaque fonctionnalité inclut un identifiant unique avec un espace de noms, des informations de version et une description de son objectif. Le document de schéma utilise le versionnement sémantique pour maintenir la rétrocompatibilité tout en activant de nouvelles fonctionnalités.

Pour de plus amples informations, veuillez consulter [Schéma pour les définitions des capacités](#).

Schéma de définition de type

Une définition de type est un type de données structuré déclaratif qui permet la réutilisabilité et la composabilité. Il définit la manière dont les informations doivent être formatées et limitées. Utilisez des définitions de type pour créer des formats de données standardisés dans votre solution IoT.

Chaque définition de type inclut :

- Un identifiant unique avec espace de noms
- Title
- Description
- Propriétés qui définissent le formatage des données et les contraintes

Les types peuvent être soit de simples primitives, telles que des entiers ou des chaînes avec des limites définies, soit des structures complexes telles que des énumérations ou des objets personnalisés avec plusieurs champs. Les définitions de type utilisent la syntaxe du schéma JSON pour spécifier les contraintes, notamment les valeurs minimales et maximales, les longueurs de chaîne et les modèles autorisés.

Pour de plus amples informations, veuillez consulter [Schéma pour les définitions de type](#).

Schéma pour les définitions des capacités

Une capacité est documentée à l'aide d'un document JSON déclaratif qui fournit un contrat clair sur la manière dont la capacité doit fonctionner au sein du système.

Pour une capacité, les éléments obligatoires sont `$id`, `name`, `extrinsicId`, `extrinsicVersion` et au moins un élément dans au moins l'une des sections suivantes :

- `properties`
- `actions`
- `events`

Les éléments facultatifs d'une fonctionnalité sont `$ref`, `title`, `description`, `version`, `$defs` et `extrinsicProperties`. Pour une fonctionnalité, vous devez vous référer à `aws.capability`.

Les sections suivantes détaillent le schéma utilisé pour les définitions des capacités.

\$id (obligatoire)

L'élément \$id identifie la définition du schéma. Il doit suivre cette structure :

- Commencez par le préfixe `/schema-versions/` URI
- Inclure le type de `capability` schéma
- Utiliser une barre oblique (`/`) comme séparateur de chemin d'URI
- Incluez l'identité du schéma, avec des fragments séparés par des points (`.`)
- Utilisez le `@` caractère pour séparer l'ID et la version du schéma
- Terminez par la version `semver`, en utilisant des points (`.`) pour séparer les fragments de version

L'identité du schéma doit commencer par un espace de noms racine de 3 à 12 caractères, suivi d'un sous-espace et d'un nom facultatifs.

La version `semver` inclut une version MAJEURE (jusqu'à 3 chiffres), une version MINEURE (jusqu'à 3 chiffres) et une version PATCH optionnelle (jusqu'à 4 chiffres).

Note

Vous ne pouvez pas utiliser les espaces de noms `aws` réservés ou `matter`

Exemple Exemple \$id

```
/schema-version/capability/aws.Recording@1.0
```

\$ref

L'élément `$ref` fait référence à une capacité existante au sein du système. Il suit les mêmes contraintes que l'élément `$id`.

Note

Une définition de type ou une capacité doit exister avec la valeur fournie dans le `$ref` fichier.

Exemple Exemple \$ref

```
/schema-version/definition/aws.capability@1.0
```

nom (obligatoire)

L'élément `name` est une chaîne représentant le nom de l'entité dans le document de schéma. Il contient souvent des abréviations et doit respecter les règles suivantes :

- Ne contiennent que des caractères alphanumériques, des points (.), des barres obliques (/), des tirets (-) et des espaces
- Commencez par une lettre
- 64 caractères maximum

L'élément `name` est utilisé dans l'interface utilisateur et la documentation de la console Amazon Web Services.

Exemple Exemples de noms

```
Door Lock  
On/Off  
Wi-Fi Network Management  
PM2.5 Concentration Measurement  
RTCSessionController  
Energy EVSE
```

title

L'élément de titre est une chaîne descriptive de l'entité représentée par le document de schéma. Il peut contenir n'importe quel caractère et est utilisé dans la documentation. La longueur maximale du titre d'une fonctionnalité est de 256 caractères.

Exemple Exemples de titres

```
Real-time Communication (RTC) Session Controller  
Energy EVSE Capability
```

description

L'élément `description` fournit une explication détaillée de l'entité représentée par le document de schéma. Il peut contenir n'importe quel caractère et est utilisé dans la documentation. La longueur maximale d'une description de capacité est de 2 048 caractères.

Exemple Description de l'exemple

```
Electric Vehicle Supply Equipment (EVSE) is equipment used to charge an Electric Vehicle (EV) or Plug-In Hybrid Electric Vehicle.  
This capability provides an interface to the functionality of Electric Vehicle Supply Equipment (EVSE) management.
```

version

L'élément `version` est facultatif. Il s'agit d'une chaîne qui représente la version du document de schéma. Il présente les contraintes suivantes :

- Utilise le format `semver`, avec les fragments de version suivants séparés par `.` (points).
 - `MAJORversion`, maximum de 3 chiffres
 - `MINORversion`, maximum de 3 chiffres
 - `PATCHversion` (facultatif), maximum de 4 chiffres
- La longueur peut être comprise entre 3 et 12 caractères.

Exemple Exemples de versions

```
1.0
```

```
1.12
```

```
1.4.1
```

Utilisation des versions de fonctionnalités

Une capacité est une entité versionnée immuable. Toute modification est censée créer une nouvelle version. Le système utilise le versionnement sémantique au format `MAJOR.MINOR.PATCH`, où :

- La version MAJEURE augmente en cas de modifications d'API incompatibles avec les versions antérieures
- La version MINOR augmente lors de l'ajout de fonctionnalités de manière rétrocompatible
- La version du PATCH augmente lorsque des ajouts mineurs non impactants sont apportés à la fonctionnalité.

Les fonctionnalités dérivées des clusters Matter sont basées sur la version 1.4 et chaque version de Matter devrait être importée dans le système. Comme la version Matter consomme à la fois les niveaux MAJOR et MINOR de semver, les intégrations gérées ne peuvent utiliser que les versions PATCH.

Lorsque vous ajoutez des versions PATCH pour Matter, assurez-vous de tenir compte du fait que ce Matter utilise des révisions séquentielles. Toutes les versions de PATCH doivent être conformes à la révision documentée dans la spécification Matter et elles doivent être rétrocompatibles.

Pour résoudre les problèmes d'incompatibilité descendante, vous devez travailler avec la Connectivity Standards Alliance (CSA) pour résoudre les problèmes mentionnés dans la spécification et publier une nouvelle révision.

AWS-les fonctionnalités gérées ont été publiées avec une version initiale de 1.0. Avec ceux-ci, les trois niveaux de version peuvent être utilisés.

Version extrinsèque (obligatoire)

Il s'agit d'une chaîne représentant une version gérée en dehors du AWS IoT système. Pour les fonctionnalités de Matter, `extrinsicVersion` des mappages vers `revision`

Il est représenté sous la forme d'une valeur entière sous forme de chaîne et sa longueur peut être comprise entre 1 et 10 chiffres numériques.

Exemple Exemples de versions

7

1567

ExtrinsiciD (obligatoire)

L'`extrinsicId` élément représente un identifiant géré en dehors du système IoT d'Amazon Web Services. Pour les fonctionnalités de `MatterClusterId`, elles `attributeId` correspondent à `commandId`, `eventId`, `fieldId`, ou, selon le contexte.

Il `extrinsicId` peut s'agir d'un entier décimal à chaînes (1 à 10 chiffres) ou d'un entier hexadécimal à chaînes (préfixe `0x` ou `0X`, suivi de 1 à 8 chiffres hexadécimaux).

Note

Pour AWS, l'ID du fournisseur (VID) est `0x1577`, et pour Matter, il est égal à 0. Le système garantit que les schémas personnalisés n'utilisent pas ces schémas réservés VIDs aux fonctionnalités.

Exemple Exemples d'identifiants extrinsèques

```
0018
0x001A
0x15771002
```

\$defs

La `$defs` section est une carte des sous-schémas qui peuvent être référencés dans le document de schéma comme le permet le schéma JSON. Dans cette carte, la clé est utilisée dans les définitions de référence locales et la valeur fournit le schéma JSON.

Note

Le système impose uniquement qu'il `$defs` s'agit d'une carte valide et que chaque sous-schéma est un schéma JSON valide. Aucune règle supplémentaire n'est appliquée.

Respectez les contraintes suivantes lorsque vous travaillez avec des définitions :

- Utilisez uniquement des caractères adaptés aux URI dans les noms de définition
- Assurez-vous que chaque valeur est un sous-schéma valide

- Incluez un nombre quelconque de sous-schémas conformes aux limites de taille du document de schéma

Propriétés extrinsèques

L'extrinsicProperties élément contient un ensemble de propriétés définies dans un système externe mais conservées dans le modèle de données. En ce qui concerne les fonctionnalités de Matter, il correspond à différents éléments non modélisés ou partiellement modélisés au sein d'un cluster, d'un attribut, d'une commande ou d'un événement ZCL.

Les propriétés extrinsèques doivent respecter les contraintes suivantes :

- Les noms de propriété doivent être alphanumériques, sans espaces ni caractères spéciaux
- Les valeurs de propriété peuvent être n'importe quelle valeur de schéma JSON
- Maximum de 20 propriétés

Le système prend en charge divers extrinsicProperties access, notamment apiMaturity, cli, cliFunctionName, et autres. Ces propriétés facilitent l'ACL pour les transformations de modèles de données AWS (et vice versa).

Note

Les propriétés extrinsèques sont prises en charge pour les éléments action eventproperty,, et struct les champs d'une capacité, mais pas pour la capacité ou le cluster lui-même.

Propriétés extrinsèques prises en charge par le système

Le système suit les attributs de cluster, d'attribut, de commande ou d'événement non modélisés ou partiellement modélisés suivants, comme extrinsicProperties lors de transformations vers ou depuis ZCL :

access

Chaque objet d'accès contient les éléments suivants :

- op- Fonctionnement modélisé comme un enum avec des valeurs :read,write, ou invoke

- `privilege`- Privilège modélisé comme un enum avec des valeurs : `viewproxy_view`, `operatemanage`, ou `administer`
- `role`- Chaîne illimitée représentant un rôle d'opérateur

`apiMaturity`

Chaîne simple illimitée représentant le niveau de maturité. Ceci est modélisé dans ZCL sous la forme d'un enum avec les valeurs suivantes : `stable`, `provisional`, ou `internal deprecated`

`side`

Modélisé sous la forme d'une énumération avec les valeurs suivantes : `either`, et `server client`

Propriétés booléennes

Les propriétés suivantes sont des indicateurs booléens :

- `isFabricScoped`
- `isFabricSensitive`
- `mustUseAtomicWrite`
- `mustUseTimedInvoke`

Propriétés des chaînes

Les propriétés suivantes sont représentées sous forme de chaînes illimitées :

- `cli`
- `cliFunctionName`
- `functionName`
- `group`
- `introducedIn`
- `manufacturerCode`
- `noDefaultImplementation`
- `presentIf`
- `priority`
- `removedIn`
- `reportableChange`

- `reportMinInterval`
- `reportMaxInterval`
- `restriction`
- `storage`

Considérations relatives aux transformations

Pour les transformations ZCL, `extrinsicProperties` elles sont stockées dans une carte sans traitement. Les schémas personnalisés qui utilisent la découverte ne sont pas soumis à une transformation ZCL. Toutefois, si vous prévoyez d'implémenter des transformations ZCL pour des schémas personnalisés à l'avenir, vous devez modéliser tous les types de chaînes simples illimités `extrinsicProperties` et définir des contraintes telles que les énumérations, les modèles (regex) et la longueur. Cette préparation garantit une manipulation correcte de ces propriétés lors de la transformation.

En revanche, AWS les transformations entre connecteurs ne `extrinsicProperties` sont pas incluses du tout, car ces détails ne sont pas obligatoires dans le format du connecteur.

Propriétés

Les propriétés représentent les états de la fonctionnalité gérés par le périphérique. Chaque état est défini comme une paire clé-valeur, où la clé décrit le nom de l'état et la valeur décrit la définition de l'état.

Lorsque vous travaillez avec des propriétés, respectez les contraintes suivantes :

- Utilisez uniquement des caractères alphanumériques dans les noms de propriétés, sans espaces ni caractères spéciaux
- Incluez un nombre quelconque de propriétés conformes aux limites de taille du document de schéma

Utilisation des propriétés

Une propriété au sein d'une fonctionnalité est un élément fondamental qui représente un état spécifique d'un appareil alimenté par des intégrations gérées. Il représente l'état ou la configuration actuelle de l'appareil. En normalisant la façon dont ces propriétés sont définies et structurées, les systèmes domotiques garantissent que les appareils de différents fabricants peuvent communiquer efficacement, créant ainsi une expérience fluide et interopérable.

Pour une propriété de capacité, les éléments obligatoires sont `extrinsicId` et `value`. Les éléments facultatifs d'une propriété de capacité sont `description`, `retrievable`, `mutable`, `reportable` et `extrinsicProperties`.

Valeur

Une structure illimitée qui permet aux constructeurs de définir toutes les contraintes conformes au schéma JSON pour définir le type de données de cette propriété.

Lorsque vous définissez des valeurs, respectez les contraintes suivantes :

- Pour les types simples, utilisez `type` et toute autre contrainte de schéma JSON native telle que `maxLength` ou `maximum`
- Pour les types composites `oneOf`, utilisez `allOf`, ou `anyOf`. Le système ne prend pas en charge le mot clé `not`
- Pour faire référence à un type global, utilisez-le `$ref` avec une référence détectable valide
- Pour la nullabilité, suivez la définition du schéma de type OpenAPI en fournissant à l'attribut `nullable` un indicateur booléen (`true` si `null` est une valeur autorisée)

Exemple :

```
{
  "$ref": "/schema-versions/definition/matter.uint16@1.4",
  "nullable": true,
  "maximum": 4096
}
```

Récupérable

Un booléen décrivant si l'état est lisible ou non.

L'aspect lisibilité de l'état dépend de la mise en œuvre de la fonctionnalité par le périphérique.

L'appareil décide si un état donné est lisible ou non. Cet aspect de l'état n'est pas encore pris en charge pour être signalé dans le rapport de capacité et n'est donc pas appliqué au sein du système.

Exemple : `true` ou `false`

Mutable

Un booléen décrivant si l'état est inscriptible ou non.

L'aspect inscriptible de l'état est reporté à la mise en œuvre de la capacité par le périphérique.

L'appareil décide si un état donné est inscriptible ou non. Cet aspect de l'état n'est pas encore pris en charge pour être signalé dans le rapport de capacité et n'est donc pas appliqué au sein du système.

Exemple : `true` ou `false`

À signaler

Un booléen décrivant si l'état est signalé par le périphérique lorsqu'il y a un changement d'état.

L'aspect déclarable de l'état est reporté à la mise en œuvre de la fonctionnalité par l'appareil.

L'appareil décide si un état donné est déclarable ou non. Cet aspect de l'état n'est pas encore pris en charge pour être signalé dans le rapport de capacité et n'est donc pas appliqué au sein du système.

Exemple : `true` ou `false`

Actions

Les actions sont des opérations gérées par un schéma qui suivent un modèle de demande-réponse. Chaque action représente une opération mise en œuvre par l'appareil.

Respectez les contraintes suivantes lors de la mise en œuvre des actions :

- N'incluez que des actions uniques dans le tableau d'actions
- Incluez autant d'actions que vous le souhaitez, conformément aux limites de taille du document de schéma

Utilisation des actions

Une action est un moyen standardisé d'interagir avec les fonctionnalités des appareils et de les contrôler dans un système d'intégration géré. Il représente une commande ou une opération spécifique qui peut être exécutée sur un appareil, avec un format structuré pour modéliser les paramètres de demande ou de réponse nécessaires. Ces actions servent de pont entre les intentions des utilisateurs et le fonctionnement des appareils, permettant ainsi un contrôle cohérent et fiable des différents types d'appareils intelligents.

Pour une action, les éléments obligatoires sont `name` et `extrinsicId`. Les éléments facultatifs sont `description`, `extrinsicProperties`, `request` et `response`.

Description

La limite de longueur maximale de la description est de 1 536 caractères.

Demande

La section de demande est facultative et peut être omise s'il n'y a aucun paramètre de demande. En cas d'omission, le système prend en charge l'envoi d'une demande sans aucune charge utile en utilisant simplement le nom de. Action Ceci est utilisé pour des actions simples, comme allumer ou éteindre une lumière.

Les actions complexes nécessitent des paramètres supplémentaires. Par exemple, une demande de diffusion de séquences vidéo peut inclure des paramètres concernant le protocole de diffusion à utiliser ou l'opportunité d'envoyer le flux à un périphérique d'affichage spécifique.

Pour une demande d'action, l'élément obligatoire est `parameters`. Les éléments facultatifs sont `description`, `extrinsicId`, et `extrinsicProperties`.

Description de la demande

La description suit le même format que celui de la section 3.5, avec une longueur maximale de 2 048 caractères.

Réponse

Dans les intégrations gérées, pour toute demande d'action envoyée via l'[SendManagedThingCommandAPI](#), la demande atteint l'appareil et attend une réponse asynchrone. La réponse à l'action définit la structure de cette réponse.

Pour une demande d'action, l'élément obligatoire est `parameters`. Les éléments facultatifs sont `name`, `description`, `extrinsicId`, `extrinsicProperties`, `errors` et `responseCode`.

Description de la réponse

La description suit le même format que [description](#), et sa longueur maximale est de 2 048 caractères.

Nom de la réponse

Le nom suit le même format que [nom \(obligatoire\)](#), avec les détails supplémentaires suivants :

- Le nom classique d'une réponse est dérivé en l'ajoutant Response au nom de l'action.
- Si vous souhaitez utiliser un autre nom, vous pouvez le fournir dans cet `name` élément. Si un `name` est fourni dans la réponse, cette valeur est plus prioritaire que le nom conventionnel.

Erreurs

Un tableau illimité de messages uniques fournis dans la réponse, en cas d'erreur lors du traitement de la demande.

Contraintes :

- Un élément de message est déclaré en tant qu'objet JSON avec les champs suivants :
 - code: chaîne contenant des caractères alphanumériques et _ (traits de soulignement), d'une longueur comprise entre 1 et 64 caractères
 - message: valeur de chaîne illimitée

Exemple Exemple de message d'erreur

```
"errors": [  
  {  
    "code": "AD_001",  
    "message": "Unable to receive signal from the sensor. Please check connection  
with the sensor."  
  }  
]
```

Code de réponse

Un code entier indiquant comment la demande a été traitée. Nous recommandons que le code de l'appareil renvoie un code utilisant la spécification du code d'état de réponse du serveur HTTP afin de garantir l'uniformité au sein du système.

Contrainte : valeur entière comprise entre 100 et 599.

Paramètres de demande ou de réponse

La section des paramètres est définie comme une carte des paires de noms et de sous-schémas. Un certain nombre de paramètres peuvent être définis dans les paramètres de demande, s'ils peuvent être intégrés au document de schéma.

Les noms de paramètres ne peuvent contenir que des caractères alphanumériques. Les espaces ou tout autre caractère ne sont pas autorisés.

champ de paramètres

Les éléments obligatoires d'un `parameter` sont `extrinsicId` et `value`. Les éléments facultatifs sont `description` et `extrinsicProperties`.

L'élément de description suit le même format que [description](#), avec une longueur maximale de 1024 caractères.

`extrinsicId` et `extrinsicProperties` dérogations

`extrinsicId` et `extrinsicProperties` suivent le même format que [ExtrinsicID \(obligatoire\)](#) et [Propriétés extrinsèques](#), avec ces informations supplémentaires :

- Si un `extrinsicId` est fourni dans la demande ou la réponse, cette valeur est plus prioritaire que la valeur fournie au niveau de l'action. Le système doit d'abord utiliser `request/response` le niveau ; en cas d'absence, utiliser le niveau d'action `extrinsicId`
- Si `extrinsicProperties` elles sont fournies dans la demande ou la réponse, ces propriétés ont une priorité supérieure à la valeur fournie au niveau de l'action. Le système doit prendre le niveau d'action `extrinsicProperties` et remplacer les paires clé-valeur fournies au niveau `request/response extrinsicProperties`

Exemple Exemple de remplacement d'ExtrinsicID et d'ExtrinsicProperties

```
{
  "name": "ToggleWithEffect",
  "extrinsicId": "0x0001",

  "extrinsicProperties": {
    "apiMaturity": "provisional",
    "introducedIn": "1.2"
  },
  "request": {
    "extrinsicProperties": {
      "apiMaturity": "stable",
      "manufacturerCode": "XYZ"
    },
    "parameters": {
      ...
    }
  },
  "response": {
```



```
    "extrinsicProperties": {
      "noDefaultImplementation": true
    },
    "parameters": {
      {
        ...
      }
    }
  }
}
```

Dans l'exemple ci-dessus, les valeurs effectives de la demande d'action seraient les suivantes :

```
# effective request
"name": "ToggleWithEffect",
"extrinsicId": "0x0001",
"extrinsicProperties": {
  "apiMaturity": "stable",
  "introducedIn": "1.2"
  "manufacturerCode": "XYZ"
},
"parameters": {
  ...
}

# effective response
"name": "ToggleWithEffectResponse",
"extrinsicId": "0x0001",
"extrinsicProperties": {
  "apiMaturity": "provisional",
  "introducedIn": "1.2"
  "noDefaultImplementation": true
},
"parameters": {
  ...
}
```

Actions intégrées

Pour toutes les fonctionnalités, vous pouvez effectuer des actions personnalisées à l'aide des mots clés `ReadState` et `UpdateState`. Ces deux mots clés d'action agiront sur les propriétés de la fonctionnalité définies dans le modèle de données.

ReadState

Envoie une commande au managedThing pour lire les valeurs de ses propriétés d'état.

ReadState À utiliser pour forcer la mise à jour de l'état de l'appareil.

UpdateState

Envoie une commande pour mettre à jour certaines propriétés.

Il peut être utile de forcer la synchronisation de l'état d'un appareil dans les scénarios suivants :

1. L'appareil est resté hors ligne pendant un certain temps et n'émettait aucun événement.
2. L'appareil vient d'être provisionné et aucun état n'est encore maintenu dans le cloud.
3. L'état de l'appareil n'est pas synchronisé avec l'état réel de l'appareil.

ReadState exemples

Vérifiez si le voyant est allumé ou éteint à l'aide de l'[SendManagedThingCommandAPI](#) :

```
{
  "Endpoints": [
    {
      "endpointId": "1",
      "capabilities": [
        {
          "id": "aws.OnOff",
          "name": "On/Off",
          "version": "1",
          "actions": [
            {
              "name": "ReadState",
              "parameters": {
                "propertiesToRead": [ "OnOff" ]
              }
            }
          ]
        }
      ]
    }
  ]
}
```

Lisez toutes les propriétés d'état de la `matter.OnOff` fonctionnalité :

```
{
  "Endpoints": [
    {
      "endpointId": "1",
      "capabilities": [
        {
          "id": "aws.OnOff",
          "name": "On/Off",
          "version": "1",
          "actions": [
            {
              "name": "ReadState",
              "parameters": {
                "propertiesToRead": [ "*" ]
                // Use the wildcard operator to read ALL state properties for a
                capability
              }
            }
          ]
        }
      ]
    }
  ]
}
```

UpdateState exemple

Changez le `OnTime` pour une lampe à l'aide de l'[SendManagedThingCommand](#) API :

```
{
  "Endpoints": [
    {
      "endpointId": "1",
      "capabilities": [
        {
          "id": "matter.OnOff",
          "name": "On/Off",
          "version": "1",
          "actions": [
            {
              "name": "UpdateState",
```

```
        "parameters": {  
            "OnTime": 5  
        }  
    }  
]  
}  
]  
}  
]  
}  
]  
}
```

Événements

Les événements sont des signaux unidirectionnels gérés par un schéma mis en œuvre par le dispositif.

Implémentez les événements en fonction des contraintes suivantes :

- Inclure uniquement les événements uniques dans le tableau des événements
- Incluez un nombre quelconque d'événements correspondant aux limites de taille du document de schéma

Événements dans les systèmes d'intégration gérés

Utilisation des événements

Un événement est un moyen standardisé d'apprendre de manière proactive les modifications apportées à un appareil ou à son environnement. Il représente un événement modélisé que l'appareil enverra au cloud pour fournir des informations sur un élément modifié sur l'appareil ou détecté dans son environnement. Comme ces événements sont modélisés, les clients peuvent les utiliser dans un flux de contrôle pour réagir à des événements spécifiques et aux informations qu'ils contiennent.

Pour un événement, les éléments obligatoires sont `name` et `extrinsicId`. Les éléments facultatifs sont `description`, `extrinsicProperties`, et `request`.

Description

La description suit le même format que celui décrit dans [description](#), avec une longueur maximale de 512 caractères.

Demande

La `request` section est facultative et peut être omise s'il n'existe aucun paramètre de demande. En cas d'omission, le système permet à un appareil d'envoyer une demande d'événement sans aucune charge utile en utilisant simplement le nom de l'événement. Ceci est utilisé dans des cas simples, tels qu'une défaillance du capteur d'une pompe ou si l'alarme est désactivée sur un détecteur de fumée ou de monoxyde de carbone.

Les actions complexes nécessitent des paramètres supplémentaires. Par exemple, une demande de diffusion de séquences vidéo peut inclure des paramètres concernant le protocole de diffusion à utiliser ou l'opportunité d'envoyer le flux à un périphérique d'affichage spécifique.

Pour une demande d'événement, l'élément obligatoire est `parameters`. Il n'y a aucun élément optionnel.

Réponse

Les réponses aux événements ne sont actuellement pas prises en charge.

Schéma pour les définitions de type

Les sections suivantes détaillent le schéma utilisé pour les définitions de type.

\$id

L'élément `$id` identifie la définition du schéma. Il doit suivre cette structure :

- Commencez par le préfixe `/schema-versions/` URI
- Inclure le type de définition schéma
- Utiliser une barre oblique (`/`) comme séparateur de chemin d'URI
- Incluez l'identité du schéma, avec des fragments séparés par des points (`.`)
- Utilisez le `@` caractère pour séparer l'ID et la version du schéma
- Terminez par la version semver, en utilisant des points (`.`) pour séparer les fragments de version

L'identité du schéma doit commencer par un espace de noms racine de 3 à 12 caractères, suivi d'un sous-espace et d'un nom facultatifs.

La version semver inclut une version MAJEURE (jusqu'à 3 chiffres), une version MINEURE (jusqu'à 3 chiffres) et une version PATCH optionnelle (jusqu'à 4 chiffres).

 Note

Vous ne pouvez pas utiliser les espaces de noms aws réservés ou matter

Exemple Exemple \$id

```
/schema-version/capability/aws.Recording@1.0
```

\$ref

L'élément \$ref fait référence à une définition de type existante dans le système. Il suit les mêmes contraintes que l'\$id élément.

 Note

Une définition de type ou une capacité doit exister avec la valeur fournie dans le \$ref fichier.

Exemple Exemple \$ref

```
/schema-version/definition/aws.capability@1.0
```

name

L'élément name est une chaîne représentant le nom de l'entité dans le document de schéma. Il contient souvent des abréviations et doit respecter les règles suivantes :

- Ne contiennent que des caractères alphanumériques, des points (.), des barres obliques (/), des tirets (-) et des espaces
- Commencez par une lettre
- 192 caractères maximum

L'élément name est utilisé dans l'interface utilisateur et la documentation de la console Amazon Web Services.

Exemple Exemples de noms

```
Door Lock
```

```
On/Off
Wi-Fi Network Management
PM2.5 Concentration Measurement
RTCSessionController
Energy EVSE
```

title

L'élément de titre est une chaîne descriptive de l'entité représentée par le document de schéma. Il peut contenir n'importe quel caractère et est utilisé dans la documentation.

Example Exemples de titres

```
Real-time Communication (RTC) Session Controller
Energy EVSE Capability
```

description

L'élément de description fournit une explication détaillée de l'entité représentée par le document de schéma. Il peut contenir n'importe quel caractère et est utilisé dans la documentation.

Example Description de l'exemple

```
Electric Vehicle Supply Equipment (EVSE) is equipment used to charge an Electric
Vehicle (EV) or Plug-In Hybrid Electric Vehicle.
    This capability provides an interface to the functionality of Electric
Vehicle Supply Equipment (EVSE) management.
```

ExtrinsicId

L'élément `extrinsicId` représente un identifiant géré en dehors du système IoT d'Amazon Web Services. Pour les fonctionnalités de `MatterClusterId`, les `attributeId` correspondent à `commandId`, `eventId`, `fieldId`, ou, selon le contexte.

Il `extrinsicId` peut s'agir d'un entier décimal à chaînes (1 à 10 chiffres) ou d'un entier hexadécimal à chaînes (préfixe 0x ou 0X, suivi de 1 à 8 chiffres hexadécimaux).

Note

Pour AWS, l'ID du fournisseur (VID) est 0x1577, et pour Matter, il est égal à 0. Le système garantit que les schémas personnalisés n'utilisent pas ces schémas réservés VIDs aux fonctionnalités.

Exemple Exemples d'identifiants extrinsèques

```
0018
0x001A
0x15771002
```

Propriétés extrinsèques

L'`extrinsicProperties` élément contient un ensemble de propriétés définies dans un système externe mais conservées dans le modèle de données. En ce qui concerne les fonctionnalités de Matter, il correspond à différents éléments non modélisés ou partiellement modélisés au sein d'un cluster, d'un attribut, d'une commande ou d'un événement ZCL.

Les propriétés extrinsèques doivent respecter les contraintes suivantes :

- Les noms de propriété doivent être alphanumériques, sans espaces ni caractères spéciaux
- Les valeurs de propriété peuvent être n'importe quelle valeur de schéma JSON
- Maximum de 20 propriétés

Le système prend en charge divers `extrinsicProperties` access, notamment `apiMaturity`, `cli`, `cliFunctionName`, et autres. Ces propriétés facilitent l'ACL pour les transformations de modèles de données AWS (et vice versa).

Note

Les propriétés extrinsèques sont prises en charge pour les éléments `action`, `eventproperty`, et `struct` les champs d'une capacité, mais pas pour la capacité ou le cluster lui-même.

Propriétés extrinsèques prises en charge par le système

Le système suit les attributs de cluster, d'attribut, de commande ou d'événement non modélisés ou partiellement modélisés suivants, comme `extrinsicProperties` lors de transformations vers ou depuis ZCL :

`access`

Chaque objet d'accès contient les éléments suivants :

- `op`- Fonctionnement modélisé comme un enum avec des valeurs : `read`, `write`, ou `invoke`
- `privilege`- Privilège modélisé comme un enum avec des valeurs : `viewproxy_view`, `operatemanage`, ou `administer`
- `role`- Chaîne illimitée représentant un rôle d'opérateur

`apiMaturity`

Chaîne simple illimitée représentant le niveau de maturité. Ceci est modélisé dans ZCL sous la forme d'un enum avec les valeurs suivantes : `stable`, `provisional`, ou `internal deprecated`

`side`

Modélisé sous la forme d'une énumération avec les valeurs suivantes : `either`, et `server client`

Propriétés booléennes

Les propriétés suivantes sont des indicateurs booléens :

- `isFabricScoped`
- `isFabricSensitive`
- `mustUseAtomicWrite`
- `mustUseTimedInvoke`

Propriétés des chaînes

Les propriétés suivantes sont représentées sous forme de chaînes illimitées :

- `cli`
- `cliFunctionName`
- `functionName`
- `group`

- `introducedIn`
- `manufacturerCode`
- `noDefaultImplementation`
- `presentIf`
- `priority`
- `removedIn`
- `reportableChange`
- `reportMinInterval`
- `reportMaxInterval`
- `restriction`
- `storage`

Considérations relatives aux transformations

Pour les transformations ZCL, `extrinsicProperties` elles sont stockées dans une carte sans traitement. Les schémas personnalisés qui utilisent la découverte ne sont pas soumis à une transformation ZCL. Toutefois, si vous prévoyez d'implémenter des transformations ZCL pour des schémas personnalisés à l'avenir, vous devez modéliser tous les types de chaînes simples illimités `extrinsicProperties` et définir des contraintes telles que les énumérations, les modèles (regex) et la longueur. Cette préparation garantit une manipulation correcte de ces propriétés lors de la transformation.

En revanche, AWS les transformations entre connecteurs ne `extrinsicProperties` sont pas incluses du tout, car ces détails ne sont pas obligatoires dans le format du connecteur.

Création et utilisation de définitions de type dans les documents de schéma de capacités

Tous les éléments des schémas sont convertis en définitions de type. Ces définitions de type sont soit des définitions de type primitives (telles que des booléens, des chaînes, des nombres), soit des définitions de type avec espace de noms (définitions de type créées à partir de définitions de types primitives pour des raisons de commodité).

Lorsque vous définissez un schéma personnalisé, vous pouvez utiliser à la fois des définitions primitives et des définitions de type d'espace de noms.

Table des matières

- [Définitions de types primitifs](#)
 - [Booléens](#)
 - [Support de type entier](#)
 - [Nombres](#)
 - [Chaînes](#)
 - [Valeurs null](#)
 - [Arrays \(tableaux\)](#)
 - [Objets](#)
- [Définitions de types avec espace de noms](#)
 - [Types matter](#)
 - [Types aws](#)
 - [Définition du type de bitmap](#)
 - [Définition du type Enum](#)

Définitions de types primitifs

Les définitions de types primitives sont les éléments de base de toutes les définitions de type définies dans les intégrations gérées. Toutes les définitions d'espaces de noms, y compris les définitions de type personnalisées, sont converties en une définition de type primitive via le `$ref` mot-clé ou le `type` mot-clé.

Tous les types primitifs sont nullable à l'aide du `nullable` mot-clé, et vous pouvez identifier tous les types primitifs à l'aide du `type` mot-clé.

Booléens

Les types booléens prennent en charge les valeurs par défaut.

Définition de l'exemple :

```
{
  "type" : "boolean",
  "default" : "false",
  "nullable" : true
}
```

Support de type entier

Les types entiers prennent en charge les éléments suivants :

- `default values`
- `maximum values`
- `minimum values`
- `exclusiveMaximum values`
- `exclusiveMinimum values`
- `multipleOf values`

Si la valeur `x` est en cours de validation, les conditions suivantes doivent être vraies :

- $x \geq \text{minimum}$
- $x > \text{exclusiveMinimum}$
- $x < \text{exclusiveMaximum}$

Note

Les nombres dont la partie fractionnaire est nulle sont considérés comme des entiers, mais les nombres à virgule flottante sont rejetés.

```
1.0 // Schema-Compliant
3.1415926 // NOT Schema-Compliant
```

Bien que vous puissiez spécifier `minimum` les deux `exclusiveMinimum` et/ou `maximum` les deux `exclusiveMaximum`, nous vous déconseillons d'utiliser les deux simultanément.

Exemples de définitions :

```
{
  "type" : "integer",
  "default" : 2,
  "nullable" : true,
  "maximum" : 10,
```

```
"minimum" : 0,  
"multipleOf": 2  
}
```

Définition alternative :

```
{  
  "type" : "integer",  
  "default" : 2,  
  "nullable" : true,  
  "exclusiveMaximum" : 11,  
  "exclusiveMinimum" : -1,  
  "multipleOf": 2  
}
```

Nombres

Utilisez le type numérique pour tous les types numériques, y compris les entiers et les nombres à virgule flottante.

Les types de numéros prennent en charge les éléments suivants :

- `default` values
- `maximum` values
- `minimum` values
- `exclusiveMaximum` values
- `exclusiveMinimum` values
- `multipleOf` valeurs. Le multiple peut être un nombre à virgule flottante.

Si la valeur `x` est en cours de validation, les conditions suivantes doivent être vraies :

- $x \geq \text{minimum}$
- $x > \text{exclusiveMinimum}$
- $x < \text{exclusiveMaximum}$

Bien que vous puissiez spécifier `minimum` les deux `exclusiveMinimum` et/ou `maximum` les deux `exclusiveMaximum`, nous vous déconseillons d'utiliser les deux simultanément.

Exemples de définitions :

```
{
  "type" : "number",
  "default" : 0.4,
  "nullable" : true,
  "maximum" : 10.2,
  "minimum" : 0.2,
  "multipleOf": 0.2
}
```

Définition alternative :

```
{
  "type" : "number",
  "default" : 0.4,
  "nullable" : true,
  "exclusiveMaximum" : 10.2,
  "exclusiveMinimum" : 0.2,
  "multipleOf": 0.2
}
```

Chaînes

Les types de chaînes prennent en charge les éléments suivants :

- `default` values
- les contraintes de longueur (doivent être des nombres non négatifs), y compris les valeurs `maxLength` et `minLength`
- `pattern` valeurs pour les expressions régulières

Lorsque vous définissez des expressions régulières, la chaîne est valide si l'expression correspond à un point quelconque de la chaîne. Par exemple, l'expression régulière `p` correspond à n'importe quelle chaîne contenant un `p`, telle que « apple », et pas seulement à la chaîne « p ». Pour plus de clarté, nous vous recommandons d'entourer les expressions régulières de `^...$` (par exemple, `^p$`), sauf si vous avez une raison spécifique de ne pas le faire.

Définition de l'exemple :

```
{
```

```
"type" : "string",
"default" : "defaultString",
"nullable" : true,
"maxLength": 10,
"minLength": 1,
"pattern" : "^[0-9a-fA-F]{2})+$"
}
```

Valeurs null

Les types nuls n'acceptent qu'une seule valeur : `null`.

Définition de l'exemple :

```
{ "type": "null" }
```

Arrays (tableaux)

Les types de tableaux prennent en charge les éléments suivants :

- `default`— une liste qui sera utilisée comme valeur par défaut.
- `items`— Définition du type JSON imposée à tous les éléments du tableau.
- Contraintes de longueur (doit être un nombre non négatif)
 - `minItems`
 - `maxItems`
- `pattern` valeurs pour Regex
- `uniqueItems`— un booléen indiquant si les éléments du tableau doivent être uniques
- `prefixItems`— un tableau dans lequel chaque élément est un schéma correspondant à chaque index du tableau du document. C'est-à-dire un tableau où le premier élément valide le premier élément du tableau d'entrée, le deuxième élément valide le deuxième élément du tableau d'entrée, et ainsi de suite.

Définition de l'exemple :

```
{
  "type": "array",
  "default": ["1", "2"],
  "items" : {
```

```

    "type": "string",
    "pattern": "^[a-zA-Z0-9_ -/]+$"
  },
  "minItems" : 1,
  "maxItems": 4,
  "uniqueItems" : true,
}

```

Exemples de validation de tableaux :

```

//Examples:
["1", "2", "3", "4"] // Schema-Compliant
[] // NOT Schema-Compliant: minItems=1
["1", "1"] // NOT Schema-Compliant: uniqueItems=true
["{"] // NOT Schema-Compliant: Does not match the RegEx pattern.

```

Définition alternative utilisant la validation des tuples :

```

{
  "type": "array",
  "prefixItems": [
    { "type": "number" },
    { "type": "string" },
    { "enum": ["Street", "Avenue", "Boulevard"] },
    { "enum": ["NW", "NE", "SW", "SE"] }
  ]
}

//Examples:
[1600, "Pennsylvania", "Avenue", "NW"] // Schema-Compliant

// And, by default, it's also okay to add additional items to end:
[1600, "Pennsylvania", "Avenue", "NW", "Washington"] // Schema-Compliant

```

Objets

Les types d'objets prennent en charge les éléments suivants :

- Contraintes immobilières
 - **properties**— Définissez les propriétés (paires clé-valeur) d'un objet à l'aide du **properties** mot-clé. La valeur de **properties** est un objet, où chaque clé est le nom d'une propriété

et chaque valeur est un schéma utilisé pour valider cette propriété. Toute propriété qui ne correspond à aucun des noms de propriété du `properties` mot clé est ignorée par ce mot clé.

- `required`— Par défaut, les propriétés définies par le `properties` mot clé ne sont pas obligatoires. Vous pouvez toutefois fournir une liste des propriétés requises à l'aide du `required` mot clé. Le `required` mot clé prend un tableau de zéro ou plusieurs chaînes. Chacune de ces chaînes doit être unique.
- `propertyNames`— Ce mot clé permet de contrôler le RegEx modèle des noms de propriétés. Par exemple, vous souhaitez peut-être faire en sorte que toutes les propriétés d'un objet portent des noms conformes à une convention spécifique.
- `patternProperties`— Cela fait correspondre les expressions régulières aux schémas. Si le nom d'une propriété correspond à l'expression régulière donnée, la valeur de la propriété doit être validée par rapport au schéma correspondant. Par exemple, utilisez `patternProperties` pour spécifier qu'une valeur doit correspondre à un schéma particulier, en fonction d'un type particulier de nom de propriété.
- `additionalProperties`— Ce mot clé contrôle la façon dont les propriétés supplémentaires sont gérées. Les propriétés supplémentaires sont des propriétés dont les noms ne figurent pas dans le mot clé `properties` ou qui correspondent à l'une des expressions régulières de `patternProperties`. Par défaut, les propriétés supplémentaires sont autorisées. La définition de ce champ sur `false` signifie qu'aucune propriété supplémentaire n'est autorisée.
- `unevaluatedProperties`— Ce mot clé est similaire à `additionalProperties` sauf qu'il peut reconnaître les propriétés déclarées dans les sous-schémas. `unevaluatedProperties` fonctionne en collectant toutes les propriétés validées avec succès lors du traitement des schémas et en les utilisant comme liste de propriétés autorisées. Cela vous permet d'effectuer des tâches plus complexes, telles que l'ajout conditionnel de propriétés. Reportez-vous à l'exemple ci-dessous pour plus de détails.
- `anyOf`— La valeur de ce mot clé DOIT être un tableau non vide. Chaque élément du tableau DOIT être un schéma JSON valide. Une instance est validée avec succès par rapport à ce mot clé si elle est validée avec succès par rapport à au moins un schéma défini par la valeur de ce mot clé.
- `oneOf`— La valeur de ce mot clé DOIT être un tableau non vide. Chaque élément du tableau DOIT être un schéma JSON valide. Une instance est validée avec succès par rapport à ce mot clé si elle est validée avec succès par rapport à exactement un schéma défini par la valeur de ce mot clé.

Exemple de obligatoire :

```
{
```

```
"type": "object",
"required": ["test"]
}

// Schema Compliant
{
  "test": 4
}

// NOT Schema Compliant
{}
```

PropertyNames exemple :

```
{
  "type": "object",
  "propertyNames": {
    "pattern": "^[A-Za-z_][A-Za-z0-9_]*$"
  }
}

// Schema Compliant
{
  "_a_valid_property_name_001": "value"
}

// NOT Schema Compliant
{
  "001 invalid": "value"
}
```

PatternProperties exemple :

```
{
  "type": "object",
  "patternProperties": {
    "^S_": { "type": "string" },
    "^I_": { "type": "integer" }
  }
}

// Schema Compliant
{ "S_25": "This is a string" }
```

```
{ "I_0": 42 }

// NOT Schema Compliant
{ "S_0": 42 } // Value must be a string
{ "I_42": "This is a string" } // Value must be an integer
```

AdditionalProperties exemple :

```
{
  "type": "object",
  "properties": {
    "test": {
      "type": "string"
    }
  },
  "additionalProperties": false
}

// Schema Compliant
{
  "test": "value"
}
OR
{}

// NOT Schema Compliant
{
  "notAllowed": false
}
```

UnevaluatedProperties exemple :

```
{
  "type": "object",
  "properties": {
    "standard_field": { "type": "string" }
  },
  "patternProperties": {
    "^@": { "type": "integer" } // Allows properties starting with '@'
  },
  "unevaluatedProperties": false // No other properties allowed
}
```

```
// Schema Compliant
{
  "standard_field": "some value",
  "@id": 123,
  "@timestamp": 1678886400
}
// This passes because "standard_field" is evaluated by properties,
// "@id" and "@timestamp" are evaluated by patternProperties,
// and no other properties remain unevaluated.

// NOT Schema Compliant
{
  "standard_field": "some value",
  "another_field": "unallowed"
}
// This fails because "another_field" is unevaluated and doesn't match
// the @ pattern, leading to a violation of unevaluatedProperties: false
```

AnyOf exemple :

```
{
  "anyOf": [
    { "type": "string", "maxLength": 5 },
    { "type": "number", "minimum": 0 }
  ]
}

// Schema Compliant
"short"
12

// NOT Schema Compliant
"too long"
-5
```

OneOf exemple :

```
{
  "oneOf": [
    { "type": "number", "multipleOf": 5 },
    { "type": "number", "multipleOf": 3 }
  ]
}
```

```
// Schema Compliant
10
9

// NOT Schema compliant
2 // Not a multiple of either 5 or 3
15 // Multiple of both 5 and 3 is rejected.
```

Définitions de types avec espace de noms

Les définitions de types avec espace de noms sont des types créés à partir de types primitifs. Ces types doivent suivre le format *namespace.typename*. Managed integrations fournit des types prédéfinis sous les espaces de noms `aws` et `matter`. Vous pouvez utiliser n'importe quel espace de noms pour les types personnalisés, à l'exception des espaces réservés `aws` et des espaces de noms `matter`.

Pour trouver les définitions de type avec espace de noms disponibles, utilisez l'[ListSchemaVersions](#) API avec le Type filtre défini sur `definition`

Types `matter`

Recherchez les types de données dans l'espace de noms `matter` à l'aide de l'[ListSchemaVersions](#) API avec le Namespace filtre défini sur `matter` et le Type filtre défini sur `definition`

Types `aws`

Recherchez les types de données dans l'espace de noms `aws` à l'aide de l'[ListSchemaVersions](#) API avec le Namespace filtre défini sur `aws` et le Type filtre défini sur `definition`

Définition du type de bitmap

Les bitmaps ont deux propriétés obligatoires :

- `typedoit` être un objet
- `properties` doit être un objet contenant chaque définition de bit. Chaque bit est un objet avec des propriétés `extrinsicId` et `value`. La valeur de chaque bit doit être un entier avec une valeur minimale de 0 et une valeur maximale d'au moins 1.

Exemple de définition d'un bitmap :

```

{
  "title" : "Sample Bitmap Type",
  "description" : "Type definition for SampleBitmap.",
  "$ref" : "/schema-versions/definition/aws.bitmap@1.0 ",
  "type" : "object",
  "additionalProperties" : false,
  "properties" : {
    "Bit1" : {
      "extrinsicId" : "0x0000",
      "value" : {
        "type" : "integer",
        "maximum" : 1,
        "minimum" : 0
      }
    },
    "Bit2" : {
      "extrinsicId" : "0x0001",
      "value" : {
        "type" : "integer",
        "maximum" : 1,
        "minimum" : 0
      }
    }
  }
}

// Schema Compliant
{
  "Bit1": 1,
  "Bit1": 0
}

// NOT Schema Compliant
{
  "Bit1": -1,
  "Bit1": 0
}

```

Définition du type Enum

Les énumérations nécessitent trois propriétés :

- typedoit être un objet

- `enum` doit être un tableau de chaînes uniques, avec au moins un élément
- `extrinsicIdMap` est un objet dont les propriétés sont les valeurs d'énumération. La valeur de chacune des propriétés doit être l'identifiant extrinsèque correspondant à la valeur d'énumération.

Exemple de définition de l'énumération :

```
{
  "title" : "SampleEnum Type",
  "description" : "Type definition for SampleEnum.",
  "$ref" : "/schema-versions/definition/aws.enum@1.0",
  "type" : "string",
  "enum" : [
    "EnumValue0",
    "EnumValue1",
    "EnumValue2"
  ],
  "extrinsicIdMap" : {
    "EnumValue0" : "0",
    "EnumValue1" : "1",
    "EnumValue2" : "2"
  }
}

// Schema Compliant
"EnumValue0"
"EnumValue1"
"EnumValue2"

// NOT Schema Compliant
"NotAnEnumValue"
```

Gérez les commandes et les événements des appareils IoT

Les commandes de l'appareil permettent de gérer à distance un appareil physique en garantissant un contrôle total sur celui-ci, en plus d'effectuer des mises à jour critiques de sécurité, logicielles et matérielles. Avec un vaste parc d'appareils, le fait de savoir quand un appareil exécute une commande permet de superviser l'ensemble de la mise en œuvre de votre appareil. Une commande de l'appareil ou une mise à jour automatique déclenchera un changement d'état de l'appareil, qui à son tour créera un nouvel événement sur l'appareil. Cet événement lié à l'appareil déclenchera l'envoi automatique d'une notification à une destination gérée par le client.

Rubriques

- [Commandes de l'appareil](#)
- [Événements liés à l'appareil](#)

Commandes de l'appareil

Une demande de commande est la commande envoyée à l'appareil. Une demande de commande inclut une charge utile qui spécifie l'action à effectuer, telle que l'allumage de l'ampoule. Pour envoyer une commande d'appareil, l'`SendManagedThingCommandAPI` est appelée au nom de l'utilisateur final par des intégrations gérées et la demande de commande est envoyée à l'appareil.

La réponse à `SendManagedThingCommand` est un `traceId` et vous pouvez l'utiliser `traceId` pour suivre la livraison des commandes et tous les flux de travail de réponse aux commandes associés dans la mesure du possible.

Pour plus d'informations sur le fonctionnement de l'`SendManagedThingCommandAPI`, consultez [SendManagedThingCommand](#).

Action **UpdateState**

Pour mettre à jour l'état d'un appareil, par exemple l'heure à laquelle une lumière s'allume, utilisez l'`UpdateState` action lorsque vous appelez l'`SendManagedThingCommandAPI`. Indiquez la propriété du modèle de données et la nouvelle valeur dans laquelle vous effectuez la mise à jour `parameters`. L'exemple ci-dessous illustre une demande d'`SendManagedThingCommandAPI` mettant à jour le nom `OnTime` d'une ampoule en 5.

```
{
  "Endpoints": [
```



```

{
  "endpointId": "1",
  "capabilities": [
    {
      "id": "matter.OnOff",
      "name": "On/Off",
      "version": "1",
      "actions": [
        {
          "name": "UpdateState",
          "parameters": {
            "OnTime": 5
          }
        }
      ]
    }
  ]
}

```

Action **ReadState**

Pour obtenir l'état le plus récent d'un appareil, y compris les valeurs actuelles de toutes les propriétés du modèle de données, utilisez l'`ReadState` action lorsque vous appelez l'`SendManagedThingCommandAPI`. Dans `propertiesToRead`, vous pouvez utiliser les options suivantes :

- Fournissez une propriété de modèle de données spécifique pour obtenir la dernière valeur, par exemple `OnOff` pour déterminer si un voyant est allumé ou éteint.
- Utilisez l'opérateur générique (*) pour lire toutes les propriétés d'état du périphérique associées à une fonctionnalité.

Les exemples ci-dessous illustrent les deux scénarios d'une demande d'`SendManagedThingCommandAPI` utilisant l'`ReadState` action :

```

{
  "Endpoints": [
    {
      "endpointId": "1",
      "capabilities": [

```

```
{
  "id": "aws.OnOff",
  "name": "On/Off",
  "version": "1",
  "actions": [
    {
      "name": "ReadState",
      "parameters": {
        "propertiesToRead": [ "OnOff" ]
      }
    }
  ]
}
```

```
{
  "Endpoints": [
    {
      "endpointId": "1",
      "capabilities": [
        {
          "id": "aws.OnOff",
          "name": "On/Off",
          "version": "1",
          "actions": [
            {
              "name": "ReadState",
              "parameters": {
                "propertiesToRead": [ "*" ]
              }
            }
          ]
        }
      ]
    }
  ]
}
```

Événements liés à l'appareil

Un événement de dispositif inclut l'état actuel de l'appareil. Cela peut signifier que l'appareil a changé d'état ou qu'il signale son état même s'il n'a pas changé. Il inclut les rapports de propriétés et les événements définis dans le modèle de données. Un événement peut être dû au fait que le cycle du lave-linge est terminé ou que le thermostat a atteint la température cible définie par l'utilisateur final.

Notifications d'événements liés à

Un utilisateur final peut s'abonner à des destinations spécifiques gérées par le client qu'il crée pour recevoir des mises à jour sur des événements spécifiques liés à un appareil. Pour créer une destination gérée par le client, appelez l'`CreateDestinationAPI`. Lorsqu'un événement lié à un appareil est signalé aux intégrations gérées par l'appareil, la destination gérée par le client est avertie s'il en existe une.

Marquage de vos ressources d'intégrations gérées

Pour vous aider à gérer et à organiser vos ressources, vous pouvez éventuellement attribuer vos propres métadonnées à chacune de ces ressources sous forme de balises. Cette section décrit les balises et vous montre comment les créer.

Principes de base des étiquettes

Vous pouvez utiliser des balises pour classer les ressources de vos intégrations gérées de différentes manières (par exemple, par objectif, propriétaire ou environnement). Cette approche est utile lorsque vous avez de nombreuses ressources du même type : elle vous permet d'identifier rapidement une ressource en fonction des balises que vous lui avez attribuées. 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 types d'objet vous permettant de suivre les appareils par type. Nous vous recommandons de créer un ensemble de clés de balise répondant à vos besoins pour chaque type de ressource. L'utilisation d'un ensemble de clés de balise cohérent facilite la gestion de vos ressources.

Vous pouvez rechercher et filtrer les ressources en fonction des balises que vous ajoutez ou appliquez. Vous pouvez également utiliser des balises pour contrôler l'accès à vos ressources, comme décrit dans [Utilisation des balises avec des politiques IAM](#).

Pour faciliter l'utilisation, l'éditeur de balises de la console AWS de gestion fournit un moyen centralisé et unifié de créer et de gérer vos balises. Pour plus d'informations, voir [Utilisation de l'éditeur de balises](#) dans [Utilisation de la console AWS de gestion](#).

Vous pouvez également travailler avec des balises à l'aide de l'API AWS CLI et de l'API d'intégrations gérées. Vous pouvez associer des balises à des objets gérés, à des profils d'approvisionnement, à des bloqueurs d'informations d'identification et à des tâches over-the-air (OTA) lorsque vous les créez en utilisant le Tags champ des commandes suivantes :

- [CreateManagedThing](#)
- [CreateProvisioningProfile](#)
- [CreateCredentialLocker](#)
- [CreateOtaTask](#)
- [CreateAccountAssociation](#)

Vous pouvez ajouter, modifier ou supprimer des balises pour les ressources existantes qui prennent en charge le balisage à l'aide des commandes suivantes :

- [TagResource](#)
- [ListTagsForResource](#)
- [UntagResource](#)

Vous pouvez modifier les clés et valeurs de balise, et vous pouvez retirer des balises d'une ressource à tout moment. Vous pouvez définir la valeur d'une balise sur une chaîne vide, mais vous ne pouvez pas définir la valeur d'une balise sur null. Si vous ajoutez une balise ayant la même clé qu'une balise existante sur cette ressource, la nouvelle valeur remplace l'ancienne valeur. Si vous supprimez une ressource, toutes les balises associées à celle-ci sont également supprimées.

Limites et restrictions liées aux balises

Les restrictions de base suivantes s'appliquent aux balises :

- Nombre maximal de balises par ressource : 50
- Longueur de clé maximale : 127 caractères Unicode en UTF-8
- Longueur de valeur maximale : 255 caractères Unicode en UTF-8
- Les clés et les valeurs des balises distinguent les majuscules et minuscules.
- N'utilisez pas le `aws :` préfixe dans vos noms et valeurs de balise. Il est réservé à AWS l'usage. Vous ne pouvez pas modifier ou supprimer des noms ou valeurs de balise ayant ce préfixe. Les balises avec ce préfixe ne sont pas comptabilisées comme vos balises pour la limite de ressources.
- Si votre schéma de balisage est utilisé pour plusieurs services et ressources, n'oubliez pas que d'autres services peuvent avoir des restrictions concernant les caractères autorisés. Les caractères autorisés incluent les lettres, les espaces et les chiffres représentables en UTF-8, ainsi que les caractères spéciaux suivants : `+ - = . _ : / @`.

Utilisation des balises avec des politiques IAM

Vous pouvez appliquer des autorisations au niveau des ressources basées sur des balises dans les politiques IAM que vous utilisez pour les actions d'API d'intégrations gérées. Vous bénéficiez ainsi d'un meilleur contrôle sur les ressources qu'un utilisateur peut créer, modifier ou utiliser. Vous pouvez utiliser l'élément `Condition` (également appelé bloc `Condition`) avec les clés et valeurs

de contexte de condition suivantes dans une politique IAM pour contrôler l'accès des utilisateurs (autorisations) en fonction des balises d'une ressource :

- Utilisez `aws:ResourceTag/tag-key: tag-value` pour accorder ou refuser aux utilisateurs des actions sur des ressources ayant des balises spécifiques.
- Utilisez `aws:RequestTag/tag-key: tag-value` pour exiger qu'une balise spécifique soit utilisée (ou ne soit pas utilisée) lorsque vous effectuez une demande d'API pour créer ou modifier une ressource qui autorise les balises.
- Utilisez `aws:TagKeys: [tag-key, ...]` pour exiger qu'un ensemble de clés de balise spécifique soit utilisé (ou ne soit pas utilisé) lorsque vous effectuez une demande d'API pour créer ou modifier une ressource qui autorise les balises.

Note

Les clés et valeurs de contexte de condition d'une politique IAM s'appliquent uniquement aux actions d'intégrations gérées pour lesquelles un identifiant pour une ressource pouvant être étiquetée est un paramètre obligatoire. Par exemple, l'utilisation de `n:GetCustomEndpointtest` pas autorisée ou refusée sur la base des clés et des valeurs du contexte des conditions, car aucune ressource balisable (objets gérés, profils d'approvisionnement, casiers d'informations d'identification, over-the-air tâches) n'est référencée dans cette demande. Pour plus d'informations sur les ressources d'intégrations gérées qui peuvent être étiquetées et les clés de condition qu'elles prennent en charge, consultez la fonctionnalité [Actions, ressources et clés de condition pour les intégrations AWS IoT gérées](#) de AWS IoT Device Management

Pour de plus amples informations, veuillez consulter [Contrôle de l'accès à l'aide d'étiquettes](#) dans le Gestion des identités et des accès AWS Guide de l'utilisateur. La section [Référence de stratégie JSON IAM](#) de ce guide fournit la syntaxe détaillée, des descriptions, ainsi que des exemples des éléments, des variables et de la logique d'évaluation des stratégies JSON dans IAM.

L'exemple de politique suivant applique deux restrictions basées sur des balises à l'`CreateManagedThingaction`. Un utilisateur IAM restreint par cette stratégie :

- Impossible de créer un objet géré avec le tag « env=prod » (dans l'exemple, voir la ligne).
`"aws:RequestTag/env" : "prod"`
- Impossible de modifier ou d'accéder à un objet géré qui possède une balise existante « env=prod » (dans l'exemple, voir la ligne). `"aws:ResourceTag/env" : "prod"`

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": "iotmanagedintegrations:CreateManagedThing",
      "Resource": "arn:aws:iotmanagedintegrations:us-east-1:123456789012:managed-thing/*",
      "Condition": {
        "StringEquals": {
          "aws:RequestTag/env": "prod"
        }
      }
    },
    {
      "Effect": "Deny",
      "Action": [
        "iotmanagedintegrations:CreateManagedThing",
        "iotmanagedintegrations>DeleteManagedThing",
        "iotmanagedintegrations:GetManagedThing",
        "iotmanagedintegrations:UpdateManagedThing"
      ],
      "Resource": "arn:aws:iotmanagedintegrations:us-east-1:123456789012:managed-thing/*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/env": "prod"
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": [
        "iotmanagedintegrations:CreateManagedThing",
        "iotmanagedintegrations>DeleteManagedThing",
        "iotmanagedintegrations:GetManagedThing",
        "iotmanagedintegrations:UpdateManagedThing"
      ],
      "Resource": "*"
    }
  ]
}
```

```
}
```

Vous pouvez également spécifier plusieurs valeurs de balise pour une clé de balise donnée en les plaçant dans une liste, comme suit :

```
"StringEquals" : {  
  "aws:ResourceTag/env" : ["dev", "test"]  
}
```

Note

Si vous autorisez ou refusez à des utilisateurs l'accès à des ressources en fonction de balises, vous devez envisager de refuser de manière explicite la possibilité pour les utilisateurs d'ajouter ces balises ou de les supprimer des 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.

Notifications d'intégrations gérées

Les notifications d'intégrations gérées fournissent des mises à jour et des informations clés sur les appareils. Les notifications incluent les événements liés aux connecteurs, les commandes des appareils, les événements liés au cycle de vie, les mises à jour OTA (Over-the-Air) et les rapports d'erreurs. Ces informations fournissent des informations exploitables pour créer des flux de travail automatisés, prendre des mesures immédiates ou stocker des données d'événements à des fins de dépannage.

Actuellement, seuls les flux de données Amazon Kinesis sont pris en charge en tant que destination pour les notifications d'intégrations gérées. Vous devez d'abord configurer un flux de données Amazon Kinesis et autoriser les intégrations gérées à accéder au flux de données avant de configurer les notifications.

Configurer Amazon Kinesis pour les notifications

Étapes de configuration d'Amazon Kinesis

- [Étape 1 : créer un flux de données Amazon Kinesis](#)
- [Étape 2 : créer une politique d'autorisation](#)
- [Étape 3 : Accédez au tableau de bord IAM et sélectionnez Rôles](#)
- [Étape 4 : Utiliser une politique de confiance personnalisée](#)
- [Étape 5 : appliquez votre politique d'autorisation](#)
- [Étape 6 : Entrez un nom de rôle](#)

Pour configurer Amazon Kinesis pour les notifications d'intégrations gérées, procédez comme suit :

Étape 1 : créer un flux de données Amazon Kinesis

Un flux de données Amazon Kinesis peut ingérer une grande quantité de données en temps réel, les stocker de manière durable et les mettre à disposition des applications.

Pour créer un flux de données Amazon Kinesis

- Pour créer un flux de données Kinesis, suivez les étapes décrites dans la section [Création et gestion des flux de données Kinesis](#).

Étape 2 : créer une politique d'autorisation

Créez une politique d'autorisation qui autorise les intégrations gérées à accéder à votre flux de données Kinesis.

Pour créer une politique d'autorisations

- Pour créer une politique d'autorisation, copiez la politique ci-dessous et suivez les étapes décrites dans [Création de politiques à l'aide de l'éditeur JSON](#)

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": "kinesis:PutRecord",
      "Resource": "*",
      "Effect": "Allow"
    }
  ]
}
```

Étape 3 : Accédez au tableau de bord IAM et sélectionnez Rôles

Ouvrez le tableau de bord IAM et cliquez sur Rôles.

Pour accéder au tableau de bord IAM

- Ouvrez le tableau de bord IAM et cliquez sur Rôles.

Pour plus d'informations, consultez la section [Création de rôles IAM](#) dans le guide de l'Gestion des identités et des accès AWSutilisateur.

Étape 4 : Utiliser une politique de confiance personnalisée

Vous pouvez utiliser une politique de confiance personnalisée pour accorder aux intégrations gérées l'accès au flux de données Kinesis.

Pour utiliser une politique de confiance personnalisée

- Créez un nouveau rôle et choisissez Politique de confiance personnalisée. Cliquez sur Suivant.

La politique suivante permet aux intégrations gérées d'assumer ce rôle, et la Condition déclaration permet d'éviter toute confusion liée aux adjoints.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "iotmanagedintegrations.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "123456789012"
        },
        "ArnLike": {
          "aws:SourceArn": "arn:aws:iotmanagedintegrations:ca-central-1:123456789012:*"
        }
      }
    }
  ]
}
```

Étape 5 : appliquez votre politique d'autorisation

Ajoutez au rôle la politique d'autorisation que vous avez créée à l'étape 2.

Pour ajouter une politique d'autorisation

- Sur la page Ajouter des autorisations, recherchez et ajoutez la politique d'autorisation que vous avez créée à l'étape 2. Cliquez sur Suivant.

Étape 6 : Entrez un nom de rôle

- Entrez un nom de rôle et cliquez sur Créer un rôle.

Configurer les notifications d'intégrations gérées

Étapes de configuration des notifications

- [Étape 1 : autoriser l'utilisateur à appeler l' CreateDestination API](#)
- [Étape 2 : appeler l' CreateDestination API](#)
- [Étape 3 : appeler l' CreateNotificationConfiguration API](#)

Pour configurer les notifications d'intégrations gérées, procédez comme suit :

Étape 1 : autoriser l'utilisateur à appeler l' CreateDestination API

- Autoriser l'utilisateur à appeler l'**CreateDestinationAPI**

La politique suivante définit les conditions requises pour que l'utilisateur appelle l'[CreateDestinationAPI](#).

Consultez la section [Accorder à un utilisateur l'autorisation de transmettre un rôle à un AWS service](#) dans le Guide de Gestion des identités et des accès AWS l'utilisateur pour obtenir des autorisations de transfert pour les intégrations gérées.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iam:PassRole",
      "Resource": "arn:aws:iam::123456789012:role/ROLE_CREATED_IN_PREVIOUS_STEP",
      "Condition": {
        "StringEquals": {
          "iam:PassedToService": "iotmanagedintegrations.amazonaws.com"
        }
      }
    }
  ]
}
```

```

        }
    },
    {
        "Effect": "Allow",
        "Action": "iotmanagedintegrations:CreateDestination",
        "Resource": "*"
    }
]
}

```

Étape 2 : appeler l' CreateDestination API

- Appelez l'**CreateDestinationAPI**

Après avoir créé votre flux de données Amazon Kinesis et votre rôle d'accès au flux, appelez l'[CreateDestination](#) API pour créer votre destination de notification vers laquelle les notifications seront acheminées. Pour le `DeliveryDestinationArn` paramètre, utilisez celui arn de votre nouveau flux de données Amazon Kinesis.

```

{
  "DeliveryDestinationArn": "Your Kinesis arn"
  "DeliveryDestinationType": "KINESIS"
  "Name": "DestinationName"
  "ClientToken": "string"
  "RoleArn": "arn:aws:iam::accountID:role/ROLE_CREATED_IN_PREVIOUS_STEP"
}

```

Note

`ClientToken` est un jeton d'identité. Si vous réessayez une demande qui s'est initialement terminée avec succès en utilisant le même jeton client et les mêmes paramètres, la nouvelle tentative aboutira sans aucune autre action.

Étape 3 : appeler l' CreateNotificationConfiguration API

- Appelez l'**CreateNotificationConfigurationAPI**

Enfin, utilisez l'[CreateNotificationConfiguration](#) API pour créer la configuration des notifications qui achemine les types d'événements choisis vers votre destination représentée par le flux de données Kinesis. Dans le `DestinationName` paramètre, utilisez le même nom de destination que lorsque vous avez initialement appelé l'`CreateDestination` API.

```
{
  "EventType": "DEVICE_EVENT"
  "DestinationName" // This name has to be identical to the name in
  createDestination API
  "ClientToken": "string"
}
```

Types d'événements surveillés avec des intégrations gérées

Les types d'événements surveillés à l'aide des notifications d'intégrations gérées sont les suivants :

- `DEVICE_COMMAND`
- État de la commande [SendManagedThingCommand](#) API. Les valeurs valides sont `succeeded` ou `failed`.

```
{
  "version": "0",
  "messageId": "6a7e8feb-b491-4cf7-a9f1-bf3703467718",
  "messageType": "DEVICE_COMMAND",
  "source": "aws.iotmanagedintegrations",
  "customerAccountId": "123456789012",
  "timestamp": "2017-12-22T18:43:48Z",
  "region": "ca-central-1",
  "resources": [
    "arn:aws:iotmanagedintegrations:ca-central-1:123456789012:managed-thing/6a7e8feb-b491-4cf7-a9f1-bf3703467718"
  ],
  "payload": {
    "traceId": "1234567890abcdef0",
    "receivedAt": "2017-12-22T18:43:48Z",
    "executedAt": "2017-12-22T18:43:48Z",
    "result": "failed"
  }
}
```

- **DEVICE_COMMAND_REQUEST**

- La demande de commande de Web Real-Time Communication (WebRTC).

La norme WebRTC permet la communication entre deux pairs. Ces pairs peuvent transmettre des données vidéo, audio et arbitraires en temps réel. Les intégrations gérées prennent en charge le WebRTC pour permettre ce type de streaming entre l'application mobile d'un client et l'appareil de l'utilisateur final. [Pour plus d'informations sur la norme WebRTC, voir WebRTC.](#)

```
{
    "version": "0",
    "messageId": "6a7e8feb-b491-4cf7-a9f1-bf3703467718",
    "messageType": "DEVICE_COMMAND_REQUEST",
    "source": "aws.iotmanagedintegrations",
    "customerAccountId": "123456789012",
    "timestamp": "2017-12-22T18:43:48Z",
    "region": "ca-central-1",
    "resources": [
        "arn:aws:iotmanagedintegrations:ca-central-1:123456789012:managed-thing/6a7e8feb-b491-4cf7-a9f1-bf3703467718"
    ],
    "payload": {
        "endpoints": [{
            "endpointId": "1",
            "capabilities": [{
                "id": "aws.DoorLock",
                "name": "Door Lock",
                "version": "1.0"
            }]
        }]
    }
}
```

- **DEVICE_DISCOVERY_STATUS**

- État de découverte de l'appareil.

```
{
    "version": "0",
    "messageId": "6a7e8feb-b491-4cf7-a9f1-bf3703467718",
    "messageType": "DEVICE_DISCOVERY_STATUS",
    "source": "aws.iotmanagedintegrations",
    "customerAccountId": "123456789012",
    "timestamp": "2017-12-22T18:43:48Z",
```

```

    "region": "ca-central-1",
    "resources": [
      "arn:aws:iotmanagedintegrations:ca-central-1:123456789012:managed-
thing/6a7e8feb-b491-4cf7-a9f1-bf3703467718"
    ],
    "payload": {
      "deviceCount": 1,
      "deviceDiscoveryId": "123",
      "status": "SUCCEEDED"
    }
  }
}

```

- **DEVICE_EVENT**

- Notification d'un événement survenant sur un appareil.

```

{
  "version": "1.0",
  "messageId": "2ed545027bd347a2b855d28f94559940",
  "messageType": "DEVICE_EVENT",
  "source": "aws.iotmanagedintegrations",
  "customerAccountId": "123456789012",
  "timestamp": "1731630247280",
  "resources": [
    "/quit/1b15b39992f9460ba82c6c04595d1f4f"
  ],
  "payload": {
    "endpoints": [{
      "endpointId": "1",
      "capabilities": [{
        "id": "aws.DoorLock",
        "name": "Door Lock",
        "version": "1.0",
        "properties": [{
          "name": "ActuatorEnabled",
          "value": "true"
        }]
      }]
    }]
  }
}

```

- **DEVICE_LIFE_CYCLE**

- État du cycle de vie de l'appareil.


```
{
  "version": "1.0.0",
  "messageId": "8d1e311a473f44f89d821531a0907b05",
  "messageType": "DEVICE_LIFE_CYCLE",
  "source": "aws.iotmanagedintegrations",
  "customerAccountId": "123456789012",
  "timestamp": "2024-11-14T19:55:57.568284645Z",
  "region": "ca-central-1",
  "resources": [
    "arn:aws:iotmanagedintegrations:ca-central-1:123456789012:managed-thing/d5c280b423a042f3933eed09cf408657"
  ],
  "payload": {
    "deviceDetails": {
      "id": "d5c280b423a042f3933eed09cf408657",
      "arn": "arn:aws:iotmanagedintegrations:ca-central-1:123456789012:managed-thing/d5c280b423a042f3933eed09cf408657",
      "createdAt": "2024-11-14T19:55:57.515841147Z",
      "updatedAt": "2024-11-14T19:55:57.515841559Z"
    },
    "status": "UNCLAIMED"
  }
}
```

- **DEVICE_OTA**
 - Une notification OTA de l'appareil.
- **DEVICE_STATE**
 - Une notification lorsque l'état d'un appareil a été mis à jour.

```
{
  "messageType": "DEVICE_STATE",
  "source": "aws.iotmanagedintegrations",
  "customerAccountId": "123456789012",
  "timestamp": "1731623291671",
  "resources": [
    "arn:aws:iotmanagedintegrations:ca-central-1:123456789012:managed-thing/61889008880012345678"
  ],
  "payload": {
    "addedStates": {
      "endpoints": [{
```

```
    "endpointId": "nonEndpointId",
    "capabilities": [{
      "id": "aws.OnOff",
      "name": "On/Off",
      "version": "1.0",
      "properties": [{
        "name": "OnOff",
        "value": {
          "propertyValue": "\"onoff\"",
          "lastChangedAt": "2024-06-11T01:38:09.000414Z"
        }
      }]
    }]
  }
}
```

Cloud-to-Cloud connecteurs (C2C)

Un cloud-to-cloud connecteur vous permet de créer et de faciliter une communication bidirectionnelle entre des appareils tiers et AWS.

Rubriques

- [Qu'est-ce qu'un connecteur cloud-to-cloud \(C2C\) ?](#)
- [Qu'est-ce que le catalogue de connecteurs C2C ?](#)
- [AWS Lambda fonctionne comme des connecteurs C2C](#)
- [Flux de travail du connecteur d'intégrations gérées](#)
- [Directives d'utilisation d'un connecteur C2C \(cloud-to-cloud\)](#)
- [Créez un connecteur C2C \(Cloud-to-Cloud\)](#)
- [Utiliser un connecteur C2C \(Cloud-to-Cloud\)](#)

Qu'est-ce qu'un connecteur cloud-to-cloud (C2C) ?

Un cloud-to-cloud connecteur est un progiciel prédéfini qui relie de manière sécurisée le point AWS Cloud de terminaison d'un fournisseur de cloud tiers. À l'aide du connecteur C2C, les fournisseurs de solutions peuvent tirer parti des intégrations gérées pour AWS IoT Device Management afin de contrôler les appareils connectés à des clouds tiers.

Les intégrations gérées incluent un catalogue de connecteurs dans lequel AWS les clients peuvent consulter et sélectionner les connecteurs auxquels ils souhaitent s'intégrer. Pour de plus amples informations, consultez [Qu'est-ce que le catalogue de connecteurs C2C ?](#).

Les intégrations gérées nécessitent que chaque connecteur soit implémenté en tant que AWS Lambda fonction.

Qu'est-ce que le catalogue de connecteurs C2C ?

Le catalogue de connecteurs d'intégrations gérées pour AWS IoT Device Management est un ensemble de connecteurs C2C qui facilitent la communication bidirectionnelle entre les intégrations gérées pour AWS IoT Device Management et un fournisseur de cloud tiers. Vous pouvez voir les connecteurs dans le AWS Management Console ou le AWS CLI.

Pour utiliser la console afin de consulter le catalogue des connecteurs d'intégrations gérées

1. Ouvrez la console des [intégrations gérées](#)
2. Dans le volet de navigation de gauche, choisissez Managed integrations
3. Dans le volet de navigation de gauche de la console des intégrations gérées, choisissez Catalog.

AWS Lambda fonctionne comme des connecteurs C2C

Chaque fonction Lambda du connecteur C2C traduit et transporte les commandes et les événements entre les intégrations gérées et les actions correspondantes sur des plateformes tierces. Pour plus d'informations sur Lambda, consultez [Qu'est-ce que](#). AWS Lambda

Supposons, par exemple, qu'un utilisateur final possède une ampoule intelligente fabriquée par un OEM tiers. Avec un connecteur C2C, un utilisateur final peut émettre une commande pour allumer ou éteindre cette lumière via une plateforme d'intégration gérée. Cette commande sera ensuite transmise à la fonction Lambda hébergée dans le connecteur, qui traduira la demande en un appel d'API à la plateforme tierce pour allumer ou éteindre l'appareil.

La fonction Lambda est requise lorsque vous appelez l'CreateCloudConnectorAPI. Le code déployé dans la fonction Lambda doit implémenter toutes les interfaces et fonctionnalités mentionnées dans. [Créez un connecteur C2C \(Cloud-to-Cloud\)](#)

Flux de travail du connecteur d'intégrations gérées

Les développeurs doivent enregistrer des connecteurs C2C avec des intégrations gérées pour. AWS IoT Device Management Ce processus d'enregistrement crée une ressource de connecteur logique à laquelle les clients peuvent accéder pour utiliser le connecteur.

Note

Un connecteur C2C est un ensemble de métadonnées créé dans le cadre d'intégrations gérées pour AWS IoT Device Management afin de décrire le connecteur.

Le schéma suivant décrit le rôle d'un connecteur C2C lors de l'envoi d'une commande depuis l'application mobile vers un appareil connecté au cloud. Le connecteur C2C agit comme une couche de traduction entre les intégrations gérées pour AWS IoT Device Management et une plateforme cloud tierce.

Directives d'utilisation d'un connecteur C2C (cloud-to-cloud)

Tout connecteur C2C que vous créez est votre contenu, et tout connecteur C2C créé par un autre client auquel vous accédez est un contenu tiers. AWS ne crée ni ne gère aucun connecteur C2C dans le cadre d'intégrations gérées.

Vous pouvez partager vos connecteurs C2C avec d'autres clients d'intégrations gérées. Si vous le faites, vous autorisez AWS en tant que fournisseur de services à répertorier ces connecteurs C2C et les informations de contact associées sur la AWS console et vous comprenez que d'autres AWS clients peuvent vous contacter. Vous êtes seul responsable de l'accès des clients à vos connecteurs C2C et de toutes les conditions régissant l'accès d'un autre AWS client à vos connecteurs C2C.

Créez un connecteur C2C (Cloud-to-Cloud)

Les sections suivantes décrivent les étapes de création d'un connecteur C2C (Cloud-to-Cloud) pour les intégrations gérées pour AWS IoT Device Management.

Rubriques

- [Conditions préalables](#)
- [Exigences relatives au connecteur C2C](#)
- [OAuth 2.0 exigences relatives à la liaison de comptes](#)
- [Mettre en œuvre les opérations d'interface du connecteur C2C](#)
- [Appelez votre connecteur C2C](#)
- [Ajoutez des autorisations à votre rôle IAM](#)
- [Testez manuellement votre connecteur C2C](#)

Conditions préalables

Avant de créer un connecteur C2C (Cloud-to-Cloud), vous devez disposer des éléments suivants :

- Et Compte AWS pour héberger votre connecteur C2C et pour l'enregistrer via des intégrations gérées. Pour plus d'informations, consultez la section [Créer un Compte AWS](#).
- Lorsque vous créez votre connecteur, vous avez besoin de certaines autorisations IAM. Pour utiliser le plugin

- Assurez-vous que les fournisseurs de cloud tiers auxquels le connecteur est destiné prennent en charge l'autorisation OAuth 2.0. Pour de plus amples informations, veuillez consulter [OAuth 2.0 exigences relatives à la liaison de comptes](#).

En outre, pour tester le connecteur, le développeur du connecteur doit disposer des éléments suivants :

- Un identifiant client provenant du cloud tiers à associer à votre connecteur C2C
- Un secret client provenant du cloud tiers à associer à votre connecteur C2C
- Une URL d'autorisation OAuth 2.0
- Une URL de jeton OAuth 2.0
- Toutes les clés d'API requises par votre API tierce
- Toutes les clés d'API requises par l'enregistrement ou la liste d'autorisation de votre API tierce pour l'URL de OAuth rappel hébergée par AWS. Certains tiers autorisent explicitement la mise en liste d'une URL de OAuth redirection, tandis que d'autres ont un flux de travail dans lequel les utilisateurs peuvent se connecter et enregistrer l' OAuth URL. Consultez le tiers concerné pour comprendre ce qui est requis pour autoriser la liste des intégrations OAuth gérées (point de terminaison de redirection).

Autorisations requises

Lorsque vous créez votre connecteur, vous avez besoin de certaines autorisations IAM. Outre les `iotmanagedintegrations` : autorisations pour les actions, vous avez besoin des autorisations suivantes :

- [CreateAccountAssociation](#), [CreateConnectorDestinationGetAccountAssociation](#), et [StartAccountAssociationRefresh](#), nécessitent `secretsmanager:GetSecretValue`
- [CreateCloudConnector](#) nécessite `lambda:Invoke`.

Pour plus d'informations sur `iotmanagedintegrations` : les autorisations et les actions, voir [Actions définies par les intégrations AWS gérées](#)

Exigences relatives au connecteur C2C

Le [connecteur C2C](#) que vous développez facilite la communication bidirectionnelle entre les intégrations gérées pour AWS IoT Device Management et le cloud d'un fournisseur tiers. Le connecteur doit implémenter des interfaces pour les intégrations gérées afin qu'AWS IoT Device

Management puisse effectuer des actions pour le compte des utilisateurs finaux. Ces interfaces fournissent des fonctionnalités permettant de découvrir les appareils des utilisateurs finaux, de lancer des commandes d'appareils envoyées depuis des intégrations gérées pour AWS IoT Device Management et d'identifier les utilisateurs à l'aide d'un jeton d'accès. Pour prendre en charge le fonctionnement de l'appareil, le connecteur doit gérer la traduction des messages de demande et de réponse entre les intégrations gérées pour AWS IoT Device Management et la plateforme tierce associée.

Les exigences relatives au connecteur C2C sont les suivantes :

- Le serveur d'autorisation tiers doit être conforme aux normes OAuth 2.0 ainsi qu'aux configurations répertoriées dans [OAuth exigences de configuration](#).
- Un connecteur C2C sera nécessaire pour interpréter les identifiants issus des AWS implémentations du modèle de données de matière et doit émettre les réponses et les événements conformes aux AWS implémentations du modèle de données de matière. Pour de plus amples informations, consultez [AWS mise en œuvre du modèle de données Matter](#).
- Un connecteur C2C doit être capable d'appeler les intégrations gérées pour AWS IoT Device Management APIs avec SigV4 authentification. Pour les événements asynchrones envoyés avec l' `SendConnectorEvent` API, les mêmes Compte AWS informations d'identification utilisées pour enregistrer le connecteur doivent être utilisées pour signer la demande associée `SendConnectorEvent` .
- Le connecteur doit implémenter les [AWS.DeactivateUser](#) opérations [AWS.ActivateUser](#) [AWS.DiscoverDevices](#) [AWS.SendCommand](#), et.
- Lorsque votre connecteur C2C reçoit des événements tiers liés aux réponses aux commandes de l'appareil ou à la découverte de l'appareil, il doit les transmettre aux intégrations gérées avec l'`SendConnectorEventAPI`. Pour plus d'informations sur ces événements et sur l'`SendConnectorEventAPI`, consultez [SendConnectorEvent](#).

 Note

L'`SendConnectorEventAPI` fait partie du SDK d'intégrations gérées et est utilisée à la place de la création et de la signature manuelles des demandes.

OAuth 2.0 exigences relatives à la liaison de comptes

Chaque connecteur C2C s'appuie sur un serveur d'autorisation OAuth 2.0 pour authentifier les utilisateurs finaux. Grâce à ce serveur, les utilisateurs finaux relient leurs comptes tiers à la plateforme de l'appareil du client. L'association de comptes est la première étape requise par un utilisateur final pour utiliser les appareils compatibles avec votre connecteur C2C. Pour plus d'informations sur les différents rôles liés à la liaison de comptes et à la OAuth version 2.0, consultez [Rôles de liaison de comptes](#).

Bien que votre connecteur C2C n'ait pas besoin d'implémenter une logique métier spécifique pour prendre en charge le flux d'autorisation, le serveur d'autorisation OAuth2 .0 associé à votre connecteur C2C doit respecter les. [OAuth exigences de configuration](#)

Note

Intégrations gérées AWS IoT Device Management uniquement pour les supports OAuth 2.0 avec un flux de code d'autorisation. Consultez la [RFC 6749](#) pour plus d'informations.

La liaison de comptes est un processus qui permet aux intégrations gérées et au connecteur d'accéder aux appareils d'un utilisateur final à l'aide d'un jeton d'accès. Ce jeton fournit des intégrations gérées pour AWS IoT Device Management avec l'autorisation de l'utilisateur final, de sorte que le connecteur puisse interagir avec les données de l'utilisateur final via des appels d'API. Pour de plus amples informations, veuillez consulter [Flux de travail de liaison de comptes](#).

Nous vous recommandons de n'enregistrer ces jetons sensibles dans aucun journal. Toutefois, s'ils sont stockés dans des journaux, nous vous recommandons d'utiliser CloudWatch les politiques de protection des données des journaux pour masquer les jetons présents dans les journaux. Pour plus d'informations, consultez [Aider à protéger les données sensibles des journaux grâce au masquage](#).

Les intégrations gérées pour AWS IoT Device Management n'obtiennent pas directement de jeton d'accès ; elles le font via le type d'octroi de code d'autorisation. Tout d'abord, les intégrations gérées pour AWS IoT Device Management doivent obtenir un code d'autorisation. Il échange ensuite le code contre un jeton d'accès et un jeton d'actualisation. Le jeton d'actualisation est utilisé pour demander un nouveau jeton d'accès lorsque l'ancien jeton d'accès expire. Si le jeton d'accès et le jeton d'actualisation ont tous deux expiré, vous devez exécuter à nouveau le flux de liaison des comptes. Vous pouvez le faire à l'aide de l'opération `StartAccountAssociationRefresh` API.

⚠ Important

Le jeton d'accès émis doit être défini par utilisateur, mais pas par OAuth client. Le jeton ne doit pas donner accès à tous les appareils de tous les utilisateurs du client.

Le serveur d'autorisation doit effectuer l'une des opérations suivantes :

- Émettez des jetons d'accès contenant un identifiant d'utilisateur final (propriétaire de la ressource) extractible, tel qu'un jeton JWT.
- Renvoie l'identifiant de l'utilisateur final pour chaque jeton d'accès émis.

OAuth exigences de configuration

Le tableau suivant illustre les paramètres requis par votre serveur OAuth d'autorisation pour les intégrations gérées permettant à AWS IoT Device Management d'effectuer la [liaison de comptes](#) :

OAuth Paramètres du serveur

Champ	Obligatoire	Commentaire
<code>clientId</code>	Oui	Un identifiant public pour votre application. Il est utilisé pour initier des flux d'authentification et peut être partagé publiquement.
<code>clientSecret</code>	Oui	Clé secrète utilisée pour authentifier l'application auprès du serveur d'autorisation, notamment lors de l'échange d'un code d'autorisation contre un jeton d'accès. Il doit rester confidentiel et ne pas être partagé publiquement.
<code>authorizationType</code>	Oui	Type d'autorisation pris en charge par cette configuration d'autorisation. Actuellement,

		« OAuth 2.0 » est la seule valeur prise en charge.
authUrl	Oui	URL d'autorisation du fournisseur de cloud tiers.
tokenUrl	Oui	URL du jeton pour le fournisseur de cloud tiers.
tokenEndpointAuthenticationScheme	Oui	Schéma d'authentification « HTTP_BASIC » ou « REQUEST_BODY_CREDENTIALS ». HTTP_BASIC indique que les informations d'identification du client sont incluses dans l'en-tête d'autorisation, tandis que le latter indique qu'elles sont incluses dans le corps de la demande.

Le OAuth serveur que vous utilisez doit être configuré de telle sorte que les valeurs des chaînes de jetons d'accès soient codées en Base64 avec le jeu de caractères UTF-8.

Rôles de liaison de comptes

Pour créer un connecteur C2C, vous aurez besoin d'un serveur d'autorisation OAuth 2.0 et d'un lien de compte. Pour de plus amples informations, veuillez consulter [Flux de travail de liaison de comptes](#).

OAuth La version 2.0 définit les quatre rôles suivants lors de la mise en œuvre de la liaison de comptes :

1. Serveur d'autorisation
2. Propriétaire de la ressource (utilisateur final)
3. Serveur de ressources
4. Client

Les éléments suivants définissent chacun de ces OAuth rôles :

Serveur d'autorisation

Le serveur d'autorisation est le serveur qui identifie et authentifie l'identité d'un utilisateur final dans un cloud tiers. Les jetons d'accès fournis par ce serveur peuvent relier le compte de plateforme client de l'utilisateur AWS final à son compte de plateforme tierce. Ce processus est appelé « liaison de comptes ».

Le serveur d'autorisation prend en charge la liaison de comptes en fournissant les éléments suivants :

- Affiche une page de connexion permettant à l'utilisateur final de se connecter à votre système. C'est ce que l'on appelle généralement un point de terminaison d'autorisation.
- Authentifie l'utilisateur final dans votre système.
- Génère un code d'autorisation identifiant l'utilisateur final.
- Transmet le code d'autorisation aux intégrations gérées pour AWS IoT Device Management.
- Accepte le code d'autorisation des intégrations gérées pour AWS IoT Device Management et renvoie un jeton d'accès que les intégrations gérées pour AWS IoT Device Management peuvent utiliser pour accéder aux données de l'utilisateur final dans votre système. Cela se fait généralement par le biais d'un URI distinct, appelé URI de jeton ou point de terminaison.

Important

Le serveur d'autorisation doit prendre en charge le flux de code d'autorisation OAuth 2.0 à utiliser avec une intégration gérée pour AWS IoT Device Management Connector. Les intégrations gérées pour AWS IoT Device Management prennent également en charge le flux de code d'autorisation avec [Proof Key for Code Exchange \(PKCE\)](#).

Le serveur d'autorisation doit soit :

- Émettre des jetons d'accès contenant un identifiant d'utilisateur final ou de propriétaire de ressource extractible, par exemple des jetons JWT
- Être en mesure de renvoyer l'identifiant de l'utilisateur final pour chaque jeton d'accès émis

Dans le cas contraire, votre connecteur ne sera pas en mesure de prendre en charge le `AWS.ActivateUser` fonctionnement requis. Cela empêchera l'utilisation du connecteur avec les intégrations gérées.

Si le développeur ou le propriétaire du connecteur ne gère pas son propre serveur d'autorisation, le serveur d'autorisation utilisé doit fournir une autorisation pour les ressources gérées par la plate-forme tierce du développeur du connecteur. Cela signifie que tous les jetons reçus par les intégrations gérées depuis le serveur d'autorisation doivent fournir des limites de sécurité significatives sur les appareils (la ressource). Par exemple, un jeton d'utilisateur final n'autorise pas les commandes sur l'appareil d'un autre utilisateur final ; les autorisations fournies par le jeton sont mappées aux ressources de la plateforme. Prenons l'exemple de Lights Incorporated. Lorsqu'un utilisateur final lance le flux de liaison du compte avec son connecteur, il est redirigé vers la page de connexion de Lights Incorporated qui fait face à son serveur d'autorisation. Une fois qu'ils se sont connectés et ont accordé des autorisations au client, ils fournissent un jeton qui permet au connecteur d'accéder aux ressources de leur compte Lights Incorporated.

Propriétaire de la ressource (utilisateur final)

En tant que propriétaire des ressources, vous autorisez les intégrations gérées pour AWS IoT Device Management à accéder aux ressources associées à votre compte en établissant un lien entre les comptes. Prenons l'exemple de l'ampoule intelligente qu'un utilisateur final a intégrée à l'application mobile Lights Incorporated. Le propriétaire de la ressource fait référence au compte de l'utilisateur final qui a acheté et intégré l'appareil. Dans notre exemple, le propriétaire de la ressource est modélisé comme un compte Lights Incorporated OAuth2 2.0. En tant que propriétaire de la ressource, ce compte fournit les autorisations nécessaires pour émettre des commandes et gérer l'appareil.

Serveur de ressources

Il s'agit du serveur qui héberge les ressources protégées qui nécessitent une autorisation d'accès (données de l'appareil). Le AWS client doit accéder à des ressources protégées pour le compte d'un utilisateur final, et il le fait par le biais d'intégrations gérées pour les connecteurs AWS IoT Device Management après la création de comptes. En prenant l'exemple de l'ampoule intelligente d'avant, le serveur de ressources est un service basé sur le cloud appartenant à Lights Incorporated qui gère l'ampoule une fois qu'elle a été intégrée. Par le biais du serveur de ressources, le propriétaire de la ressource peut envoyer des commandes à l'ampoule intelligente, par exemple l'allumer et l'éteindre. La ressource protégée fournit uniquement des autorisations au compte de l'utilisateur final et à d'autres comptes accounts/entities auxquels il peut avoir accordé des autorisations.

Client

Dans ce contexte, le client est votre connecteur C2C. Un client est défini comme une application qui est autorisée à accéder aux ressources d'un serveur de ressources pour le compte de

l'utilisateur final. Le processus de liaison des comptes représente le connecteur, le client, qui demande l'accès aux ressources d'un utilisateur final dans le cloud tiers.

Bien que le connecteur soit le OAuth client, les intégrations gérées pour AWS IoT Device Management exécutent les opérations pour le compte du connecteur. Par exemple, les intégrations gérées pour AWS IoT Device Management demandent au serveur d'autorisation d'obtenir un jeton d'accès. Le connecteur est toujours considéré comme le client car c'est le seul composant qui accède à la ressource protégée (données de l'appareil) sur le serveur de ressources.

Pensez à l'ampoule intelligente qui a été intégrée par un utilisateur final. Une fois la liaison des comptes établie entre la plateforme client et le serveur d'autorisation de Lights Incorporated, le connecteur lui-même communiquera avec le serveur de ressources pour récupérer des informations sur l'ampoule intelligente de l'utilisateur final. Le connecteur peut ensuite recevoir des commandes de l'utilisateur final. Cela inclut l'allumage ou l'extinction de la lumière en leur nom via le serveur de ressources Lights Incorporated. Ainsi, nous désignons le connecteur comme client.

Flux de travail de liaison de comptes

Pour que les intégrations gérées par un client pour la plateforme AWS IoT Device Management interagissent avec les appareils d'un utilisateur final sur votre plateforme tierce via votre connecteur C2C, celui-ci obtient le jeton d'accès via le flux de travail suivant :

1. Lorsqu'un utilisateur lance l'intégration d'appareils tiers via l'application client, les intégrations gérées pour AWS IoT Device Management renvoient l'URI d'autorisation ainsi que le `AssociationId`.
2. Le front-end de l'application stocke `AssociationId` et redirige l'utilisateur final vers la page de connexion de la plateforme tierce.
 - L'utilisateur final se connecte. L'utilisateur final accorde au client l'accès aux données de son appareil.
3. La plateforme tierce crée un code d'autorisation. L'utilisateur final est redirigé vers les intégrations gérées pour l'URI de rappel de la plateforme AWS IoT Device Management, y compris le code joint à la demande de redirection.
4. Managed Integrations échange ce code avec l'URI du jeton de plateforme tierce.
5. L'URI du jeton valide le code d'autorisation et renvoie un jeton d'accès OAuth2 .0 et un jeton d'actualisation, associés à l'utilisateur final.

6. Managed Integrations appelle le connecteur C2C et AWS . `ActivateUser` fonctionne pour terminer le flux de liaison de comptes et obtenir. `UserId`
7. Les intégrations gérées OAuth RedirectUrl renvoient (depuis la configuration de la politique de connecteur) la page d'authentification réussie à l'application client.

 Note

En cas d'échec, les intégrations gérées pour AWS IoT Device Management ajoutent les paramètres de requête `error` et `error_description` à l'URL en fournissant les détails de l'erreur à l'application client.

8. L'application client redirige l'utilisateur final vers le OAuthRedirectUrl. À ce stade, le front-end AssociationId de l'application connaît l'association dès la première étape.

Toutes les demandes ultérieures effectuées à partir d'intégrations gérées pour AWS IoT Device Management via le connecteur C2C vers la plateforme cloud tierce, telles que les commandes pour découvrir des appareils et envoyer des commandes, incluront le jeton d'accès OAuth2 .0.

Le schéma suivant montre la relation entre les principaux éléments de la liaison de comptes :

Mettre en œuvre les opérations d'interface du connecteur C2C

Managed Integrations for AWS IoT Device Management définit quatre opérations que vous AWS Lambda devez gérer pour être considéré comme un connecteur. Votre connecteur C2C doit implémenter chacune des opérations suivantes :

1. [AWS.ActivateUser](#)- Les intégrations gérées pour le AWS IoT Device Management service appellent cette API pour récupérer un identifiant utilisateur unique au monde associé au jeton OAuth2 .0 fourni. Cette opération peut éventuellement être utilisée pour effectuer toute exigence supplémentaire relative au processus de liaison de comptes.
2. [AWS.DiscoverDevices](#)- les intégrations gérées pour le service AWS IoT Device Management appelle cette API à votre connecteur pour découvrir les appareils des utilisateurs
3. [AWS.SendCommand](#)- les intégrations gérées pour le service AWS IoT Device Management appelle cette API à votre connecteur pour envoyer des commandes aux appareils des utilisateurs

4. [AWS.DeactivateUser](#)- les intégrations gérées pour le service AWS IoT Device Management appelle cette API à votre connecteur pour désactiver le jeton d'accès de l'utilisateur afin de dissocier votre serveur d'autorisation.

Les intégrations gérées pour AWS IoT Device Management toujours invoquer la fonction Lambda avec une charge utile de chaîne JSON via l'action. AWS Lambda `invokeFunction` Les opérations de demande doivent inclure un `operationName` champ dans chaque charge utile de demande. Pour en savoir plus, consultez [Invoke](#) dans le guide de référence des AWS Lambda API.

Chaque délai d'invocation est fixé à deux secondes, et si l'appel échoue, il sera réessayé cinq fois.

Le Lambda que vous implémentez pour votre connecteur analysera un élément de la charge utile `operationName` de la demande et implémentera les fonctionnalités correspondantes pour le mapper vers le cloud tiers :

```
public ConnectorResponse handleRequest(final ConnectorRequest request)
    throws OperationFailedException {
    Operation operation;
    try {
        operation = Operation.valueOf(request.payload().operationName());
    } catch (IllegalArgumentException ex) {
        throw new ValidationException(
            "Unknown operation '%s'".formatted(request.payload().operationName()),
            ex
        );
    }

    return switch (operation) {
        case ActivateUser -> activateUserManager.activateUser(request);
        case DiscoverDevices -> deviceDiscoveryManager.listDevices(request);
        case SendCommand -> sendCommandManager.sendCommand(request);
        case DeactivateUser -> deactivateUser.deactivateUser(request);
    };
}
```

Note

Le développeur du connecteur doit implémenter les `deactivateUser.deactivateUser` opérations

`activateUserManager.activateUser(request)deviceDiscoveryManager.listDevices()` et répertoriées dans l'exemple précédent.

L'exemple suivant détaille une demande de connecteur générique provenant d'intégrations gérées, dans laquelle des champs communs à chaque interface requise sont présents. Dans l'exemple, vous pouvez voir qu'il existe à la fois un en-tête de demande et une charge utile de demande. Les en-têtes de requête sont communs à toutes les interfaces d'exploitation.

```
{
  "header": {
    "auth": {
      "token": "ashriu32yr97feqy7afsaf",
      "type": "OAuth2.0"
    }
  },
  "payload": {
    "operationName": "AWS.SendCommand",
    "operationVersion": "1.0",
    "connectorId": "exampleId",
    ...
  }
}
```

En-têtes de demande par défaut

Les champs d'en-tête par défaut sont les suivants.

```
{
  "header": {
    "auth": {
      "token": string,    // end user's Access Token
      "type": ENUM ["OAuth2.0"],
    }
  }
}
```

Toute API hébergée par un connecteur doit traiter les paramètres d'en-tête suivants :

En-têtes et champs par défaut

Champ	Obligatoire/Facultatif	Description
<code>header:auth</code>	Oui	Informations d'autorisation fournies par le constructeur du connecteur C2C lors de son enregistrement du connecteur.
<code>header:auth:token</code>	Oui	Jeton d'autorisation de l'utilisateur généré par le fournisseur de cloud tiers et lié à <code>connectorAssociationID</code> .
<code>header:auth:type</code>	Oui	Le type d'autorisation requis.

Note

Le jeton d'accès de l'utilisateur final sera joint à toutes les demandes adressées à votre connecteur. Vous pouvez supposer que la liaison de compte entre l'utilisateur final et le client des intégrations gérées a déjà eu lieu.

Charge utile de la requête

Outre les en-têtes communs, chaque demande aura une charge utile. Bien que cette charge utile contienne des champs uniques pour chaque type d'opération, chaque charge utile possède un ensemble de champs par défaut qui seront toujours présents.

Champs de charge utile de la demande :

- `operationName`: opération d'une demande donnée, égale à l'une des valeurs suivantes : `AWS.ActivateUser`, `AWS.SendCommand`, `AWS.DiscoverDevices`, `AWS.DeactivateUser`.
- `operationVersion`: Chaque opération est versionnée afin de permettre son évolution dans le temps et de fournir une définition d'interface stable pour les connecteurs tiers. Les intégrations gérées transfèrent un champ de version dans la charge utile de toutes les demandes.
- `connectorId`: ID du connecteur auquel la demande a été envoyée.

En-têtes de réponse par défaut

Chaque opération répondra par une intégration gérée ACK pour AWS IoT Device Management qui confirmera que votre connecteur C2C a reçu la demande et a commencé à la traiter. Voici un exemple générique de cette réponse :

```
{
  "header":{
    "responseCode": 200
  },
  "payload":{
    "responseMessage": "Example response!"
  }
}
```

Chaque réponse d'opération doit avoir l'en-tête commun suivant :

```
{
  "header": {
    "responseCode": Integer
  }
}
```

Le tableau suivant répertorie l'en-tête de réponse par défaut :

En-tête et champ de réponse par défaut

Champ	Obligatoire/Facultatif	Commentaire
header:responseCode	Oui	ENUM de valeurs indiquant le statut d'exécution de la demande.

Dans les différentes interfaces de connecteur et schémas d'API décrits dans ce document, il existe un Message champ responseMessage or. Il s'agit d'un champ facultatif utilisé par le connecteur C2C Lambda pour répondre à n'importe quel contexte concernant la demande et son exécution. De préférence, toute erreur entraînant un code d'état autre que 200 doit inclure une valeur de message décrivant l'erreur.

Répondre aux demandes de fonctionnement du connecteur C2C avec l'API SendConnectorEvent

Managed Integrations for AWS IoT Device Management s'attend à ce que votre connecteur se comporte de manière asynchrone pour chaque AWS .SendCommand opération.

AWS.DiscoverDevices Cela signifie que la réponse initiale à ces opérations « reconnaît » simplement que votre connecteur C2C a reçu la demande.

À l'aide de l'API SendConnectorEvent, votre connecteur est censé envoyer les types d'événements de la liste ci-dessous à for AWS .DiscoverDevices and AWS .SendCommand operations, ainsi que des événements proactifs sur les appareils (tels que l'allumage et l'extinction manuels d'un voyant). Pour lire une explication détaillée de ces types d'événements et de leurs cas d'utilisation [Implémentez l'AWS. DiscoverDevices opération](#) et [Implémentez l'AWS. SendCommand opération](#), reportez-vous aux sections et [Envoyer des événements liés à un appareil avec l' API SendConnectorEvent](#).

Par exemple, si votre connecteur C2C reçoit une DiscoverDevices demande, les intégrations gérées pour AWS IoT Device Management s'attendent à ce qu'il réponde de manière synchrone avec le format de réponse défini ci-dessus. Vous devez ensuite appeler l'API SendConnectorEvent avec la structure de demande définie dans [Implémentez l'AWS. DiscoverDevices opération](#), pour un événement DEVICE_DISCOVERY. L'appel SendConnectorEvent d'API peut avoir lieu partout où vous avez accès aux informations d'identification Compte AWS Lambda de votre connecteur C2C. Le flux de découverte des appareils échoue tant que Managed Integrations for AWS IoT Device Management ne reçoit pas cet événement.

Note

L'appel d'API SendConnectorEvent peut également avoir lieu avant la réponse d'appel Lambda du connecteur C2C si nécessaire. Cependant, ce flux contredit le modèle asynchrone du développement logiciel.

- **SendConnectorEvent**- Votre connecteur appelle cette API Managed Integrations for AWS IoT Device Management pour envoyer les événements liés aux appareils aux intégrations gérées pour AWS IoT Device Management. Seuls 3 types d'événements sont acceptés par les intégrations gérées :
 - « DEVICE_DISCOVERY » — Cette opération événementielle doit être utilisée pour envoyer la liste des appareils découverts dans un cloud tiers pour un jeton d'accès spécifique.

- « `DEVICE_COMMAND_RESPONSE` » — Cette opération d'événement doit être utilisée pour envoyer un événement de périphérique spécifique à la suite de l'exécution d'une commande.
- « `DEVICE_EVENT` » — Cette opération événementielle doit être utilisée pour tout événement provenant de l'appareil qui n'est pas le résultat direct d'une commande basée sur l'utilisateur. Cela peut servir de type d'événement général pour signaler de manière proactive les modifications de l'état de l'appareil ou les notifications.

Implémentez l'AWS. `ActivateUser` opération

L'AWS. `ActivateUser` opération est requise pour les intégrations gérées permettant à AWS IoT Device Management de récupérer un identifiant utilisateur à partir du jeton OAuth2 .0 d'un utilisateur final. Les intégrations gérées pour AWS IoT Device Management transmettront le OAuth jeton dans l'en-tête de la demande et s'attendent à ce que votre connecteur inclue l'identifiant utilisateur unique au monde dans la charge utile de la réponse. Cette opération a lieu après un flux de liaison de comptes réussi.

La liste suivante décrit les exigences relatives à votre connecteur afin de faciliter un flux AWS. `Activate` d'utilisateurs réussi.

- Votre connecteur Lambda C2C peut traiter AWS. `ActivateUser` un message de demande d'opération provenant d'intégrations gérées pour AWS IoT Device Management.
- Votre connecteur Lambda C2C peut déterminer un identifiant utilisateur unique à partir d'un jeton .0 OAuth2 fourni. Normalement, il peut être extrait du jeton lui-même, s'il s'agit d'un jeton JWT, ou demandé au serveur d'autorisation par le jeton.

Flux de travail AWS. `ActivateUser`

1. Intégrations gérées pour AWS IoT Device Management appeler votre connecteur C2C Lambda avec la charge utile suivante :

```
{
  "header": {
    "auth": {
      "token": "ashriu32yr97feqy7afsaf",
      "type": "OAuth2.0"
    }
  },
  "payload": {
```

```
"operationName": "AWS.ActivateUser",
"operationVersion": "1.0.0",
"connectorId": "Your-Connector-ID",
}
}
```

2. Le connecteur C2C détermine l'ID utilisateur, soit à partir du jeton, soit en interrogeant votre serveur de ressources tiers, à inclure dans la `AWS.ActivateUser` réponse.
3. Le connecteur C2C répond à `AWS.ActivateUser` l'appel de l'opération Lambda, y compris la charge utile par défaut ainsi que l'identifiant utilisateur correspondant dans le champ `userId`

```
{
  "header": {
    "responseCode": 200
  },
  "payload": {
    "responseMessage": "Successfully activated user with connector-id `Your-Connector-Id.",
    "userId": "123456"
  }
}
```

Implémentez l'AWS. DiscoverDevices opération

La découverte des appareils aligne la liste des appareils physiques détenus par l'utilisateur final avec les représentations numériques de ces appareils gérés dans le cadre d'intégrations gérées pour AWS IoT Device Management. Elle est effectuée par un AWS client sur des appareils appartenant à l'utilisateur final uniquement une fois que la liaison du compte est terminée entre l'utilisateur et les intégrations gérées pour AWS IoT Device Management. La découverte d'appareils est un processus asynchrone dans le cadre duquel les intégrations gérées pour AWS IoT Device Management appellent un connecteur pour lancer la demande de découverte d'appareils. Un connecteur C2C renvoie une liste des appareils des utilisateurs finaux découverts de manière asynchrone avec un identifiant de référence (appelé `deviceDiscoveryId`) généré par les intégrations gérées.

Le schéma suivant illustre le flux de travail de découverte des appareils entre l'utilisateur final et les intégrations gérées pour AWS IoT Device Management :

AWS.DiscoverDevices flux de travail

1. Le client lance le processus de découverte de l'appareil pour le compte de l'utilisateur final.
2. Managed Integrations for AWS IoT Device Management génère un identifiant de référence appelé `deviceDiscoveryId` pour la demande de découverte de l'appareil générée par le AWS client.
3. Managed Integrations for AWS IoT Device Management envoie une demande de découverte de périphériques au connecteur C2C à l'aide de l'interface d'`AWS.DiscoverDevicesexploitation`, y compris une copie valide OAuth `accessToken` de l'utilisateur final ainsi que le `deviceDiscoveryId`.
4. Vos magasins `deviceDiscoveryId` de connecteurs seront inclus dans l'`DEVICE_DISCOVERY` événement. Cet événement contiendra également une liste des appareils de l'utilisateur final découverts et doit être envoyé aux intégrations gérées pour AWS IoT Device Management avec l'`SendConnectorEventAPI` en tant qu'`DEVICE_DISCOVERY` événement.
5. Votre connecteur C2C doit appeler le serveur de ressources pour récupérer tous les appareils appartenant à l'utilisateur final.
6. Votre connecteur C2C Lambda répond à l'appel Lambda `invokeFunction ()` avec la réponse `ACK` aux intégrations gérées pour AWS IoT Device Management, agissant comme réponse initiale à l'opération. `AWS.DiscoverDevices` Les intégrations gérées informent le client doté d'un `ACK` du processus de découverte des appareils qu'il a lancé.
7. Votre serveur de ressources vous envoie une liste des appareils détenus et exploités par l'utilisateur final.
8. Votre connecteur convertit chaque appareil de l'utilisateur final en intégrations gérées pour le format d'appareil requis pour AWS IoT Device Management `ConnectorDeviceId`, `ConnectorDeviceName` y compris le rapport de capacité pour chaque appareil.
9. Le connecteur C2C fournit également le `UserId` nom du propriétaire des appareils découverts. Il peut être extrait de votre serveur de ressources soit dans le cadre de la liste des appareils, soit dans le cadre d'un appel distinct en fonction de l'implémentation de votre serveur de ressources.
10. Ensuite, votre connecteur C2C appellera les intégrations gérées pour l'API AWS IoT Device Management via SigV4 à l'aide d'informations d'identification de Compte AWS et avec le paramètre de fonctionnement défini comme « `DEVICE_DISCOVERY` ». `SendConnectorEvent` Chaque appareil de la liste des appareils envoyés aux intégrations gérées pour AWS IoT Device Management sera représenté par des paramètres spécifiques à l'appareil tels que `connectorDeviceId`, `connectorDeviceName`, et `a. capabilityReport`.

- En fonction de la réponse de votre serveur de ressources, vous devez informer les intégrations gérées pour AWS IoT Device Management en conséquence.

Par exemple, si votre serveur de ressources reçoit une réponse paginée à la liste des appareils découverts pour un utilisateur final, vous pouvez envoyer pour chaque sondage un événement d'DEVICE_DISCOVERY opération individuel, avec un statusCode paramètre de 3xx. Si la découverte de votre appareil est toujours en cours, répétez les étapes 5, 6 et 7.

11. Managed Integrations for AWS IoT Device Management envoie une notification au client concernant la découverte des appareils de l'utilisateur final.
12. Si votre connecteur C2C envoie un événement de DEVICE_DISCOVERY fonctionnement avec le statusCode paramètre mis à jour avec une valeur de 200, les intégrations gérées informeront le client de la fin du processus de découverte des appareils.

Important

Les étapes 7 à 11 peuvent avoir lieu avant l'étape 6, si vous le souhaitez. Par exemple, si votre plate-forme tierce dispose d'une API pour répertorier les appareils d'un utilisateur final, l'événement DEVICE_DISCOVERY peut être envoyé `SendConnectorEvent` avant que le connecteur Lambda C2C ne réponde avec l'ACK typique.

Exigences relatives au connecteur C2C pour la découverte des appareils

La liste suivante décrit les exigences relatives à votre connecteur C2C afin de faciliter la découverte réussie des appareils.

- Le connecteur C2C Lambda a peut traiter un message de demande de découverte d'appareil provenant d'intégrations gérées pour AWS IoT Device Management et gérer l'opération.
`AWS.DiscoverDevices`
- Votre connecteur C2C peut appeler les intégrations gérées pour AWS IoT Device Management APIs via SigV4 en utilisant les informations d'identification du connecteur Compte AWS utilisé pour enregistrer le connecteur.

Processus de découverte des appareils

Les étapes suivantes décrivent le processus de découverte des appareils avec votre connecteur C2C et les intégrations gérées pour AWS IoT Device Management.

Processus de découverte des appareils

1. Les intégrations gérées déclenchent la découverte des appareils :

- Envoyez une requête POST à DiscoverDevices avec la charge utile JSON suivante :

```
/DiscoverDevices
{
  "header": {
    "auth": {
      "token": "ashriu32yr97feqy7afsaf",
      "type": "OAuth2.0"
    }
  },
  "payload": {
    "operationName": "AWS.DiscoverDevices",
    "operationVersion": "1.0",
    "connectorId": "Your-Connector-Id",
    "deviceDiscoveryId": "12345678"
  }
}
```

2. Le connecteur accuse réception de la découverte :

- Le connecteur envoie un accusé de réception avec la réponse JSON suivante :

```
{
  "header": {
    "responseCode": 200
  },
  "payload": {
    "responseMessage": "Discovering devices for discovery-job-id '12345678' with connector-id `Your-Connector-Id`"
  }
}
```

3. Le connecteur envoie un événement Device Discovery :

- Envoyez une requête POST à /connector-event/*{your_connector_id}* avec la charge utile JSON suivante :

```
AWS API - /SendConnectorEvent
URI - POST /connector-event/{your_connector_id}
```



```

{
  "UserId": "6109342",
  "Operation": "DEVICE_DISCOVERY",
  "OperationVersion": "1.0",
  "StatusCode": 200,
  "DeviceDiscoveryId": "12345678",
  "ConnectorId": "Your_connector_Id",
  "Message": "Device discovery for discovery-job-id '12345678' successful",
  "Devices": [
    {
      "ConnectorDeviceId": "Your_Device_Id_1",
      "ConnectorDeviceName": "Your-Device-Name",
      "CapabilityReport": {
        "nodeId": "1",
        "version": "1.0.0",
        "endpoints": [{
          "id": "1",
          "deviceTypes": ["Camera"],
          "clusters": [{
            "id": "0x0006",
            "revision": 1,
            "attributes": [{
              "id": "0x0000",
            }],
            "commands": ["0x00", "0x01"],
            "events": ["0x00"]
          }]
        }]
      }
    ]
  }
}

```

Construit un CapabilityReport pour l'événement DISCOVER_DEVICES

Comme le montre la structure d'événements définie ci-dessus, chaque appareil signalé dans un événement DISCOVER_DEVICES, servant de réponse à une `AWS.DiscoverDevices` opération, aura besoin d'un `CapabilityReport` pour décrire les capacités du périphérique correspondant. Un `CapabilityReport` indique les intégrations gérées pour les fonctionnalités des appareils AWS IoT Device Management dans un format compatible avec Matter. Les champs suivants doivent être renseignés dans le `CapabilityReport` :

- `nodeId`, String : identifiant du nœud de l'appareil contenant les éléments suivants endpoints
- `version`, String : version de ce nœud de périphérique, définie par le développeur du connecteur
- `endpoints`, Liste <Cluster>: liste des AWS implémentations du modèle de données Matter prises en charge par ce point de terminaison de l'appareil.
 - `id`, String : identifiant du point de terminaison défini par le développeur du connecteur
 - `deviceTypes`, Liste <String>: Liste des types d'appareils capturés par ce terminal, c'est-à-dire « Appareil photo ».
- `clusters`, Liste <Cluster>: liste des AWS implémentations du modèle de données Matter pris en charge par ce point de terminaison.
 - `id`, String : identifiant du cluster tel que défini par la norme Matter.
 - `revision`, Integer : numéro de révision du cluster tel que défini par la norme Matter.
 - `attributes`, Carte<String, Object> : carte des identifiants d'attributs et de leurs valeurs d'état actuelles de l'appareil correspondantes, avec les identifiants et les valeurs valides définis par la norme de matière.
 - `id`, String : ID d'attribut tel que défini par AWS les implémentations du modèle de données Matter.
 - `value`, Objet : valeur actuelle de l'attribut défini par l'ID de l'attribut. Le type de « valeur » peut changer en fonction de l'attribut. Le `value` champ est facultatif pour chaque attribut et ne doit être inclus que si votre connecteur lambda peut déterminer l'état actuel lors de la découverte.
 - `commands`, Liste <String>: liste des commandes IDs prises en charge par ce cluster, telle que définie par la norme Matter.
 - `events`, Liste <String>: liste des événements IDs pris en charge par ce cluster, telle que définie par la norme Matter.

Pour la liste actuelle des fonctionnalités prises en charge et leurs [AWS implémentations correspondantes du modèle de données de matière](#), reportez-vous à la dernière version de la documentation du modèle de données.

Implémentez l'AWS. SendCommand opération

L'AWS . SendCommandopération permet aux intégrations gérées pour AWS IoT Device Management d'envoyer des commandes initiées par l'utilisateur final via le AWS client à votre serveur de ressources. Votre serveur de ressources peut prendre en charge plusieurs types de

périphériques, chaque type ayant son propre modèle de réponse. L'exécution de commandes est un processus asynchrone dans le cadre duquel les intégrations gérées pour AWS IoT Device Management envoient une demande d'exécution de commande avec un « TraceID », que votre connecteur inclura dans une réponse de commande renvoyée aux intégrations gérées via l'API « SendConnectorEvent ». Les intégrations gérées pour AWS IoT Device Management attendent du serveur de ressources qu'il renvoie une réponse confirmant la réception de la commande, mais n'indiquant pas nécessairement que la commande a été exécutée.

Le schéma suivant illustre le flux d'exécution des commandes avec un exemple dans lequel l'utilisateur final essaie d'allumer les lumières de sa maison :

Flux de travail d'exécution des commandes des appareils

1. Un utilisateur final envoie une commande pour allumer une lampe à l'aide de l'application du AWS client.
2. Le client transmet les informations de commande aux intégrations gérées pour AWS IoT Device Management avec les informations relatives aux appareils de l'utilisateur final.
3. Les intégrations gérées génèrent un « TraceID » que votre connecteur utilisera pour renvoyer les réponses aux commandes au service.
4. Les intégrations gérées pour AWS IoT Device Management envoient la demande de commande à votre connecteur via l'interface d'AWS . SendCommandexploitation.
 - La charge utile définie par cette interface comprend l'identifiant de l'appareil, les commandes de l'appareil formulées sous la forme Matterendpoints/clusters/commands, le jeton d'accès de l'utilisateur final et d'autres paramètres requis.
5. Votre connecteur stocke les éléments `traceId` à inclure dans la réponse à la commande.
 - Votre connecteur traduit la demande de commande d'intégrations gérées dans le format approprié de votre serveur de ressources.
6. Votre connecteur obtient le jeton `UserId` d'accès fourni à l'utilisateur final et l'associe à la commande.
 - a. Ils `UserId` peuvent être récupérés auprès de votre serveur de ressources à l'aide d'un appel séparé ou extraits du jeton d'accès dans le cas de JWT et de jetons similaires.
 - b. La mise en œuvre dépend de votre serveur de ressources et des détails de votre jeton d'accès.

7. Votre connecteur appelle le serveur de ressources pour qu'il « allume » la lumière de l'utilisateur final.
8. Le serveur de ressources interagit avec le périphérique.
 - a. Le connecteur transmet aux intégrations gérées pour AWS IoT Device Management que le serveur de ressources a délivré la commande, en répondant par un ACK comme réponse de commande synchrone initiale.
 - b. Les intégrations gérées le retransmettent ensuite à l'application client.
9. Une fois que l'appareil a allumé la lumière, cet événement est capturé par votre serveur de ressources.
10. Votre serveur de ressources envoie l'événement du périphérique au connecteur.
11. Votre connecteur transforme l'événement de périphérique généré par le serveur de ressources en intégrations gérées, type d'opération d'événement `DEVICE_COMMAND_RESPONSE`.
12. Votre connecteur appelle `SendConnectorEventAPI` avec l'opération « `DEVICE_COMMAND_RESPONSE` ».
 - Il joint les intégrations `traceId` fournies par les intégrations gérées pour AWS IoT Device Management dans la demande initiale.
13. Les intégrations gérées informent le client du changement d'état de l'appareil de l'utilisateur final.
14. Le client informe l'utilisateur final que le voyant de l'appareil est allumé.

 Note

La configuration de votre serveur de ressources détermine la logique de gestion des messages de demande de commande et de réponse du périphérique ayant échoué. Cela inclut les tentatives de nouvelle tentative de message utilisant le même `ReferencID` pour la commande.

Exigences relatives au connecteur C2C pour l'exécution des commandes de l'appareil

La liste suivante décrit les exigences relatives à votre connecteur C2C afin de faciliter l'exécution réussie des commandes du périphérique.

- Le connecteur Lambda C2C peut `AWS . SendCommand` traiter les messages de demande d'opération provenant d'intégrations gérées pour AWS IoT Device Management.

- Votre connecteur C2C doit suivre les commandes envoyées à votre serveur de ressources et le mapper avec le `TraceID` approprié.
- Vous pouvez appeler des intégrations gérées pour les API du service AWS IoT Device Management via SigV4 en utilisant les AWS informations d'identification Compte AWS utilisées pour enregistrer le connecteur C2C.

1. Les intégrations gérées envoient une commande au connecteur (reportez-vous à l'étape 4 du schéma précédent).

```

•
  /Send-Command
  {
    "header": {
      "auth": {
        "token": "ashriu32yr97feqy7afsaf",
        "type": "OAuth2.0"
      }
    },
    "payload": {
      "operationName": "AWS.SendCommand",
      "operationVersion": "1.0",
      "connectorId": "Your-Connector-Id",
      "connectorDeviceId": "Your_Device_Id",
      "traceId": "traceId-3241u78123419",
      "endpoints": [{
        "id": "1",
        "clusters": [{
          "id": "0x0202",
          "commands": [{
            "0xff01": {
              "0x0000": "3"
            }
          ]
        }]
      }]
    }
  }

```

2. Commande ACK du connecteur C2C (reportez-vous à l'étape 7 du schéma précédent où le connecteur envoie l'ACK aux intégrations gérées pour AWS IoT Device Management Service).

•

```
{
  "header":{
    "responseCode":200
  },
  "payload":{
    "responseMessage": "Successfully received send-command request for
connector 'Your-Connector-Id' and connector-device-id 'Your_Device_Id'"
  }
}
```

3. Le connecteur envoie un événement Device Command Response (reportez-vous à l'étape 11 du schéma précédent).

•

AWS-API: /SendConnectorEvent
 URI: POST /connector-event/{*Your-Connector-Id*}

```
{
  "UserId": "End-User-Id",
  "Operation": "DEVICE_COMMAND_RESPONSE",
  "OperationVersion": "1.0",
  "StatusCode": 200,
  "Message": "Example message",
  "ConnectorDeviceId": "Your_Device_Id",
  "TraceId": "traceId-3241u78123419",
  "MatterEndpoint": {
    "id": "1",
    "clusters": [{
      "id": "0x0202",
      "attributes": [
        {
          "0x0000": "3"
        }
      ],
      "commands": [
        "0xff01":
        {
          "0x0000": "3"
        }
      ]
    }
  ]
}
```

}

Note

Les modifications de l'état de l'appareil résultant de l'exécution d'une commande ne seront pas reflétées dans les intégrations gérées pour AWS IoT Device Management tant que l'événement `DEVICE_COMMAND_RESPONSE` correspondant n'aura pas été reçu via l'API. `SendConnectorEvent` Cela signifie que tant que Managed Integrations ne recevra pas l'événement de l'étape 3 précédente, que votre réponse d'appel du connecteur indique un succès ou non, l'état de l'appareil ne sera pas mis à jour.

Interprétation des « points de terminaison » inclus dans `AWS.SendCommand` demande

Les intégrations gérées utiliseront les fonctionnalités de l'appareil signalées lors de la découverte de l'appareil pour déterminer les commandes qu'un appareil peut accepter. Les capacités de chaque appareil sont modélisées grâce à AWS des implémentations du modèle Matter Data ; ainsi, toutes les commandes entrantes seront dérivées du champ « commandes » d'un cluster donné. Il est de la responsabilité de votre connecteur d'analyser le champ ``endpoints``, de déterminer la commande Matter correspondante et de la traduire de telle sorte que la bonne commande atteigne le périphérique. Cela implique généralement de traduire le modèle de données Matter dans les demandes d'API associées.

Une fois la commande exécutée, votre connecteur détermine quels « attributs » définis par les AWS implémentations du Matter Data Model ont changé en conséquence. Ces modifications sont ensuite signalées aux intégrations gérées pour AWS IoT Device Management via les événements API `DEVICE_COMMAND_RESPONSE` envoyés avec l'API. `SendConnectorEvent`

Considérez le champ ``endpoints`` inclus dans l'exemple de charge utile suivant : `AWS.SendCommand`

```
"endpoints": [{
  "id": "1",
  "clusters": [{
    "id": "0x0202",
    "commands": [{
      "0xff01":
        {
          "0x0000": "3"
        }
    ]
  }]
}]
```

```
    }]  
  }]  
}]
```

À partir de cet objet, le connecteur peut déterminer les éléments suivants :

1. Définissez les informations du point de terminaison et du cluster :
 - a. Réglez le point `id` de terminaison sur « 1 ».
-  **Note**

Si un appareil définit plusieurs points de terminaison tels qu'un seul cluster (par exemple On/Off) can control multiple capabilities (i.e. turn a light on/off as well as turning a strobe on/off), cet identifiant est utilisé pour acheminer la commande vers la fonctionnalité appropriée.
- b. Réglez le cluster `id` sur « 0x0202 » (cluster de contrôle du ventilateur).
 2. Définissez les informations de commande :
 - a. Définissez l'identifiant de commande sur « 0xff01 » (commande Update State définie par AWS).
 - b. Mettez à jour les identificateurs d'attributs inclus avec les valeurs fournies dans la demande.
 3. Mettez à jour l'attribut :
 - a. Définissez l'identifiant d'attribut sur « 0x0000 » (FanMode attribut du cluster de contrôle des ventilateurs).
 - b. Définissez la valeur de l'attribut sur « 3 » (vitesse du ventilateur élevée).

Managed Integrations a défini deux types de commandes « personnalisés » qui ne sont pas strictement définis par AWS les implémentations du modèle de données Matter : les commandes ReadState et UpdateState . Pour obtenir et définir les attributs de cluster définis par Matter, les intégrations gérées enverront à votre connecteur une AWS . SendCommand demande avec une commande IDs relative à UpdateState (id : 0xff01) ou ReadState (id : 0xff02), avec les paramètres correspondants des attributs qui doivent être mis à jour ou lus. Ces commandes peuvent être invoquées pour TOUT type d'appareil pour les attributs définis comme mutables (modifiables) ou

recupérables (lisibles) à partir de l' AWS implémentation correspondante du modèle de données de matière.

Envoyer des événements liés à un appareil avec l' SendConnectorEventAPI

Vue d'ensemble des événements déclenchés par un

Bien que l'SendConnectorEventAPI soit utilisée pour répondre AWS .SendCommand et effectuer des AWS.DiscoverDevices opérations de manière asynchrone, elle est également utilisée pour informer les intégrations gérées de tout événement initié par l'appareil. Les événements initiés par un appareil peuvent être définis comme tout événement généré par un appareil sans commande initiée par l'utilisateur. Ces événements peuvent inclure, sans toutefois s'y limiter, les changements d'état de l'appareil, la détection de mouvements, le niveau de charge de la batterie, etc. Vous pouvez renvoyer ces événements aux intégrations gérées à l'aide de l'SendConnectorEventAPI avec l'opération DEVICE_EVENT.

La section suivante utilise une caméra intelligente installée dans une maison comme exemple pour expliquer plus en détail le déroulement de ces événements :

Flux de travail lié aux événements liés

1. Votre caméra détecte un mouvement pour lequel elle génère un événement qui est envoyé à votre serveur de ressources.
2. Votre serveur de ressources traite l'événement et l'envoie à votre connecteur C2C.
3. Votre connecteur traduit cet événement en intégrations gérées pour l'DEVICE_EVENTinterface AWS IoT Device Management.
4. Votre connecteur C2C envoie cet événement d'appareil aux intégrations gérées à l'aide de l'SendConnectorEventAPI avec l'opération définie sur « DEVICE_EVENT ».
5. Les intégrations gérées identifient le client concerné et lui transmettent cet événement.
6. Le client reçoit cet événement et l'affiche à l'utilisateur via un identifiant utilisateur.

Pour plus d'informations sur le fonctionnement de l'SendConnectorEventAPI, consultez SendConnectorEvent le guide de référence de l'API sur les intégrations gérées pour AWS IoT Device Management.

Exigences relatives aux événements déclenchés par l'appareil

Voici quelques exigences relatives aux événements déclenchés par l'appareil.

- Votre ressource de connecteur C2C doit être en mesure de recevoir des événements de périphérique asynchrones depuis votre serveur de ressources
- Votre ressource de connecteur C2C doit être en mesure d'appeler des intégrations gérées pour les API du service AWS IoT Device Management via SigV4 en utilisant les AWS informations d'identification Compte AWS utilisées pour enregistrer le connecteur C2C.

L'exemple suivant illustre un connecteur qui envoie un événement provenant d'un appareil via l' `SendConnectorEvent` API :

```
AWS-API: /SendConnectorEvent
URI: POST /connector-event/{Your-Connector-Id}

{
  "UserId": "Your-End-User-ID",
  "Operation": "DEVICE_EVENT",
  "OperationVersion": "1.0",
  "StatusCode": 200,
  "Message": None,
  "ConnectorDeviceId": "Your_Device_Id",
  "MatterEndpoint": {
    "id": "1",
    "clusters": [{
      "id": "0x0202",
      "attributes": [
        {
          "0x0000": "3"
        }
      ]
    }]
  }
}
```

Dans l'exemple suivant, nous voyons ce qui suit :

- Cela provient du point de terminaison de l'appareil dont l'ID est égal à 1.
- La capacité du périphérique à laquelle cet événement se rapporte possède un ID de cluster 0x0202, appartenant au cluster de sujets Fan Control.

- L'attribut qui a changé a l'ID 0x000, correspondant à l'énumération du mode ventilateur au sein du cluster. Il a été mis à jour à la valeur 3, correspondant à la valeur de High.
- Comme `connectorId` il s'agit d'un paramètre renvoyé par le service cloud lors de sa création, les connecteurs doivent effectuer des requêtes à l'aide `GetCloudConnector` et filtrer par `lambdaARN`. Le propre lambda ARN est interrogé à l'aide de l'API `Lambda.get_function_url_config` Cela permet d'`CloudConnectorId` accéder dynamiquement dans le lambda, et non de le configurer statiquement comme précédemment.

Implémentez l'AWS. `DeactivateUser` opération

Vue d'ensemble de la désactivation des utilisateurs

La désactivation des jetons d'accès utilisateur fournis est requise lorsqu'un AWS client supprime son compte client ou lorsqu'un utilisateur final souhaite dissocier son compte du système du système du AWS client. Dans les deux cas d'utilisation, les intégrations gérées doivent faciliter ce flux de travail à l'aide du connecteur C2C.

L'image ci-dessous illustre la dissociation d'un compte d'utilisateur final du système

Flux de travail de désactivation des utilisateurs

1. L'utilisateur lance le processus de dissociation entre le compte du AWS client et le serveur d'autorisation tiers associé au connecteur C2C.
2. Le client initie la suppression de l'association de l'utilisateur par le biais d'intégrations gérées pour AWS IoT Device Management.
3. Managed Integrations lance le processus de désactivation via une demande adressée à votre connecteur via l'AWS. `DeactivateUser` interface Operation.
 - Le jeton d'accès `/user` est inclus dans l'en-tête de la demande.
4. Votre connecteur C2C accepte la demande et appelle votre serveur d'autorisation pour révoquer le jeton et tout accès qu'il fournit.
 - Par exemple, les événements provenant d'un compte utilisateur non lié ne doivent plus être envoyés aux intégrations gérées après leur exécution. `AWS.DeactivateUser`
5. Votre serveur d'autorisation révoque l'accès et renvoie une réponse à votre connecteur C2C.

6. Votre connecteur C2C envoie aux intégrations gérées pour AWS IoT Device Management un ACK indiquant que le jeton d'accès de l'utilisateur a été révoqué.
7. Les intégrations gérées suppriment toutes les ressources appartenant à l'utilisateur final qui étaient associées à votre serveur de ressources.
8. Managed Integrations envoie un ACK au client, indiquant que toutes les associations relatives à votre système ont été supprimées.
9. Le client informe l'utilisateur final que son compte a été dissocié de votre plateforme.

AWS.DeactivateUser exigences

- La fonction Lambda du connecteur C2C reçoit un message de demande des intégrations gérées pour gérer l'opération. `AWS.DeactivateUser`
- Le connecteur C2C doit révoquer le jeton OAuth2.0 fourni et le jeton d'actualisation correspondant de l'utilisateur au sein de votre serveur d'autorisation.

Voici un exemple de `AWS.DeactivateUser` demande que votre connecteur recevra :

```
{
  "header": {
    "auth": {
      "token": "ashriu32yr97feqy7afsaf",
      "type": "OAuth2.0"
    }
  },
  "payload": {
    "operationName": "AWS.DeactivateUser"
    "operationVersion": "1.0"
    "connectorId": "Your-connector-Id"
  }
}
```

Appelez votre connecteur C2C

AWS Lambda permet des politiques basées sur les ressources pour autoriser qui peut invoquer un Lambda. Les intégrations gérées pour AWS IoT Device Management étant une Service AWS, vous

devez autoriser les intégrations gérées à appeler votre connecteur Lambda C2C via la politique de ressources.

Attachez une politique de ressources avec au moins les autorisations minimales suivantes à votre connecteur C2C Lambda. Cela fournit des intégrations gérées avec des privilèges d'appel de fonctions Lambda. Cette politique inclut une Condition clé pour vous aider à limiter l'utilisabilité de votre connectorId site aux seuls utilisateurs prévus.

JSON

```
{
  "Version": "2012-10-17",
  "Id": "default",
  "Statement": [
    {
      "Sid": "Your-Desired-Policy-ID",
      "Effect": "Allow",
      "Principal": {
        "Service": "iotmanagedintegrations.amazonaws.com"
      },
      "Action": "lambda:InvokeFunction",
      "Resource": "arn:aws:lambda:ca-central-1:444455556666:function:connector-
lambda-name",
      "Condition": {
        "StringEquals": {
          "aws:SourceArn": "arn:aws:iotmanagedintegrations:ca-
central-1:444455556666:account-association/account-association-id"
        }
      }
    }
  ]
}
```

Ajoutez des autorisations à votre rôle IAM

Toutes les intégrations gérées APIs nécessitent l'authentification AWS SigV4 pour être invoquées. SigV4 est un protocole de signature permettant d'authentifier les demandes d' AWS API à l'aide de vos informations d'identification Compte AWS . Le rôle IAM que vous utilisez pour appeler les intégrations gérées APIs doit disposer des autorisations suivantes pour pouvoir invoquer correctement les : APIs

```
"Version": "2012-10-17",
"Statement": [
{
  "Sid": "Statement1",
  "Effect": "Allow",
  "Action": [
    "iotmanagedintegrations:Your-Required-Actions"
  ],
  "Resource": [
    "Your-Resource"
  ]
}]
}
```

Pour plus d'informations sur l'ajout de ces autorisations, contactez Support.

Ressources supplémentaires

Pour enregistrer votre connecteur C2C, vous aurez besoin des éléments suivants :

- L'ARN Lambda désignant le connecteur que vous souhaitez enregistrer.

Testez manuellement votre connecteur C2C

Pour tester manuellement votre connecteur C2C end-to-end, vous devez simuler à la fois le client et l'utilisateur final.

Vous aurez besoin des ressources suivantes :

- Un AWS Lambda ARN désignant le connecteur que vous souhaitez tester.
- Un compte utilisateur de test OAuth 2.0 depuis votre plateforme cloud.
- Un connecteur enregistré avec des intégrations gérées pour AWS IoT Device Management. Pour de plus amples informations, veuillez consulter [Utiliser un connecteur C2C \(Cloud-to-Cloud\)](#).

Utiliser un connecteur C2C (Cloud-to-Cloud)

Un connecteur C2C gère la traduction des messages de demande et de réponse et permet la communication entre les intégrations gérées et le cloud d'un fournisseur tiers. Il facilite le contrôle

unifié entre différents types d'appareils, plateformes et protocoles, permettant d'intégrer et de gérer des appareils tiers.

La procédure suivante répertorie les étapes d'utilisation du connecteur C2C.

Étapes d'utilisation du connecteur C2C :

1. CreateCloudConnector

Configurez un connecteur pour permettre une communication bidirectionnelle entre vos intégrations gérées et les clouds de fournisseurs tiers.

Lors de la configuration du connecteur, fournissez les informations suivantes :

- Nom : Choisissez un nom descriptif pour le connecteur.
- Description : fournissez un bref résumé de l'objectif et des fonctionnalités du connecteur.
- AWS Lambda ARN : Spécifiez l'Amazon Resource Name (ARN) de la AWS Lambda fonction qui alimentera le connecteur.

Créez et déployez une AWS Lambda fonction qui communique avec un fournisseur tiers APIs pour créer un connecteur. Ensuite, appelez l'[CreateCloudConnector](#) API dans les intégrations gérées et fournissez la AWS Lambda fonction ARN pour l'enregistrement. Assurez-vous que la AWS Lambda fonction est déployée dans le AWS compte où vous avez créé le connecteur dans les intégrations gérées. Un identifiant de connecteur unique vous sera attribué pour identifier l'intégration.

Exemple de demande et de réponse à l' CreateCloudConnector API :

Request:

```
{
  "Name": "CreateCloudConnector",
  "Description": "Testing for C2C",
  "EndpointType": "LAMBDA",
  "EndpointConfig": {
    "lambda": {
      "arn": "arn:aws:lambda:us-east-1:xxxxxx:function:TestingConnector"
    }
  },
  "ClientToken": "abc"
```

```
}

Response:

{
    "Id": "string"
}
```

Flux de création :

 Note

Utilisez le [GetCloudConnector](#), [UpdateCloudConnectorDeleteCloudConnector](#), et selon [ListCloudConnectors](#) APIs les besoins pour cette procédure.

2. CreateConnectorDestination

Configurez les destinations pour fournir les paramètres et les informations d'authentification dont les connecteurs ont besoin pour établir des connexions sécurisées avec les clouds de fournisseurs tiers. Utilisez Destinations pour enregistrer vos informations d'authentification tierces avec des intégrations gérées, telles que les détails d'autorisation OAuth 2.0, y compris l'URL d'autorisation, le schéma d'authentification et l'emplacement des informations d'identification qu'ils contiennent. AWS Secrets Manager

Conditions préalables

Avant de créer un ConnectorDestination, vous devez :

- Appelez l'[CreateCloudConnector](#) API pour créer un connecteur. L'ID renvoyé par la fonction est utilisé dans l'appel d'[CreateConnectorDestination](#) API.
- Récupérez le `tokenUrl` pour la plate-forme 3P du connecteur. (Vous pouvez échanger un `AuthCode` contre un `AccessToken`).
- Récupérez l'`AuthURL` de la plate-forme 3P du connecteur. (Les utilisateurs finaux peuvent s'authentifier à l'aide de leur nom d'utilisateur et de leur mot de passe).
- Utilisez le `clientId` et `clientSecret` (depuis la plateforme 3P) dans le gestionnaire secret de votre compte.

Exemple de demande et de réponse à l' CreateConnectorDestination API :

Request:

```
{
  "Name": "CreateConnectorDestination",
  "Description": "CreateConnectorDestination",
  "AuthType": "OAUTH",
  "AuthConfig": {
    "oAuth": {
      "authUrl": "https://xxxx.com/oauth2/authorize",
      "tokenUrl": "https://xxxx/oauth2/token",
      "scope": "testScope",
      "tokenEndpointAuthenticationScheme": "HTTP_BASIC",
      "oAuthCompleteRedirectUrl": "about:blank",
      "proactiveRefreshTokenRenewal": {
        "enabled": false,
        "DaysBeforeRenewal": 30
      }
    }
  },
  "CloudConnectorId": "<connectorId>", // The connectorID instance from response
  "SecretsManager": {
    "arn": "arn:aws:secretsmanager:*****:secret:*****",
    "versionId": "*****"
  },
  "ClientToken": "****"
}
```

Response:

```
{
  "Id": "string"
}
```

Flux de création de destinations cloud :

Note

Utilisez le [GetCloudConnector](#), [UpdateCloudConnectorDeleteCloudConnector](#), et selon [ListCloudConnectors](#) APIs les besoins pour cette procédure.

3. CreateAccountAssociation

Les associations représentent les relations entre les comptes cloud tiers des utilisateurs finaux et une destination de connecteur. Après avoir créé une association et lié les utilisateurs finaux à des intégrations gérées, leurs appareils sont accessibles via un identifiant d'association unique. Cette intégration permet trois fonctions clés : la découverte d'appareils, l'envoi de commandes et la réception d'événements.

Conditions préalables

Avant de créer un, AccountAssociationvous devez effectuer les opérations suivantes :

- Appelez l'[CreateConnectorDestination](#)API pour créer une destination. L'ID renvoyé par la fonction est utilisé dans l'appel [CreateAccountAssociation](#)d'API.
- Appelez l'[CreateAccountAssociation](#)API.

Exemple de demande et de réponse à l' CreateAccountAssociation API :

Request:

```
{
  "Name": "CreateAccountAssociation",
  "Description": "CreateAccountAssociation",
  "ConnectorDestinationId": "<destinationId>", //The destinationID from
  destination creation.
  "ClientToken": "****"
}
```

Response:

```
{
  "Id":"string"
}
```

Note

Utilisez le [GetCloudConnector](#), [UpdateCloudConnectorDeleteCloudConnector](#), et selon [ListCloudConnectors](#) APIs les besoins pour cette procédure.

An AccountAssociation possède un état qui est demandé à partir [GetAccountAssociation](#) de et. [ListAccountAssociations](#) APIs Cela APIs montre l'état de l'association.

L'[StartAccountAssociationRefresh](#) API permet d'actualiser un AccountAssociation état lorsque son jeton d'actualisation expire.

4. Découverte des appareils

Chaque objet géré est lié à des informations spécifiques à l'appareil, telles que son numéro de série et un modèle de données. Le modèle de données décrit les fonctionnalités de l'appareil, indiquant s'il s'agit d'une ampoule, d'un interrupteur, d'un thermostat ou d'un autre type d'appareil. Pour découvrir un appareil 3P et créer un ManagedThing pour l'appareil 3P, vous devez suivre les étapes ci-dessous dans l'ordre.

- a. Appelez [StartDeviceDiscovery](#) API pour démarrer le processus de découverte des appareils.

Exemple de demande et de réponse à l' StartDeviceDiscovery API :

Request:

```
{
  "DiscoveryType": "CLOUD",
  "AccountAssociationId": "*****",
  "ClientToken": "abc"
}
```

Response:

```
{
  "Id": "string",
  "StartedAt": number
}
```

- b. Appelez [GetDeviceDiscovery](#) API pour vérifier l'état du processus de découverte.

- c. Appelez [ListDiscoveredDevices](#) l'API pour répertorier les appareils découverts.

Exemple de demande et de réponse à l' `ListDiscoveredDevices` API :

Request:

```
//Empty body
```

Response:

```
{
  "Items": [
    {
      "Brand": "string",
      "ConnectorDeviceId": "string",
      "ConnectorDeviceName": "string",
      "DeviceTypes": [ "string" ],
      "DiscoveredAt": number,
      "ManagedThingId": "string",
      "Model": "string",
      "Modification": "string"
    }
  ],
  "NextToken": "string"
}
```

- d. Appelez [CreateManagedThing](#) l'API pour sélectionner les appareils de la liste de découverte à importer dans les intégrations gérées.

Exemple de demande et de réponse à l' `CreateManagedThing` API :

Request:

```
{
  "Role": "DEVICE",
  "AuthenticationMaterial": "CLOUD:XXXX:<connectorDeviceId1>",
  "AuthenticationMaterialType": "DISCOVERED_DEVICE",
  "Name": "sample-device-name"
  "ClientToken": "xxx"
}
```

Response:

```
{
  "Arn": "string", // This is the ARN of the managedThing
  "CreatedAt": number,
  "Id": "string"
}
```

- e. Appelez [GetManagedThing](#) l'API pour afficher cette nouvelle création `managedThing`. Le statut sera `UNASSOCIATED`.
- f. Appelez [RegisterAccountAssociation](#) l'API pour l'associer `managedThing` à un élément spécifique `accountAssociation`. À la fin d'une [RegisterAccountAssociation](#) API réussie, l'état `managedThing` passe à `ASSOCIATED` l'état.

Exemple de demande et de réponse à l' `RegisterAccountAssociation` API :

Request:

```
{
  "AccountAssociationId": "string",
  "DeviceDiscoveryId": "string",
  "ManagedThingId": "string"
}
```

Response:

```
{
  "AccountAssociationId": "string",
  "DeviceDiscoveryId": "string",
  "ManagedThingId": "string"
}
```

5. Envoyer une commande à l'appareil 3P

Pour contrôler un appareil récemment intégré, utilisez l'[SendManagedThingCommand](#) API, avec l'ID d'association créé précédemment et une action de contrôle basée sur les fonctionnalités prises en charge par l'appareil. Le connecteur utilise les informations d'identification stockées lors du processus de liaison des comptes pour s'authentifier auprès du cloud tiers et appeler l'appel d'API correspondant à l'opération.

Exemple de demande et de réponse à l' `SendManagedThingCommand` API :

Request:

```
{
  "AccountAssociationId": "string",
  "ConnectorAssociationId": "string",
  "Endpoints": [
    {
      "capabilities": [
        {
          "actions": [
            {
              "actionTraceId": "string",
              "name": "string",
              "parameters": JSON value,
              "ref": "string"
            }
          ],
          "id": "string",
          "name": "string",
          "version": "string"
        }
      ],
      "endpointId": "string"
    }
  ]
}
```

Response:

```
{
  "TraceId": "string"
}
```

Envoyer la commande au flux de périphériques 3P :

6. Le connecteur envoie des événements aux intégrations gérées

L'[SendConnectorEvent](#) API capture quatre types d'événements, du connecteur aux intégrations gérées, représentés par les valeurs d'énumération suivantes pour le paramètre Operation Type :

- **DEVICE_COMMAND_RESPONSE** : réponse asynchrone envoyée par le connecteur en réponse à une commande.

- **DEVICE_DISCOVERY** : En réponse à un processus de découverte d'appareils, le connecteur envoie la liste des appareils découverts aux intégrations gérées, il utilise l'API. [SendConnectorEvent](#)
- **DEVICE_EVENT** : envoie les événements de l'appareil reçus.
- **DEVICE_COMMAND_REQUEST** : demandes de commande lancées depuis le périphérique. Par exemple, les flux de travail WebRTC.

Le connecteur peut également transmettre les événements de l'appareil à l'aide de l'[SendConnectorEvent](#) API, avec un `userId` paramètre facultatif.

- Pour les événements liés à un appareil avec `userId` :

Exemple de demande et de réponse à l' `SendConnectorEvent` API :

Request:

```
{
  "UserId": "*****",
  "Operation": "DEVICE_EVENT",
  "OperationVersion": "1.0",
  "StatusCode": 200,
  "ConnectorId": "*****",
  "ConnectorDeviceId": "****",
  "TraceId": "****",
  "MatterEndpoint": {
    "id": "***",
    "clusters": [{
      .....
    }]
  }
}
```

Response:

```
{
  "ConnectorId": "string"
}
```

- Pour les événements liés à un appareil sans `userId` :

Exemple de demande et de réponse à l' SendConnectorEvent API :

Request:

```
{
  "Operation": "DEVICE_EVENT",
  "OperationVersion": "1.0",
  "StatusCode": 200,
  "ConnectorId": "*****",
  "ConnectorDeviceId": "*****",
  "TraceId": "*****",
  "MatterEndpoint": {
    "id": "***",
    "clusters": [{
      ....
    }]
  }
}
```

Response:

```
{
  "ConnectorId": "string"
}
```

Pour supprimer le lien entre un compte en particulier managedThing et une association de comptes, utilisez le mécanisme de désenregistrement :

Exemple de demande et de réponse à l' DeregisterAccountAssociation API :

Request:

```
{
  "AccountAssociationId": "*****",
  "ManagedThingId": "*****"
}
```

Response:

HTTP/1.1 200 // Empty body

Envoyer le flux d'événements :

7. Mettez à jour le statut du connecteur sur « Listé » pour le rendre visible aux autres clients des intégrations gérées

Par défaut, les connecteurs sont privés et ne sont visibles que par le AWS compte qui les a créés. Vous pouvez choisir de rendre un connecteur visible pour les autres clients des intégrations gérées.

Pour partager votre connecteur avec d'autres utilisateurs, utilisez l'option Rendre visible AWS Management Console sur la page de détails du connecteur pour soumettre votre identifiant de connecteur à des AWS fins de révision. Une fois approuvé, le connecteur est accessible à tous les utilisateurs des intégrations gérées de la même manière Région AWS. En outre, vous pouvez restreindre l'accès à un AWS compte spécifique IDs en modifiant la politique d'accès relative à la AWS Lambda fonction associée au connecteur. Pour vous assurer que votre connecteur est utilisable par d'autres clients, gérez les autorisations d'accès IAM sur votre fonction Lambda depuis AWS d'autres comptes vers votre connecteur visible.

Passez en revue les Service AWS conditions et les politiques de votre organisation qui régissent le partage des connecteurs et les autorisations d'accès avant de rendre les connecteurs visibles aux autres clients des intégrations gérées.

Intégrations gérées (SDK) Hub

Consultez les rubriques de cette section pour apprendre à intégrer et à contrôler les appareils IoT Hub à l'aide du SDK Hub d'intégrations gérées. Pour plus d'informations sur le SDK pour terminaux d'intégrations gérées, consultez le. [Intégrations gérées SDK pour appareils finaux](#)

Architecture du SDK Hub

Intégration des appareils

Découvrez comment les composants du SDK Hub prennent en charge l'intégration des appareils avant de commencer à travailler avec des intégrations gérées. Cette section couvre les composants architecturaux essentiels dont vous avez besoin pour l'intégration des appareils, notamment la manière dont le fournisseur principal et les plugins spécifiques au protocole fonctionnent ensemble pour gérer l'authentification, la communication et la configuration utilisateur des appareils.

Composants du SDK Hub pour l'intégration des appareils

Composants du SDK

- [Provisionneur principal](#)
- [Plug-ins d'approvisionnement spécifiques au protocole](#)
- [Intergiciel spécifique au protocole](#)

Provisionneur principal

Le fournisseur principal est le composant central qui orchestre l'intégration des appareils dans le cadre du déploiement de votre hub IoT. Il coordonne toutes les communications entre les intégrations gérées et les plugins d'approvisionnement spécifiques à votre protocole, garantissant ainsi une intégration sécurisée et fiable des appareils. Lorsque vous embarquez un appareil, le fournisseur principal gère le flux d'authentification, gère la messagerie MQTT et traite les demandes de l'appareil par le biais des fonctions suivantes :

Connexion MQTT

Crée des connexions avec le courtier MQTT pour la publication et l'abonnement de rubriques dans le cloud.

File d'attente et gestionnaire de messages

Traite les demandes d'ajout et de suppression d'appareils entrantes dans l'ordre.

Interface du plug-in de protocole

Fonctionne avec des plug-ins d'approvisionnement spécifiques au protocole pour l'intégration des appareils en gérant les modes d'authentification et de connexion radio.

Client du SDK Hub APIs

Recevez et transférez des rapports sur les capacités des appareils à partir de plug-ins CDMB spécifiques au protocole vers des intégrations gérées.

Plug-ins d'approvisionnement spécifiques au protocole

Les plug-ins d'approvisionnement spécifiques au protocole sont des bibliothèques qui gèrent l'intégration des appareils pour différents protocoles de communication. Chaque plugin traduit les commandes du fournisseur principal en actions spécifiques au protocole pour vos appareils IoT. Ces plug-ins exécutent les tâches suivantes :

- Initialisation du middleware spécifique au protocole
- Configuration du mode de connexion radio basée sur les demandes du fournisseur principal
- Suppression de l'appareil via des appels d'API intergiciels

Intergiciel spécifique au protocole

Un intergiciel spécifique au protocole agit comme une couche de traduction entre les protocoles de votre appareil et les intégrations gérées. Ce composant traite les communications dans les deux sens : il reçoit les commandes des plug-ins du fournisseur et les envoie aux piles de protocoles, tout en collectant les réponses des appareils et en les acheminant vers le système.

Flux d'intégration des appareils

Passez en revue la séquence des opérations qui se produisent lorsque vous intégrez des appareils à l'aide du SDK Hub. Cette section montre comment les composants interagissent pendant le processus d'intégration et décrit les méthodes d'intégration prises en charge.

Flux d'intégration

- [Configuration simple \(SS\)](#)
- [Configuration sans contact \(ZTS\)](#)
- [Configuration guidée par l'utilisateur \(UGS\)](#)

Configuration simple (SS)

L'utilisateur final allume l'appareil IoT et scanne son code QR à l'aide de l'application du fabricant de l'appareil. L'appareil est ensuite inscrit dans le cloud d'intégrations gérées et se connecte au hub IoT.

Configuration sans contact (ZTS)

La configuration sans contact (ZTS) rationalise l'intégration des appareils en les pré-associant en amont dans la chaîne d'approvisionnement. Par exemple, au lieu que les utilisateurs finaux scannent le code QR de l'appareil, cette étape est effectuée plus tôt pour pré-associer les appareils aux comptes clients. Par exemple, cette étape peut être effectuée dans le centre de distribution.

Lorsque l'utilisateur final reçoit et allume l'appareil, celui-ci s'inscrit automatiquement dans le cloud d'intégrations gérées et se connecte au hub IoT sans nécessiter d'actions de configuration supplémentaires.

Configuration guidée par l'utilisateur (UGS)

L'utilisateur final allume l'appareil et suit des étapes interactives pour l'intégrer aux intégrations gérées. Cela peut inclure d'appuyer sur un bouton du hub IoT, d'utiliser une application du fabricant de l'appareil ou d'appuyer sur les boutons du hub et de l'appareil. Vous pouvez utiliser cette méthode en cas d'échec de la configuration simple.

Contrôle de l'appareil

Les intégrations gérées gèrent l'enregistrement des appareils, l'exécution des commandes et le contrôle. Vous pouvez créer des expériences pour les utilisateurs finaux sans connaître les protocoles spécifiques à l'appareil en utilisant sa gestion des appareils indépendante du fournisseur et du protocole.

Grâce au contrôle des appareils, vous pouvez afficher et modifier les états des appareils, tels que la luminosité de l'ampoule ou la position de la porte. La fonctionnalité émet des événements pour les changements d'état, que vous pouvez utiliser à des fins d'analyse, de règles et de surveillance.

Fonctions principales

Modifier ou lire l'état de l'appareil

Affichez et modifiez les attributs des appareils en fonction des types d'appareils. Vous pouvez accéder à :

- État de l'appareil : valeurs actuelles des attributs de l'appareil
- État de connectivité : état d'accessibilité de l'appareil
- État de santé : valeurs du système telles que le niveau de la batterie et l'intensité du signal (RSSI)

Notification de changement d'état

Recevez des événements lorsque les attributs ou les états de connectivité de l'appareil changent, tels que le réglage de la luminosité de l'ampoule ou le changement de l'état du verrouillage des portes.

Mode hors ligne

Les appareils communiquent avec d'autres appareils sur le même hub IoT, même sans connexion Internet. Les états des appareils sont synchronisés avec le cloud lorsque la connectivité est rétablie.

Synchronisation des états

Suivez les changements d'état provenant de plusieurs sources, des applications du fabricant de l'appareil et des réglages manuels de l'appareil.

Passez en revue les composants et les processus du SDK Hub dont vous avez besoin pour contrôler les appareils via des intégrations gérées. Cette rubrique décrit comment l'agent Edge, le Common

Data Model Bridge (CDBM) et les plug-ins spécifiques au protocole fonctionnent ensemble pour gérer les commandes des appareils, gérer les états des appareils et traiter les réponses via différents protocoles.

Flux de contrôle des appareils

Le schéma suivant illustre le flux de contrôle de l'end-to-end appareil en décrivant comment un utilisateur final active une prise intelligente Zigbee.

Composants du SDK Hub pour le contrôle des appareils

L'architecture du SDK Hub utilise les composants suivants pour traiter et acheminer les commandes de contrôle des appareils dans votre implémentation IoT. Chaque composant joue un rôle spécifique dans la traduction des commandes du cloud en actions de l'appareil, dans la gestion de l'état de l'appareil et dans le traitement des réponses. Les sections suivantes décrivent comment ces composants fonctionnent ensemble dans votre déploiement :

Le SDK Hub comprend les composants suivants et facilite l'intégration et le contrôle des appareils sur les hubs IoT.

Composants principaux :

Agent Edge

Agit comme une passerelle entre le hub IoT et les intégrations gérées.

Pont de modèle de données commun (CDBM)

Effectue des traductions entre le modèle de AWS données et les modèles de données du protocole local tels que Z-Wave et Zigbee. Il inclut un CDBM de base et des plug-ins CDBM spécifiques au protocole.

Approvisionnementneur

Gère la découverte et l'intégration des appareils. Il inclut un fournisseur principal et des plug-ins d'approvisionnement spécifiques au protocole pour les tâches d'intégration spécifiques au protocole.

Composants secondaires

Intégration au hub

Fournit au hub des certificats clients et des clés pour une communication sécurisée dans le cloud.

Proxy MQTT

Fournit des connexions MQTT au cloud d'intégrations gérées.

Logger

Écrit les journaux localement ou dans le cloud d'intégrations gérées.

Installation et validation du SDK Hub pour les intégrations gérées

Choisissez entre les méthodes de déploiement suivantes pour installer le SDK Managed Integrations Hub sur vos appareils, AWS IoT Greengrass pour un déploiement automatique ou une installation manuelle de scripts. Cette section décrit les étapes de configuration et de validation pour les deux approches.

Méthodes de déploiement

- [Installez le SDK du Hub avec AWS IoT Greengrass](#)
- [Déployez le SDK du Hub à l'aide d'un script](#)
- [Déployer le SDK Hub avec systemd](#)

Installez le SDK du Hub avec AWS IoT Greengrass

Déployez les composants du SDK du Hub d'intégrations gérées pour vos appareils à l'aide de AWS IoT Greengrass (version Java).

Note

Vous devez déjà avoir configuré et comprendre AWS IoT Greengrass. Pour plus d'informations, consultez la section [Contenu AWS IoT Greengrass](#) de la documentation du guide du AWS IoT Greengrass développeur.

L' AWS IoT Greengrass utilisateur doit être autorisé à modifier les répertoires suivants :

- /dev/aipc
- /data/aws/iotmi/config
- /data/ace/kvstorage

Rubriques

- [Déployer des composants localement](#)
- [déploiement dans le cloud](#)
- [Vérifier le provisionnement du hub](#)
- [Vérifier le fonctionnement du CDMB](#)
- [Vérifier le fonctionnement du LPW-Provisioner](#)

Déployer des composants localement

Utilisez l'[CreateDeployment](#) AWS IoT Greengrass API de votre appareil pour déployer les composants du SDK Hub. Les numéros de version ne sont pas statiques et peuvent varier en fonction de la version que vous utilisez à ce moment-là. Utilisez le format suivant pour **version** : com.Amazon.io TManagedIntegrationsDevice. AceCommon=0.2.0.

```
/greengrass/v2/bin/greengrass-cli deployment create \  
--recipeDir recipes \  
--artifactDir artifacts \  
-m "com.amazon.IoTManagedIntegrationsDevice.AceCommon=version" \  
-m "com.amazon.IoTManagedIntegrationsDevice.HubOnboarding=version" \  
-m "com.amazon.IoTManagedIntegrationsDevice.AceZigbee=version" \  
-m "com.amazon.IoTManagedIntegrationsDevice.LPW-Provisioner=version" \  
-m "com.amazon.IoTManagedIntegrationsDevice.Agent=version" \  
-m "com.amazon.IoTManagedIntegrationsDevice.MQTTProxy=version" \  
-m "com.amazon.IoTManagedIntegrationsDevice.CDMB=version" \  
-m "com.amazon.IoTManagedIntegrationsDevice.AceZwave=version"
```

déploiement dans le cloud

Suivez les instructions du [guide du AWS IoT Greengrass développeur](#) pour effectuer les étapes suivantes :

1. Téléchargez des artefacts sur Amazon S3.
2. Mettez à jour les recettes pour inclure l'emplacement de l'artefact Amazon S3.

3. Créez un déploiement dans le cloud sur l'appareil pour les nouveaux composants.

Vérifier le provisionnement du hub

Confirmez la réussite du provisionnement en vérifiant votre fichier de configuration. Ouvrez le `/data/aws/iotmi/config/iotmi_config.json` fichier et vérifiez que l'état est défini sur `PROVISIONED`.

Vérifier le fonctionnement du CDMB

Vérifiez dans le fichier journal les messages de démarrage CDMB et la réussite de l'initialisation. L'*logs file* emplacement peut varier en fonction de l'endroit où AWS IoT Greengrass il est installé.

```
tail -f -n 100 /greengrass/v2/logs/com.amazon.IoTManagedIntegrationsDevice.CDMB.log
```

exemple

```
[2024-09-06 02:31:54.413758906][IoTManagedIntegrationsDevice_CDMB][info] Successfully  
subscribed to topic: south/bF|gi_044F8821D0193608C8D5BF80858E20A56E3A8490/control  
[2024-09-06 02:31:54.513956059][IoTManagedIntegrationsDevice_CDMB][info] Successfully  
subscribed to topic: south/bF|gi_044F8821D0193608C8D5BF80858E20A56E3A8490/setup
```

Vérifier le fonctionnement du LPW-Provisioner

Vérifiez dans le fichier journal les messages de démarrage de LPW-Provisioner et la réussite de l'initialisation. L'*logs file* emplacement peut varier en fonction de l'endroit où AWS IoT Greengrass il est installé.

```
tail -f -n 100 /greengrass/v2/logs/com.amazon.IoTManagedIntegrationsDevice.LPW-  
Provisioner.log
```

exemple

```
[2024-09-06 02:33:22.068898877][LPWProvisionerCore][info] Successfully subscribed to  
topic: south/bF|gi_044F8821D0193608C8D5BF80858E20A56E3A8490/setup
```

Déployez le SDK du Hub à l'aide d'un script

Déployez les composants du SDK Hub d'intégrations gérées manuellement à l'aide de scripts d'installation, puis validez le déploiement. Cette section décrit les étapes d'exécution du script et le processus de vérification.

Rubriques

- [Préparez votre environnement](#)
- [Exécutez le script du SDK Hub](#)
- [Vérifier le provisionnement du hub](#)
- [Vérifier le fonctionnement de l'agent](#)
- [Vérifier le fonctionnement du LPW-Provisioner](#)

Préparez votre environnement

Effectuez les étapes suivantes avant d'exécuter le script d'installation du SDK :

1. Créez un dossier nommé `middleware` à l'intérieur du `artifacts` dossier.
2. Copiez les fichiers du `middleware` de votre hub dans le `middleware` dossier.
3. Exécutez les commandes d'initialisation avant de démarrer le SDK.

Important

Répétez les commandes d'initialisation après chaque redémarrage du hub.

```
#Get the current user
_user=$(whoami)

#Get the current group
_grp=$(id -gn)

#Display the user and group
echo "Current User: $_user"
echo "Current Group: $_grp"

sudo mkdir -p /dev/aipc/
sudo chown -R $_user:$_grp /dev/aipc
```

```
sudo mkdir -p /data/ace/kvstorage
sudo chown -R $_user:$_grp /data/ace/kvstorage
```

Exécutez le script du SDK Hub

Accédez au répertoire des artefacts et exécutez le `start_iotmi_sdk.sh` script. Ce script lance les composants du SDK du hub dans le bon ordre. Consultez les exemples de journaux suivants pour vérifier la réussite du démarrage :

Note

Les journaux de tous les composants en cours d'exécution se trouvent dans le `artifacts/logs` dossier.

```
hub@hub-293ea release_Oct_17$ ./start_iotmi_sdk.sh
-----Stopping SDK running processes---
DeviceAgent: no process found
-----Starting SDK-----
-----Creating logs directory-----
Logs directory created.
-----Verifying Middleware paths-----
All middleware libraries exist
-----Verifying Middleware pre reqs---
AIPC and KVstroage directories exist
-----Starting HubOnboarding-----
-----Starting MQTT Proxy-----
-----Starting Event Manager-----
-----Starting Zigbee Service-----
-----Starting Zwave Service-----
/data/release_Oct_17/middleware/AceZwave/bin /data/release_Oct_17
/data/release_Oct_17
-----Starting CDMB-----
-----Starting Agent-----
-----Starting Provisioner-----
-----Checking SDK status-----
hub          6199  1.7  0.7 1004952 15568 pts/2    Sl+  21:41   0:00 ./iotmi_mqtt_proxy -
C /data/aws/iotmi/config/iotmi_config.json
Process 'iotmi_mqtt_proxy' is running.
hub          6225  0.0  0.1 301576  2056 pts/2    Sl+  21:41   0:00 ./middleware/
AceCommon/bin/ace_eventmgr
```

```

Process 'ace_eventmgr' is running.
hub          6234  104  0.2 238560  5036 pts/2    Sl+  21:41   0:38 ./middleware/
AceZigbee/bin/ace_zigbee_service
Process 'ace_zigbee_service' is running.
hub          6242   0.4  0.7 1569372 14236 pts/2    Sl+  21:41   0:00 ./zwave_svc
Process 'zwave_svc' is running.
hub          6275   0.0  0.2 1212744  5380 pts/2    Sl+  21:41   0:00 ./DeviceCdm
Process 'DeviceCdm' is running.
hub          6308   0.6  0.9 1076108 18204 pts/2    Sl+  21:41   0:00 ./
IoTManagedIntegrationsDeviceAgent
Process 'DeviceAgent' is running.
hub          6343   0.7  0.7 1388132 13812 pts/2    Sl+  21:42   0:00 ./
iotmi_lpw_provisioner
Process 'iotmi_lpw_provisioner' is running.
-----Successfully Started SDK----
```

Vérifier le provisionnement du hub

Vérifiez que le `iot_provisioning_state` champ `/data/aws/iotmi/config/iotmi_config.json` est défini sur `PROVISIONED`.

Vérifier le fonctionnement de l'agent

Vérifiez dans le fichier journal les messages de démarrage de l'agent et la réussite de l'initialisation.

```
tail -f -n 100 logs/agent_logs.txt
```

exemple

```
[2024-09-06 02:31:54.413758906][Device_Agent][info] Successfully subscribed to topic:
south/bF|gi_044F8821D0193608C8D5BF80858E20A56E3A8490/control
[2024-09-06 02:31:54.513956059][Device_Agent][info] Successfully subscribed to topic:
south/bF|gi_044F8821D0193608C8D5BF80858E20A56E3A8490/setup
```

Note

Vérifiez que la `iotmi.db` base de données existe dans votre artifacts répertoire.

Vérifier le fonctionnement du LPW-Provisioner

Vérifiez dans le fichier journal les messages de LPW-Provisioner démarrage et la réussite de l'initialisation.

```
tail -f -n 100 logs/provisioner_logs.txt
```

Le code suivant en présente un exemple.

```
[2024-09-06 02:33:22.068898877][LPWProvisionerCore][info] Successfully subscribed to  
topic: south/bf|gi_044F8821D0193608C8D5BF80858E20A56E3A8490/setup
```

Déployer le SDK Hub avec systemd

Important

Suivez le `hubSystemdSetup` répertoire `readme.md` du fichier `release.tgz` pour obtenir les dernières mises à jour.

Cette section décrit les scripts et les processus de déploiement et de configuration des services sur un périphérique hub basé sur Linux.

Présentation

Le processus de déploiement comprend deux scripts principaux :

- `copy_to_hub.sh`: s'exécute sur la machine hôte pour copier les fichiers nécessaires sur le hub
- `setup_hub.sh`: s'exécute sur le hub pour configurer l'environnement et déployer les services

En outre, `systemd/deploy_iotshd_services_on_hub.sh` gère la séquence d'amorçage des processus et la gestion des autorisations de processus, et est automatiquement déclenché par `setup_hub.sh`.

Prérequis

Les conditions requises répertoriées sont requises pour un déploiement réussi.

- le service systemd est disponible sur le hub
- Accès SSH au périphérique hub
- Privilèges Sudo sur le périphérique du hub
- scputilitaire installé sur la machine hôte
- sedutilitaire installé sur la machine hôte
- utilitaire de décompression installé sur la machine hôte

Structure de fichier

La structure du fichier est conçue pour faciliter l'organisation et la gestion de ses différents composants, en permettant un accès et une navigation efficaces dans le contenu.

```
hubSystemdSetup/
### README.md
### copy_to_hub.sh
### setup_hub.sh
### iotshd_config.json # Sample configuration file
### local_certs/ # Directory for DHA certificates
### systemd/
    ### *.service.template # Systemd service templates
    ### deploy_iotshd_services_on_hub.sh
```

Dans le fichier tgz de la version SDK, la structure globale du fichier est la suivante :

```
IoT-managed-integrations-Hub-SDK-aarch64-v1.0.0.tgz
###package/
    ###greengrass/
        ###artifacts/
        ###recipes/
    ###hubSystemdSetup/
        ### REAME.md
        ### copy_to_hub.sh
        ### setup_hub.sh
        ### iotshd_config.json # Sample configuration file
        ### local_certs/ # Directory for DHA certificates
        ### systemd/
            ### *.service.template # Systemd service templates
            ### deploy_iotshd_services_on_hub.sh
```

Configuration initiale

Extraire le package du SDK

```
tar -xzf managed-integrations-Hub-SDK-vVersion-linux-aarch64-timestamp.tgz
```

Accédez au répertoire extrait et préparez le package :

```
# Create package.zip containing required artifacts
zip -r package.zip package/greengrass/artifacts
# Move package.zip to the hubSystemdSetup directory
mv package.zip ../hubSystemdSetup/
```

Ajouter des fichiers de configuration de l'appareil

Suivez les deux étapes répertoriées pour créer les fichiers de configuration de l'appareil et copiez-les dans le hub.

1. [Ajoutez des fichiers de configuration de périphérique](#) pour créer les fichiers de configuration de périphérique nécessaires. Le SDK utilise ce fichier pour sa fonction.
2. [Copiez les fichiers de configuration](#) pour copier les fichiers de configuration créés sur le hub.

Copier des fichiers sur le hub

Exécutez le script de déploiement depuis votre machine hôte :

```
chmod +x copy_to_hub.sh
./copy_to_hub.sh hub_ip_address package_file
```

Exemple exemple

```
./copy_to_hub.sh 192.168.50.223 ~/Downloads/EAR3-package.zip
```

Cela copie :

- Le fichier du package (renommé package.zip sur le hub)
- Fichiers de configuration
- Certificats
- fichiers de service Systemd

Configurer le hub

Une fois les fichiers copiés, connectez-vous au hub en SSH et exécutez le script de configuration :

```
ssh hub@hub_ip
chmod +x setup_hub.sh
sudo ./setup_hub.sh
```

Configurations des utilisateurs et des groupes

Par défaut, nous utilisons le hub utilisateur et le hub de groupe pour les composants du SDK. Il existe plusieurs manières de les configurer :

- Utilisez un utilisateur/un groupe personnalisé :

```
sudo ./setup_hub.sh --user=USERNAME --group=GROUPNAME
```

- Créez-les manuellement avant d'exécuter le script de configuration :

```
sudo groupadd -f GROUPNAME
sudo useradd -r -g GROUPNAME USERNAME
```

- Ajoutez les commandes `setup_hub.sh`.

Gérez les services

Pour redémarrer tous les services, exécutez le script suivant depuis le hub :

```
sudo /usr/local/bin/deploy_iotshd_services_on_hub.sh
```

Le script de configuration créera les répertoires nécessaires, définira les autorisations appropriées et déploiera les services automatiquement. Si vous n'utilisez pas SSH/SCP, vous devez le modifier en fonction de votre méthode de `copy_to_hub.sh` déploiement spécifique. Assurez-vous que tous les fichiers de certificats et toutes les configurations sont correctement configurés avant le déploiement.

Intégrez vos hubs aux intégrations gérées

Configurez les appareils de votre hub pour communiquer avec les intégrations gérées en configurant la structure de répertoire, les certificats et les fichiers de configuration des appareils requis. Cette section décrit comment les composants du sous-système d'intégration du hub fonctionnent ensemble,

où stocker les certificats et les fichiers de configuration, comment créer et modifier le fichier de configuration de l'appareil, et les étapes pour terminer le processus de provisionnement du hub.

Sous-système d'intégration du hub

Le sous-système d'intégration du hub utilise les composants principaux suivants pour gérer le provisionnement et la configuration des appareils :

Composant d'intégration du hub

Gère le processus d'intégration du hub en coordonnant l'état du hub, l'approche d'approvisionnement et le matériel d'authentification.

Fichier de configuration de l'appareil

Stocke les données de configuration essentielles du hub sur l'appareil, notamment :

- État de provisionnement de l'appareil (provisionné ou non provisionné)
- Certificat et emplacements clés
- Informations d'authentification D'autres processus du SDK, tels que le proxy MQTT, font référence à ce fichier pour déterminer l'état du hub et les paramètres de connexion.

Interface de gestion de certificats

Fournit une interface utilitaire pour lire et écrire les certificats et les clés des appareils. Vous pouvez implémenter cette interface pour qu'elle fonctionne avec :

- Stockage du système de fichiers
- Modules de sécurité matériels (HSM)
- Modules de plateforme fiables (TPM)
- Solutions de stockage sécurisées personnalisées

composant proxy MQTT

Gère device-to-cloud la communication en utilisant :

- Certificats et clés clients provisionnés
- Informations sur l'état de l'appareil provenant du fichier de configuration
- Connexions MQTT aux intégrations gérées

Le schéma suivant décrit l'architecture du sous-système d'intégration du hub et ses composants. Si vous ne l'utilisez pas AWS IoT Greengrass, vous pouvez ignorer cette composante du diagramme.

Configuration de l'intégration du hub

Effectuez ces étapes de configuration pour chaque appareil du hub avant de commencer le processus d'intégration du provisionnement de la flotte. Cette section décrit comment créer des objets gérés, configurer des structures de répertoires et configurer les certificats requis.

Étapes de configuration

- [Étape 1 : enregistrer un point de terminaison personnalisé](#)
- [Étape 2 : Création d'un profil d'approvisionnement](#)
- [Étape 3 : Création d'un objet géré \(approvisionnement de flotte\)](#)
- [Étape 4 : Création de la structure du répertoire](#)
- [Étape 5 : ajouter du matériel d'authentification à l'appareil du hub](#)
- [Étape 6 : Création du fichier de configuration de l'appareil](#)
- [Étape 7 : Copiez le fichier de configuration sur votre hub](#)

Étape 1 : enregistrer un point de terminaison personnalisé

Créez un point de communication dédié que vos appareils utiliseront pour échanger des données avec des intégrations gérées. Ce point de terminaison établit un point de connexion sécurisé pour tous les device-to-cloud messages, y compris les commandes de l'appareil, les mises à jour de statut et les notifications.

Pour enregistrer un point de terminaison

- Utilisez l'[RegisterCustomEndpoint](#) API pour créer un point de terminaison pour la communication device-to-managed des intégrations.

RegisterCustomEndpointExemple de demande

```
aws iot-managed-integrations register-custom-endpoint
```

Réponse :

```
{
```

```
[ACCOUNT-PREFIX]-ats.iot.AWS-REGION.amazonaws.com  
}
```

Note

Stockez l'adresse du point de terminaison. Vous en aurez besoin pour les futures communications entre appareils.

Pour renvoyer les informations du point de terminaison, utilisez l'`GetCustomEndpointAPI`.

Pour plus d'informations, consultez l'[RegisterCustomEndpointAPI](#) et l'[GetCustomEndpointAPI](#) dans le Guide de référence de l'API des intégrations gérées.

Étape 2 : Création d'un profil d'approvisionnement

Un profil d'approvisionnement contient les informations d'identification de sécurité et les paramètres de configuration dont vos appareils ont besoin pour se connecter aux intégrations gérées.

Pour créer un profil d'approvisionnement de flotte

- Appelez l'[CreateProvisioningProfileAPI](#) pour générer ce qui suit :
 - Un modèle de provisionnement qui définit les paramètres de connexion des appareils
 - Un certificat de réclamation et une clé privée pour l'authentification de l'appareil

Important

Stockez le certificat de réclamation, la clé privée et l'identifiant du modèle en toute sécurité. Vous aurez besoin de ces informations d'identification pour intégrer les appareils aux intégrations gérées. Si vous perdez ces informations d'identification, vous devez créer un nouveau profil d'approvisionnement.

CreateProvisioningProfile exemple de demande

```
aws iot-managed-integrations create-provisioning-profile \  
--provisioning-type FLEET_PROVISIONING \  

```

```
--name PROFILE_NAME
```

Réponse :

```
{
  "Arn": "arn:aws:iotmanagedintegrations:AWS-REGION:ACCOUNT-ID:provisioning-
profile/PROFILE-ID",
  "ClaimCertificate":
    "-----BEGIN CERTIFICATE-----
MIICiTCCAfICCQD6m7....w3rrszlaEXAMPLE=
-----END CERTIFICATE-----",
  "ClaimCertificatePrivateKey":
    "-----BEGIN RSA PRIVATE KEY-----
MIICiTCCAfICCQ...3rrszlaEXAMPLE=
-----END RSA PRIVATE KEY-----",
  "Id": "PROFILE-ID",
  "PROFILE-NAME",
  "ProvisioningType": "FLEET_PROVISIONING"
}
```

Étape 3 : Création d'un objet géré (approvisionnement de flotte)

Utilisez l'`CreateManagedThingAPI` pour créer un objet géré pour votre appareil hub. Chaque hub a besoin de son propre équipement géré avec des matériaux d'authentification uniques. Pour plus d'informations, consultez l'[CreateManagedThingAPI](#) dans le manuel de référence des API d'intégrations gérées.

Lorsque vous créez un objet géré, spécifiez les paramètres suivants :

- **Role**: Définissez cette valeur sur `CONTROLLER` pour les hubs qui ne prennent pas en charge le commandement et le contrôle, sinon définissez cette valeur sur `DEVICE`.
- **AuthenticationMaterialType**: Définissez cette valeur sur `WIFI_SETUP_QR_BAR_CODE`.
- **AuthenticationMaterial**: Incluez les champs suivants. Vous pouvez utiliser l'un UPC ou l'autre EAN, mais pas les deux.
 - **SN**: le numéro de série unique de cet appareil
 - **UPC**: le code produit universel pour cet appareil
 - **EAN**: le numéro d'article international de cet appareil

⚠ Important

Chaque appareil doit avoir un numéro de série (SN) unique dans son matériel d'authentification.

CreateManagedThing Exemple de demande :

```
{
  "Role": "CONTROLLER",
  "AuthenticationMaterialType": "WIFI_SETUP_QR_BAR_CODE",
  "AuthenticationMaterial": "SN:123456789524;UPC:829576019524"
}
```

Pour plus d'informations, consultez la référence [CreateManagedThing](#) de l'API des intégrations gérées.

(Facultatif) Obtenez un objet géré

ProvisioningStatus L'objet que vous gérez doit l'être PRE_ASSOCIATED avant de pouvoir continuer. Pour plus d'informations **ProvisioningStatus**, consultez la section [Approvisionnement des appareils](#). Utilisez l'GetManagedThingAPI pour vérifier que votre objet géré existe et qu'il est prêt à être provisionné. Pour plus d'informations, consultez la référence [GetManagedThing](#) de l'API des intégrations gérées.

Étape 4 : Création de la structure du répertoire

Créez des répertoires pour vos fichiers de configuration et vos certificats. Par défaut, le processus d'intégration du hub utilise le /data/aws/iotmi/config/iotmi_config.json.

Vous pouvez spécifier des chemins personnalisés pour les certificats et les clés privées dans le fichier de configuration. Ce guide utilise le chemin par défaut /data/aws/iotmi/certs.

```
mkdir -p /data/aws/iotmi/config
mkdir -p /data/aws/iotmi/certs
```

```
/data/
  aws/
    iotmi/
      config/
      certs/
```

Étape 5 : ajouter du matériel d'authentification à l'appareil du hub

Copiez les certificats et les clés sur votre périphérique hub, puis créez un fichier de configuration spécifique à l'appareil. Ces fichiers établissent une communication sécurisée entre votre hub et les intégrations gérées pendant le processus de provisionnement.

Pour copier le certificat de réclamation et la clé

- Copiez ces fichiers d'authentification depuis votre réponse `CreateProvisioningProfile` API vers votre périphérique hub :
 - `claim_cert.pem`: le certificat de réclamation (commun à tous les appareils)
 - `claim_pk.key`: clé privée pour le certificat de réclamation

Placez les deux fichiers dans le `/data/aws/iotmi/certs` répertoire.

Important

Lorsque vous stockez des certificats et des clés privées au format PEM, assurez-vous que le formatage est correct en gérant correctement les caractères de nouvelle ligne. Pour les fichiers codés au format PEM, les caractères de nouvelle ligne (`\n`) doivent être remplacés par de véritables séparateurs de ligne, car le simple fait de stocker les nouvelles lignes échappées ne sera pas correctement récupéré ultérieurement.

Note

Si vous utilisez le stockage sécurisé, stockez ces informations d'identification dans votre emplacement de stockage sécurisé plutôt que dans le système de fichiers. Pour de plus amples informations, veuillez consulter [Créez un gestionnaire de certificats personnalisé pour un stockage sécurisé](#).

Étape 6 : Création du fichier de configuration de l'appareil

Créez un fichier de configuration contenant les identifiants uniques des appareils, les emplacements des certificats et les paramètres de provisionnement. Le SDK utilise ce fichier lors de l'intégration du

hub pour authentifier votre appareil, gérer l'état du provisionnement et enregistrer les paramètres de connexion.

 Note

Chaque périphérique hub nécessite son propre fichier de configuration avec des valeurs uniques spécifiques au périphérique.

Utilisez la procédure suivante pour créer ou modifier votre fichier de configuration, puis copiez-le dans le hub.

- Créez ou modifiez le fichier de configuration (approvisionnement du parc).

Configurez les champs obligatoires suivants dans le fichier de configuration de l'appareil :

- Chemins de certification
 1. `iot_claim_cert_path`: Emplacement de votre certificat de réclamation (`claim_cert.pem`)
 2. `iot_claim_pk_path`: Emplacement de votre clé privée (`claim_pk.key`)
 3. `SECURE_STORAGE` à utiliser pour les deux champs lors de la mise en œuvre du gestionnaire de certificats de stockage sécurisé
- Réglages de connexion
 1. `fp_template_name`: Le ProvisioningProfile nom d'avant.
 2. `endpoint_url`: URL du point de terminaison de vos intégrations gérées provenant de la réponse de l'`RegisterCustomEndpointAPI` (identique pour tous les appareils d'une région).
- Identifiants de l'appareil
 1. `SN`: numéro de série de l'appareil correspondant à votre appel d'`CreateManagedThingAPI` (unique par appareil)
 2. `UPCCode` produit universel issu de votre appel d'`CreateManagedThingAPI` (identique pour tous les appareils de ce produit)

```
{  
  "ro": {  
    "iot_provisioning_method": "FLEET_PROVISIONING",
```

```

    "iot_claim_cert_path": "<SPECIFY_THIS_FIELD>",
    "iot_claim_pk_path": "<SPECIFY_THIS_FIELD>",
    "fp_template_name": "<SPECIFY_THIS_FIELD>",
    "endpoint_url": "<SPECIFY_THIS_FIELD>",
    "SN": "<SPECIFY_THIS_FIELD>",
    "UPC": "<SPECIFY_THIS_FIELD>"
  },
  "rw": {
    "iot_provisioning_state": "NOT_PROVISIONED"
  }
}

```

Contenu du fichier de configuration

Vérifiez le contenu du `iotmi_config.json` fichier.

Table des matières

Clé	Valeurs	Ajouté par le client ?	Remarques
<code>iot_provisioning_method</code>	FLEET_PROVISIONING	Oui	Spécifiez la méthode de provisionnement que vous souhaitez utiliser.
<code>iot_claim_cert_path</code>	Le chemin du fichier que vous spécifiez ou <code>SECURE_STORAGE</code> . Par exemple, <code>/data/aws/iotmi/certs/claim_cert.pem</code>	Oui	Spécifiez le chemin du fichier que vous souhaitez utiliser ou <code>SECURE_STORAGE</code> .
<code>iot_claim_pk_path</code>	Le chemin du fichier que vous spécifiez ou <code>SECURE_STORAGE</code> . Par exemple, <code>/data/aws/iotmi/certs/claim_pk.pem</code>	Oui	Spécifiez le chemin du fichier que vous souhaitez utiliser ou <code>SECURE_STORAGE</code> .

Clé	Valeurs	Ajouté par le client ?	Remarques
fp_template_name	Le nom du modèle de provisionnement de flotte doit être identique au nom du ProvisioningProfile modèle utilisé précédemment.	Oui	Égal au nom du ProvisioningProfile qui a été utilisé précédemment
endpoint_url	URL du point de terminaison pour les intégrations gérées.	Oui	Vos appareils utilisent cette URL pour se connecter au cloud d'intégrations gérées. Pour obtenir ces informations, utilisez l' RegisterCustomEndpointAPI .
SN	Le numéro de série de l'appareil. Par exemple, AIDACKCEVSQ6C2EXAMPLE .	Oui	Vous devez fournir ces informations uniques pour chaque appareil.
UPC	Code produit universel de l'appareil. Par exemple, 841667145075 .	Oui	Vous devez fournir ces informations pour l'appareil.
managed_thing_id	L'identifiant de l'objet géré.	Non	Ces informations sont ajoutées ultérieurement par le processus d'intégration après le provisionnement du hub.
iot_provisioning_state	État du provisionnement.	Oui	L'état de provisionnement doit être défini comme NOT_PROVISIONED .
iot_permanent_cert_path	Le chemin du certificat IoT. Par exemple, /data/aws/iotmi/iot_cert.pem .	Non	Ces informations sont ajoutées ultérieurement par le processus d'intégration après le provisionnement du hub.

Clé	Valeurs	Ajouté par le client ?	Remarques
<code>iot_permanent_pk_path</code>	Le chemin du fichier de clé privée de l'IoT. Par exemple, <code>/data/aws/iotmi/iot_pk.pem</code> .	Non	Ces informations sont ajoutées ultérieurement par le processus d'intégration après le provisionnement du hub.
<code>client_id</code>	L'ID client qui sera utilisé pour les connexions MQTT.	Non	Ces informations sont ajoutées ultérieurement par le processus d'intégration après le provisionnement du hub, pour que d'autres composants puissent être consommés.
<code>mqtt_keep_alive_interval</code>	La plage est comprise entre 30 et 1200, et les unités sont exprimées en secondes. La valeur par défaut est 300.	Oui	Utilisez-le pour définir un intervalle de conservation pour les connexions MQTT.
<code>event_manager_upper_bound</code>	La valeur par défaut est 500.	Non	Ces informations sont ajoutées ultérieurement par le processus d'intégration après le provisionnement du hub, pour que d'autres composants puissent être consommés.

Étape 7 : Copiez le fichier de configuration sur votre hub

Copiez votre fichier de configuration `/data/aws/iotmi/config` ou le chemin de votre répertoire personnalisé. Vous fournirez ce chemin vers le HubOnboarding binaire lors du processus d'intégration.

Pour le provisionnement de flottes

```
/data/
aws/
```

```
iotmi/  
  config/  
    iotmi_config.json  
  certs/  
    claim_cert.pem  
    claim_pk.key
```

Intégrez les appareils et faites-les fonctionner dans le hub

Configurez vos appareils pour qu'ils soient intégrés à votre hub d'intégrations gérées en créant un objet géré et en le connectant à votre hub. Les appareils peuvent être intégrés à un hub via une configuration simple ou une configuration guidée par l'utilisateur.

Rubriques

- [Configuration simple pour embarquer et faire fonctionner les appareils](#)
- [Configuration guidée par l'utilisateur pour embarquer et faire fonctionner les appareils](#)

Configuration simple pour embarquer et faire fonctionner les appareils

Configurez vos appareils pour qu'ils soient intégrés à votre hub d'intégrations gérées en créant un objet géré et en le connectant à votre hub. Cette section décrit les étapes à suivre pour terminer le processus d'intégration de l'appareil à l'aide d'une configuration simple.

Conditions préalables

Effectuez les étapes suivantes avant de tenter d'intégrer un appareil :

- Intégrez un appareil central au hub d'intégrations gérées.
- Installez la dernière version de AWS CLI depuis le manuel [Managed Integrations AWS CLI Command Reference](#)
- Abonnez-vous aux notifications d'événements [DEVICE_LIFE_CYCLE](#).

Étapes de configuration

- [Étape 1 : créer un casier d'identifiants](#)
- [Étape 2 : ajoutez le casier d'identifiants à votre hub](#)
- [Étape 3 : Créez un objet géré avec des informations d'identification.](#)

- [Étape 4 : Branchez l'appareil et vérifiez son état.](#)
- [Étape 5 : Obtenir les fonctionnalités de l'appareil](#)
- [Étape 6 : Envoyer une commande à l'objet géré](#)
- [Étape 7 : supprimer l'objet géré de votre hub](#)

Étape 1 : créer un casier d'identifiants

Créez un casier d'informations d'identification pour votre appareil.

Pour créer un casier d'informations d'identification

- Utilisez la commande `create-credential-locker`. L'exécution de cette commande déclenchera la création de toutes les ressources de fabrication, y compris la paire de clés de configuration Wi-Fi et le certificat de l'appareil.

Exemple `create-credential-locker`

```
aws iot-managed-integrations create-credential-locker \
  --name "DEVICE_NAME"
```

Réponse :

```
{
  "Id": "LOCKER_ID"
  "Arn": "arn:aws:iotmanagedintegrations:AWS_REGION:AWS_ACCOUNT_ID:credential-
  locker/LOCKER_ID"
  "CreatedAt": "2025-06-09T13:58:52.977000+08:00"
}
```

Pour plus d'informations, consultez la [create-credential-locker](#) commande dans le manuel de référence des AWS CLI commandes des intégrations gérées.

Étape 2 : ajoutez le casier d'identifiants à votre hub

Ajoutez le casier d'identifiants à votre hub.

Pour ajouter un casier d'identifiants à votre hub

- Utilisez la commande suivante pour ajouter un casier d'informations d'identification à votre hub.

```
aws iotmi --region AWS_REGION --endpoint AWS_ENDPOINT update-managed-thing \  
--identifier "HUB_MANAGED_THING_ID" --credential-locker-id "LOCKER_ID"
```

Étape 3 : Créez un objet géré avec des informations d'identification.

Créez un objet géré avec les informations d'identification de votre appareil. Chaque appareil doit être géré par lui-même.

Pour créer un objet géré

- Utilisez la `create-managed-thing` commande pour créer un objet géré pour votre appareil.

Exemple create-managed-thing

```
#ZWAVE:  
aws iot-managed-integrations create-managed-thing --role DEVICE \  
--authentication-material '900137947003133...' \ #auth material from zwave qr code  
--authentication-material-type ZWAVE_QR_BAR_CODE \  
--credential-locker-id ${locker_id}  
  
#ZIGBEE:  
aws iot-managed-integrations create-managed-thing --role DEVICE \  
--authentication-material 'Z:286...$I:A4DC00.' \ #auth material from zigbee qr code  
--authentication-material-type ZIGBEE_QR_BAR_CODE \  
--credential-locker-id ${locker_id}
```

Note

Il existe des commandes distinctes pour les appareils Z-wave et Zigbee.

Réponse :

```
{  
  "Id": "DEVICE_MANAGED_THING_ID"  
  "Arn": "arn:aws:iotmanagedintegrations:AWS_REGION:AWS_ACCOUNT_ID:managed-  
thing/DEVICE_MANAGED_THING_ID"  
  "CreatedAt": "2025-06-09T13:58:52.977000+08:00"  
}
```

Pour plus d'informations, consultez la [create-managed-thing](#) commande dans le manuel de référence des AWS CLI commandes des intégrations gérées.

Étape 4 : Branchez l'appareil et vérifiez son état.

Branchez l'appareil et vérifiez son état.

- Utilisez la `get-managed-thing` commande pour vérifier l'état de votre appareil. `ProvisioningStatus` L'objet que vous gérez doit être `ACTIVÉ`. Pour plus d'informations `ProvisioningStatus`, consultez la section [Approvisionnement des appareils](#).

Exemple `get-managed-thing`

```
#KINESIS NOTIFICATION:
{
  "version": "1.0.0",
  "messageId": "4ac684bb7f4c41adbb2eccc1e7991xxx",
  "messageType": "DEVICE_LIFE_CYCLE",
  "source": "aws.iotmanagedintegrations",
  "customerAccountId": "12345678901",
  "timestamp": "2025-06-10T05:30:59.852659650Z",
  "region": "us-east-1",
  "resources": ["XXX"],
  "payload": {
    "deviceDetails": {
      "id": "1e84f61fa79a41219534b6fd57052XXX",
      "arn": "XXX",
      "createdAt": "2025-06-09T06:24:34.336120179Z",
      "updatedAt": "2025-06-10T05:30:59.784157019Z"
    },
    "status": "ACTIVATED"
  }
}
aws iot-managed-integrations get-managed-thing \
--identifier :"DEVICE_MANAGED_THING_ID"
```

Réponse :

```
{
  "Id": :"DEVICE_MANAGED_THING_ID"
```

```
"Arn": "arn:aws:iotmanagedintegrations:AWS_REGION:AWS_ACCOUNT_ID:managed-thing/MANAGED_THING_ID"
"CreatedAt": "2025-06-09T13:58:52.977000+08:00"
}
```

Pour plus d'informations, consultez la [get-managed-thing](#) commande dans le manuel de référence des AWS CLI commandes des intégrations gérées.

Étape 5 : Obtenir les fonctionnalités de l'appareil

Utilisez la `get-managed-thing-capabilities` commande pour obtenir l'ID de votre terminal et afficher la liste des actions possibles pour votre appareil.

Pour obtenir les fonctionnalités d'un appareil

- Utilisez la `get-managed-thing-capabilities` commande et notez l'ID du point de terminaison.

Exemple `get-managed-thing-capabilities`

```
aws iotmi get-managed-thing-capabilities \
--identifier "DEVICE_MANAGED_THING_ID"
```

Réponse :

```
{
  "ManagedThingId": "1e84f61fa79a41219534b6fd57052cbc",
  "CapabilityReport": {
    "version": "1.0.0",
    "nodeId": "zw.FCB10009+06",
    "endpoints": [
      {
        "id": "ENDPOINT_ID"
        "deviceTypes": [
          "On/Off Switch"
        ],
        "capabilities": [
          {
            "id": "matter.OnOff@1.4",
            "name": "On/Off",
            "version": "6",
```

```

        "properties": [
            "OnOff"
        ],
        "actions": [
            "Off",
            "On"
        ],
        "events": []
    }
    ...
}

```

Pour plus d'informations, consultez la [get-managed-thing-capabilities](#) commande dans le manuel de référence des AWS CLI commandes des intégrations gérées.

Étape 6 : Envoyer une commande à l'objet géré

Utilisez la `send-managed-thing-command` commande pour envoyer une commande d'action à bascule à votre objet géré.

Pour envoyer une commande à votre objet géré

- Utilisez la `send-managed-thing-command` commande pour envoyer une commande à votre objet géré.

Exemple `send-managed-thing-command`

```

json=$(jq -cr '.|@json') <<EOF
[
  {
    "endpointId": "1",
    "capabilities": [
      {
        "id": "matter.OnOff@1.4",
        "name": "On/Off",
        "version": "1",
        "actions": [
          {
            "name": "Toggle",
            "parameters": {}
          }
        ]
      }
    ]
  }
]

```



```
}  
]  
}  
]  
EOF  
aws iot-managed-integrations send-managed-thing-command \  
--managed-thing-id "DEVICE_MANAGED_THING_ID" --endpoints "ENDPOINT_ID"
```

Note

Cet exemple utilise jq cli to, mais vous pouvez également transmettre la chaîne entière endpointId.

Réponse :

```
{  
  "TraceId": "TRACE_ID"  
}
```

Pour plus d'informations, consultez la [send-managed-thing-command](#) commande dans le manuel de référence des AWS CLI commandes des intégrations gérées.

Étape 7 : supprimer l'objet géré de votre hub

Nettoyez votre hub en supprimant l'élément géré.

Pour supprimer un objet géré

- Utilisez la `delete-managed-thing` commande pour supprimer un objet géré du hub de votre appareil.

Exemple `delete-managed-thing`

```
aws iot-managed-integrations delete-managed-thing \  
--identifier "DEVICE_MANAGED_THING_ID"
```

Pour plus d'informations, consultez la [delete-managed-thing](#) commande dans le manuel de référence des AWS CLI commandes des intégrations gérées.

 Note

Si l'appareil est bloqué dans un DELETE_IN_PROGRESS état, ajoutez le `--force` drapeau à `delete-managed-thing` command.

 Note

Pour les appareils Z-wave, vous devez mettre l'appareil en mode couplage après avoir exécuté la commande.

Configuration guidée par l'utilisateur pour embarquer et faire fonctionner les appareils

Configurez vos appareils pour qu'ils soient intégrés à votre hub d'intégrations gérées en créant un objet géré et en le connectant à votre hub. Cette section décrit les étapes à suivre pour terminer le processus d'intégration de l'appareil à l'aide de la configuration guidée par l'utilisateur.

Conditions préalables

Effectuez les étapes suivantes avant de tenter d'intégrer un appareil :

- Intégrez un appareil central au hub d'intégrations gérées.
- Installez la dernière version de AWS CLI depuis le manuel [Managed Integrations AWS CLI Command Reference](#)
- Abonnez-vous aux notifications d'événements [DEVICE_DISCOVERY-STATUS](#).

Étapes de configuration guidées par l'utilisateur

- [Prérequis : activer le mode de jumelage sur votre appareil Z Wave](#)
- [Étape 1 : démarrer la découverte des appareils](#)
- [Étape 2 : demander l'ID de la tâche de découverte](#)

- [Étape 3 : créer un objet géré pour votre appareil](#)
- [Étape 4 : interroger l'objet géré](#)
- [Étape 5 : Bénéficiez de fonctionnalités de gestion des objets](#)
- [Étape 6 : Envoyer une commande à l'objet géré](#)
- [Étape 7 : vérifier l'état de l'objet géré](#)
- [Étape 8 : Supprimer l'objet géré de votre hub](#)

Prérequis : activer le mode de jumelage sur votre appareil Z Wave

Activez le mode de couplage sur l'appareil Z-wave. Le mode de couplage peut varier pour chaque appareil Z-Wave. Reportez-vous donc aux instructions de l'appareil pour configurer correctement le mode de couplage. Il s'agit généralement d'un bouton sur lequel l'utilisateur doit appuyer.

Étape 1 : démarrer la découverte des appareils

Lancez la découverte des appareils pour votre hub afin d'obtenir un identifiant de tâche de découverte qui sera utilisé pour intégrer votre appareil.

Pour démarrer la découverte des appareils

- Utilisez la [start-device-discovery](#) commande pour obtenir l'ID de la tâche de découverte.

Exemple start-device-discovery

```
#For Zigbee
aws iot-managed-integrations start-device-discovery --discovery-type ZIGBEE \
--controller-identifier HUB_MANAGED_THING_ID

#For Zwave
aws iot-managed-integrations start-device-discovery --discovery-type ZWAVE \
--controller-identifier HUB_MANAGED_THING \
--authentication-material-type ZWAVE_INSTALL_CODE \
--authentication-material 13333

#For Cloud
aws iot-managed-integrations start-device-discovery --discovery-type CLOUD \
--account-association-id C2C_ASSOCIATION_ID \

#For Custom
aws iot-managed-thing start-device-discovery --discovery-type CUSTOM \
```

```
--controller-identifier HUB_MANAGED_THING_ID \  
--custom-protocol-detail NAME : NON_EMPTY_STRING \
```

Réponse :

```
{  
  "Id": DISCOVERY_JOB_ID,  
  "StartedAt": "2025-06-03T14:43:12.726000-07:00"  
}
```

 Note

Il existe des commandes distinctes pour les appareils Z-wave et Zigbee.

Pour plus d'informations, consultez l'[start-device-discovery](#) API dans le manuel de référence des AWS CLI commandes d'intégrations gérées.

Étape 2 : demander l'ID de la tâche de découverte

Utilisez la `list-discovered-devices` commande pour obtenir le matériel d'authentification de votre appareil.

Pour demander votre identifiant de tâche de découverte

- Utilisez l'ID de la tâche de découverte avec la `list-discovered-devices` commande pour obtenir le matériel d'authentification de votre appareil.

```
aws iot-managed-integrations list-discovered-devices --identifier DISCOVERY_JOB_ID
```

Réponse :

```
"Items": [  
  {  
    "DeviceTypes": [],  
    "DiscoveredAt": "2025-06-03T14:43:37.619000-07:00",  
    "AuthenticationMaterial": AUTHENTICATION_MATERIAL  
  }  
]
```

]

Étape 3 : créer un objet géré pour votre appareil

Utilisez la `create-managed-thing` commande pour créer un objet géré pour votre appareil. Chaque appareil doit être géré par lui-même.

Pour créer un objet géré

- Utilisez la `create-managed-thing` commande pour créer un objet géré pour votre appareil.

Exemple `create-managed-thing`

```
aws iot-managed-integrations create-managed-thing \  
  --role DEVICE --authentication-material-type DISCOVERED_DEVICE \  
  --authentication-material "AUTHENTICATION_MATERIAL"
```

Réponse :

```
{  
  "Id": "DEVICE_MANAGED_THING_ID"  
  "Arn": "arn:aws:iotmanagedintegrations:AWS_REGION:AWS_ACCOUNT_ID:managed-  
thing/DEVICE_MANAGED_THING_ID"  
  "CreatedAt": "2025-06-09T13:58:52.977000+08:00"  
}
```

Pour plus d'informations, consultez la [create-managed-thing](#) commande dans le manuel de référence des AWS CLI commandes des intégrations gérées.

Étape 4 : interroger l'objet géré

Vous pouvez vérifier si un objet géré est activé à l'aide de la `get-managed-thing` commande.

Pour interroger un objet géré

- Utilisez la `get-managed-thing` commande pour vérifier si le statut de provisionnement de l'objet géré est défini sur `ACTIVATED`. Pour plus d'informations sur l'état du provisionnement, consultez la section [Appvisionnement des appareils](#).

Exemple `get-managed-thing`

```
aws iot-managed-integrations get-managed-thing \  
  --identifiant "DEVICE_MANAGED_THING_ID"
```

Réponse :

```
{  
  "Id": "DEVICE_MANAGED_THING_ID",  
  "Arn": "arn:aws:iotmanagedintegrations:AWS_REGION:AWS_ACCOUNT_ID:managed-  
thing/DEVICE_MANAGED_THING_ID,  
  "Role": "DEVICE",  
  "ProvisioningStatus": "ACTIVATED",  
  "MacAddress": "MAC_ADDRESS",  
  "ParentControllerId": "PARENT_CONTROLLER_ID",  
  "CreatedAt": "2025-06-03T14:46:35.149000-07:00",  
  "UpdatedAt": "2025-06-03T14:46:37.500000-07:00",  
  "Tags": {}  
}
```

Pour plus d'informations, consultez la [get-managed-thing](#) commande dans le manuel de référence des AWS CLI commandes des intégrations gérées.

Étape 5 : Bénéficiez de fonctionnalités de gestion des objets

Vous pouvez consulter la liste des actions disponibles pour un objet géré en utilisant le `get-managed-thing-capabilities`.

Pour obtenir les fonctionnalités d'un appareil

- Utilisez la `get-managed-thing-capabilities` commande pour obtenir l'ID du point de terminaison. Notez également la liste des actions possibles.

Exemple `get-managed-thing-capabilities`

```
aws iot-managed-integrations get-managed-thing-capabilities \  
  --identifiant "DEVICE_MANAGED_THING_ID"
```

Réponse :

```
{
```

```
"ManagedThingId": "DEVICE_MANAGED_THING_ID",
"CapabilityReport": {
  "version": "1.0.0",
  "nodeId": "zb.539D+4A1D",
  "endpoints": [
    {
      "id": "1",
      "deviceTypes": [
        "Unknown Device"
      ],
      "capabilities": [
        {
          "id": "matter.OnOff@1.4",
          "name": "On/Off",
          "version": "6",
          "properties": [
            "OnOff",
            "OnOff",
            "OnTime",
            "OffWaitTime"
          ],
          "actions": [
            "Off",
            "On",
            "Toggle",
            "OffWithEffect",
            "OnWithRecallGlobalScene",
            "OnWithTimedOff"
          ],
          ...
        }
      ]
    }
  ]
}
```

Pour plus d'informations, consultez la [get-managed-thing-capabilities](#) commande dans le manuel de référence des AWS CLI commandes des intégrations gérées.

Étape 6 : Envoyer une commande à l'objet géré

Vous pouvez utiliser cette `send-managed-thing-command` commande pour envoyer une commande d'action à l'objet que vous gérez.

Envoyez une commande à l'objet géré à l'aide d'une action de bascule.

- Utilisez la `send-managed-thing-command` commande pour envoyer une commande d'action à bascule.

Exemple `send-managed-thing-command`

```
json=$(jq -cr '.|@json') <<EOF
[
  {
    "endpointId": "1",
    "capabilities": [
      {
        "id": "matter.OnOff@1.4",
        "name": "On/Off",
        "version": "1",
        "actions": [
          {
            "name": "Toggle",
            "parameters": {}
          }
        ]
      }
    ]
  }
]
EOF
aws iot-managed-integrations send-managed-thing-command \
--managed-thing-id ${device_managed_thing_id} --endpoints ENDPOINT_ID
```

Note

Cet exemple utilise `jq cli to`, mais vous pouvez également transmettre la chaîne entière `endpointId`

Réponse :

```
{
  "TraceId": TRACE_ID
```



```
}
```

Pour plus d'informations, consultez la [send-managed-thing-command](#) commande dans le manuel de référence des AWS CLI commandes des intégrations gérées.

Étape 7 : vérifier l'état de l'objet géré

Vérifiez l'état de l'objet géré pour valider la réussite de l'action de bascule.

Pour vérifier l'état de l'appareil d'un objet géré

- Utilisez la `get-managed-thing-state` commande pour valider la réussite de l'action de bascule.

Exemple `get-managed-thing-state`

```
aws iot-managed-integrations get-managed-thing-state --managed-thing-id DEVICE_MANAGED_THING_ID
```

Réponse :

```
{
  "Endpoints": [
    {
      "endpointId": "1",
      "capabilities": [
        {
          "id": "matter.OnOff@1.4",
          "name": "On/Off",
          "version": "1.4",
          "properties": [
            {
              "name": "OnOff",
              "value": {
                "propertyValue": true,
                "lastChangedAt": "2025-06-03T21:50:39.886Z"
              }
            }
          ]
        }
      ]
    }
  ]
}
```

```
}  
  ]  
}  
]
```

Pour plus d'informations, consultez la [get-managed-thing-state](#) commande dans le manuel de référence des AWS CLI commandes des intégrations gérées.

Étape 8 : Supprimer l'objet géré de votre hub

Nettoyez votre hub en supprimant l'élément géré.

Pour supprimer un objet géré

- Utilisez la [delete-managed-thing](#) commande pour supprimer un objet géré.

Exemple delete-managed-thing

```
aws iot-managed-integrations delete-managed-thing \  
  --identifiant MANAGED_THING_ID
```

Pour plus d'informations, consultez la [delete-managed-thing](#) commande dans le manuel de référence des AWS CLI commandes des intégrations gérées.

Note

Si l'appareil est bloqué dans un DELETE_IN_PROGRESS état, ajoutez l'--forceindicateur à la delete-managed-thing commande.

Note

Pour les appareils Z-wave, vous devez mettre l'appareil en mode couplage après avoir exécuté la commande.

Créez un gestionnaire de certificats personnalisé pour un stockage sécurisé

La gestion des certificats des appareils est cruciale lors de l'intégration au hub d'intégrations gérées. Bien que les certificats soient stockés dans le système de fichiers par défaut, vous pouvez créer un gestionnaire de certificats personnalisé pour une sécurité renforcée et une gestion flexible des informations d'identification.

Le SDK End Device pour les intégrations gérées fournit un gestionnaire de certificats pour une interface de stockage sécurisée que vous pouvez implémenter sous la forme d'une bibliothèque d'objets partagés (.so). Créez votre implémentation de stockage sécurisé pour lire et écrire des certificats, puis liez le fichier de bibliothèque au HubOnboarding processus lors de l'exécution.

Définition et composants de l'API

Consultez le `secure_storage_cert_handler_interface.hpp` fichier suivant pour comprendre les composants de l'API et les exigences de votre implémentation

Rubriques

- [Définition de l'API](#)
- [Composants clés](#)

Définition de l'API

Contenu de `secure_storage_cert_handler_interface.hpp`

```
/*
 * Copyright 2024 Amazon.com, Inc. or its affiliates. All rights reserved.
 *
 * AMAZON PROPRIETARY/CONFIDENTIAL
 *
 * You may not use this file except in compliance with the terms and
 * conditions set forth in the accompanying LICENSE.txt file.
 *
 * THESE MATERIALS ARE PROVIDED ON AN "AS IS" BASIS. AMAZON SPECIFICALLY
 * DISCLAIMS, WITH RESPECT TO THESE MATERIALS, ALL WARRANTIES, EXPRESS,
 * IMPLIED, OR STATUTORY, INCLUDING THE IMPLIED WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT.
 */
```

```

#ifndef SECURE_STORAGE_CERT_HANDLER_INTERFACE_HPP
#define SECURE_STORAGE_CERT_HANDLER_INTERFACE_HPP

#include <iostream>
#include <memory>

namespace IoTManagedIntegrationsDevice {
namespace CertHandler {
/**
 * @enum CERT_TYPE_T
 * @brief enumeration defining certificate types.
 */
typedef enum { CLAIM = 0, DHA = 1, PERMANENT = 2 } CERT_TYPE_T;
class SecureStorageCertHandlerInterface {
public:
    /**
     * @brief Read certificate and private key value of a particular certificate
     * type from secure storage.
     */
    virtual bool read_cert_and_private_key(const CERT_TYPE_T cert_type,
                                           std::string &cert_value,
                                           std::string &private_key_value) = 0;

    /**
     * @brief Write permanent certificate and private key value to secure storage.
     */
    virtual bool write_permanent_cert_and_private_key(
        std::string_view cert_value, std::string_view private_key_value) = 0;
};
    std::shared_ptr<SecureStorageCertHandlerInterface>
createSecureStorageCertHandler();
} //namespace CertHandler
} //namespace IoTManagedIntegrationsDevice

#endif //SECURE_STORAGE_CERT_HANDLER_INTERFACE_HPP

```

Composants clés

- CERT_TYPE_T : différents types de certificats sur le hub.
 - RÉCLAMATION - le certificat de réclamation initialement enregistré sur le hub sera échangé contre un certificat permanent.
 - DHA : inutilisé pour le moment.

- **PERMANENT** - certificat permanent pour se connecter au point de terminaison des intégrations gérées.
- **read_cert_and_private_key** - (FONCTION À IMPLÉMENTER) Lit le certificat et la valeur de la clé dans l'entrée de référence. Cette fonction doit être capable de lire à la fois le certificat CLAIM et le certificat PERMANENT, et elle est différenciée par le type de certificat mentionné ci-dessus.
- **write_permanent_cert_and_private_key** - (FONCTION À IMPLÉMENTER) écrit le certificat permanent et la valeur de la clé à l'emplacement souhaité.

Exemple de construction

Séparez vos en-têtes d'implémentation internes de l'interface publique (`secure_storage_cert_handler_interface.hpp`) pour conserver une structure de projet propre. Grâce à cette séparation, vous pouvez gérer les composants publics et privés lors de la création de votre gestionnaire de certificats.

Note

Déclarer `secure_storage_cert_handler_interface.hpp` comme public.

Rubriques

- [Structure du projet](#)
- [Hériter de l'interface](#)
- [Mise en œuvre](#)
- [CMakeList.txt](#)

Structure du projet

Hériter de l'interface

Créez une classe concrète qui hérite de l'interface. Masquez ce fichier d'en-tête et les autres fichiers dans un répertoire distinct afin de différencier facilement les en-têtes privés et publics lors de la création.

```
#ifndef IOTMANAGEDINTEGRATIONSDEVICE_SDK_STUB_SECURE_STORAGE_CERT_HANDLER_HPP
```

```
#define IOTMANAGEDINTEGRATIONSDEVICE_SDK_STUB_SECURE_STORAGE_CERT_HANDLER_HPP

#include "secure_storage_cert_handler_interface.hpp"

namespace IoTManagedIntegrationsDevice::CertHandler {
    class StubSecureStorageCertHandler : public SecureStorageCertHandlerInterface {
    public:
        StubSecureStorageCertHandler() = default;

        bool read_cert_and_private_key(const CERT_TYPE_T cert_type,
                                      std::string &cert_value,
                                      std::string &private_key_value) override;

        bool write_permanent_cert_and_private_key(
            std::string_view cert_value, std::string_view private_key_value) override;
        /*
         * any other resource for function you might need
         */

    };
}

#endif //IOTMANAGEDINTEGRATIONSDEVICE_SDK_STUB_SECURE_STORAGE_CERT_HANDLER_HPP
```

Mise en œuvre

Implémentez la classe de stockage définie ci-dessus,src/
stub_secure_storage_cert_handler.cpp.

```
/*
 * Copyright 2024 Amazon.com, Inc. or its affiliates. All rights reserved.
 *
 * AMAZON PROPRIETARY/CONFIDENTIAL
 *
 * You may not use this file except in compliance with the terms and
 * conditions set forth in the accompanying LICENSE.txt file.
 *
 * THESE MATERIALS ARE PROVIDED ON AN "AS IS" BASIS. AMAZON SPECIFICALLY
 * DISCLAIMS, WITH RESPECT TO THESE MATERIALS, ALL WARRANTIES, EXPRESS,
 * IMPLIED, OR STATUTORY, INCLUDING THE IMPLIED WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT.
```

```

*/

#include "stub_secure_storage_cert_handler.hpp"

using namespace IoTManagedIntegrationsDevice::CertHandler;

bool StubSecureStorageCertHandler::write_permanent_cert_and_private_key(
    std::string_view cert_value, std::string_view private_key_value) {
    // TODO: implement write function
    return true;
}

bool StubSecureStorageCertHandler::read_cert_and_private_key(const CERT_TYPE_T
cert_type,
                                                             std::string &cert_value,
                                                             std::string
&private_key_value) {
    std::cout<<"Using Stub Secure Storage Cert Handler, returning dummy values";
    cert_value = "StubCertVal";
    private_key_value = "StubKeyVal";
    // TODO: implement read function
    return true;
}

```

Implémentez la fonction d'usine définie dans l'interface, `src/secure_storage_cert_handler.cpp`.

```

#include "stub_secure_storage_cert_handler.hpp"

std::shared_ptr<IoTManagedIntegrationsDevice::CertHandler::SecureStorageCertHandlerInterface>
IoTManagedIntegrationsDevice::CertHandler::createSecureStorageCertHandler() {
    // TODO: replace with your implementation
    return
std::make_shared<IoTManagedIntegrationsDevice::CertHandler::StubSecureStorageCertHandler>();
}

```

CMakeList.txt

```
#project name must stay the same
project(SecureStorageCertHandler)

# Public Header files. The interface definition must be in top level with exactly
the same name
#ie. Not in anotherDir/secure_storage_cert_handler_interface.hpp
set(PUBLIC_HEADERS
    ${PROJECT_SOURCE_DIR}/include
)

# private implementation headers.
set(PRIVATE_HEADERS
    ${PROJECT_SOURCE_DIR}/internal/stub
)

#set all sources
set(SOURCES
    ${PROJECT_SOURCE_DIR}/src/secure_storage_cert_handler.cpp
    ${PROJECT_SOURCE_DIR}/src/stub_secure_storage_cert_handler.cpp
)

# Create the shared library
add_library(${PROJECT_NAME} SHARED ${SOURCES})
target_include_directories(
    ${PROJECT_NAME}
    PUBLIC
        ${PUBLIC_HEADERS}
    PRIVATE
        ${PRIVATE_HEADERS}
)

# Set the library output location. Location can be customized but version must
stay the same
set_target_properties(${PROJECT_NAME} PROPERTIES
    LIBRARY_OUTPUT_DIRECTORY ${CMAKE_BINARY_DIR}/../lib
    VERSION 1.0
    SOVERSION 1
)

# Install rules
install(TARGETS ${PROJECT_NAME}
```



```
        LIBRARY DESTINATION lib
        ARCHIVE DESTINATION lib
    )

    install(FILES ${HEADERS}
            DESTINATION include/SecureStorageCertHandler
    )
```

Utilisation

Après la compilation, vous disposerez d'un fichier de bibliothèque d'objets `libSecureStorageCertHandler.so` partagé et de ses liens symboliques associés. Copiez le fichier de bibliothèque et les liens symboliques vers l'emplacement de bibliothèque attendu par le HubOnboarding binaire.

Rubriques

- [Considérations clés](#)
- [Utilisez un stockage sécurisé](#)

Considérations clés

- Vérifiez que votre compte utilisateur dispose d'autorisations de lecture et d'écriture à la fois pour le HubOnboarding binaire et pour la `libSecureStorageCertHandler.so` bibliothèque.
- `secure_storage_cert_handler_interface.hpp` Conservez-le comme seul fichier d'en-tête public. Tous les autres fichiers d'en-tête doivent rester dans votre implémentation privée.
- Vérifiez le nom de votre bibliothèque d'objets partagés. Lors de la compilation `libSecureStorageCertHandler.so`, le nom de fichier HubOnboarding peut nécessiter une version spécifique, telle que `libSecureStorageCertHandler.so.1.0`. Utilisez la `ldd` commande pour vérifier les dépendances des bibliothèques et créer des liens symboliques selon les besoins.
- Si votre implémentation de la bibliothèque partagée comporte des dépendances externes, stockez-les dans un répertoire HubOnboarding accessible, tel qu'un `/usr/lib` or the `iotmi_common` répertoire.

Utilisez un stockage sécurisé

Mettez à jour votre `iotmi_config.json` fichier en configurant à la fois `iot_claim_cert_path` et `iot_claim_pk_path` sur **SECURE_STORAGE**.

```
{
  "ro": {
    "iot_provisioning_method": "FLEET_PROVISIONING",
    "iot_claim_cert_path": "SECURE_STORAGE",
    "iot_claim_pk_path": "SECURE_STORAGE",
    "fp_template_name": "device-integration-example",
    "iot_endpoint_url": "[ACCOUNT-PREFIX]-ats.iot.AWS-REGION.amazonaws.com",
    "SN": "1234567890",
    "UPC": "1234567890"
  },
  "rw": {
    "iot_provisioning_state": "NOT_PROVISIONED"
  }
}
```

Plug-in de protocole personnalisé

Vous pouvez utiliser un plugin de protocole personnalisé pour intégrer vos protocoles IoT propriétaires dans les intégrations gérées pour AWS IoT Device Management l'écosystème. Grâce à des interfaces SDK bien définies, vous pouvez intégrer des appareils, définir des fonctionnalités et gérer des flux de contrôle en temps réel tout en maintenant une compatibilité totale avec les intégrations gérées et les composants du SDK du hub.

Les listes suivantes présentent les principales fonctionnalités du plugin de protocole personnalisé.

Personnalisation du modèle de données

Définissez vos propres schémas de modèles de AWS données et chargez-les dans des intégrations gérées pendant le flux de provisionnement. Vous pouvez utiliser ces schémas ultérieurement dans vos flux de travail.

Implémentation flexible des plugins

- Créez votre propre composant de plugin en utilisant [Client du SDK Hub](#).
- Implémentez des plug-ins distincts pour différentes fonctionnalités, telles que le provisionnement et le contrôle, ou créez un client unifié pour les deux.

- Maintenez une frontière claire entre les actifs d'intégrations gérés et vos propres actifs, tels que les piles de middleware, pour une implémentation de logique de code découplée et conviviale pour le développement.

Rétrocompatibilité

Pour les clients existants, intégrez facilement votre nouveau protocole personnalisé tout en maintenant le fonctionnement des types de radio existants tels quels.

Le schéma suivant illustre l'architecture du plug-in de protocole personnalisé.

Client du SDK Hub

La bibliothèque client du SDK Hub sert d'interface entre le SDK Hub pour les intégrations gérées et votre propre pile de protocoles exécutée sur le même hub. Il expose un ensemble de données publiques APIs pour faciliter l'interaction de votre pile de protocoles avec les composants du SDK Device Hub. Les cas d'utilisation incluent le contrôle personnalisé des plugins, le provisionneur de plugins personnalisé et le contrôleur local.

Rubriques

- [Obtenez votre SDK Hub d'intégrations gérées](#)
- [À propos de la boîte à outils du SDK Hub](#)
- [Créez votre application personnalisée avec le client Hub SDK](#)
- [Exécution de votre application personnalisée](#)
- [Référence de l'API du client Hub SDK](#)
- [Types de données](#)

Obtenez votre SDK Hub d'intégrations gérées

Le client Hub SDK est fourni avec le SDK d'intégrations gérées. Contactez-nous depuis la [console des intégrations gérées](#) pour accéder au SDK du hub.

À propos de la boîte à outils du SDK Hub

Après le téléchargement, vous verrez un `IotMI-DeviceSDK-Toolkit` dossier contenant tous les fichiers d'en-tête publics et `.so` les fichiers que vous pouvez utiliser dans votre application. L'équipe

des intégrations gérées fournit également un exemple `main.cpp` à des fins de démonstration, ainsi que le fichier binaire de l'application de démonstration `bin/` que vous pouvez directement exécuter. Vous pouvez éventuellement l'utiliser comme point de départ pour votre application.

Créez votre application personnalisée avec le client Hub SDK

Suivez les étapes ci-dessous pour créer votre application personnalisée.

1. Incluez les fichiers d'en-tête (.h) et les fichiers d'objets partagés (.so) dans votre application.

Vous devez inclure les fichiers d'en-tête publics (.h) et les fichiers d'objets partagés (.so) dans votre application. Pour les fichiers .so, vous pouvez les placer dans un dossier lib. La mise en page finale sera similaire à celle-ci :

```
### include
#   ### iotmi_device_sdk_client
#   #   ### iotmi_device_sdk_client_common_types.h
#   ### iotmi_device_sdk_client.h
#   ### iotshd_status.h
### lib
#   ### libiotmi_devicesdk_client_module.so
#   ### libiotmi_log_c.so
```

2. Créez un client Hub SDK dans votre application principale.
 - a. Dans votre application principale, vous devez d'abord initialiser le client du SDK Hub avant de pouvoir l'utiliser pour traiter les demandes. Vous pouvez simplement créer le client avec `unclientId`.
 - b. Une fois que vous avez le client, vous pouvez le connecter au SDK Device pour les intégrations gérées.

Voici un exemple de création du client du SDK Hub et de la procédure de connexion.

```
#include <cstdlib>
#include <string>
#include "iotshd_status.h"
#include "iotmi_device_sdk_client.h"

auto client = std::make_unique<DeviceSDKClient>(your_own_clientId);
iotmi_statusCode_t status = client->connect();
```

 Note

your_own_clientId doit être identique à celui que vous spécifiez [start-device-discovery](#) dans la configuration guidée par l'utilisateur ou [create-managed-thing](#) dans le flux de provisionnement de configuration simple.

3. Publiez et abonnez-vous en suivant les étapes ci-dessous.

- a. Une fois la connexion établie, vous pouvez désormais vous abonner aux tâches entrantes à partir du SDK du hub d'intégrations gérées. Les tâches entrantes peuvent être des tâches de contrôle ou des tâches de provisionnement. Vous devez également définir votre propre fonction de rappel lors de la réception des tâches et votre propre contexte personnalisé à des fins de suivi.

```
// subscribe to provisioning tasks
iotmi_statusCode_t status = client->iotmi_provision_subscribe_to_tasks(
    example_subscriber_callback, custom_context);

// subscribe to control tasks
iotmi_statusCode_t status = client->iotmi_control_subscribe_to_tasks(
    example_subscriber_callback, custom_context);
```

- b. Une fois la connexion établie, vous pouvez désormais publier les demandes de votre application vers le SDK du Hub d'intégrations gérées. Vous pouvez définir votre propre type de message de tâche, avec différentes charges utiles pour différents objectifs commerciaux. Les demandes peuvent inclure à la fois des demandes de contrôle et des demandes de fourniture, comme dans le cas du flux d'abonnement. Enfin, vous pouvez attribuer une adresse pour `rspPayload` obtenir la réponse du SDK du Hub d'intégrations gérées de manière synchronisée.

```
// publish control request
iotmi_client_request_t api_payload = {
    .messageType = C2MIMessageType::C2MI_CONTROL_EVENT,
    .reqPayload = (uint8_t *)"define_your_req_payload",
    .rspPayload = (uint8_t *)calloc(1000, sizeof(uint8_t))
};

status = client->iotmi_control_publish_request(&api_payload);
```

```
// publish provision request
iotmi_client_request_t api_payload = {
    .messageType = C2MIMessageType::C2MI_DEVICE_ONBOARDED,
    .reqPayload = (uint8_t *)"define_your_req_payload",
    .rspPayload = (uint8_t *)calloc(1000, sizeof(uint8_t))
};
status = client->iotmi_provision_publish_request(&api_payload);
```

4. Créez le votre CMakeLists.txt et créez votre application à partir de là. La sortie finale peut être un fichier binaire exécutable tel que MyFirstApplication

Exécution de votre application personnalisée

Avant d'exécuter votre application personnalisée, suivez les étapes suivantes pour configurer votre hub et démarrer le SDK du hub d'intégrations gérées :

- Suivez les instructions d'intégration à [Intégrez vos hubs aux intégrations gérées](#) l'adresse.
- Terminez le processus d'installation décrit dans [Installation et validation du SDK Hub pour les intégrations gérées](#).

Une fois les conditions requises remplies, vous pouvez exécuter votre application personnalisée. Par exemple :

```
./MyFirstApplication
```

Important

Vous devez mettre à jour manuellement le script de démarrage indiqué dans la liste [Déployez le SDK du Hub à l'aide d'un script](#) avec un script pour démarrer votre propre application.

L'ordre est important, ne le modifiez pas.

Mettez à jour les éléments suivants. Modification

```
./IotMI-DeviceSDK-Toolkit/bin/DeviceSDKClientDemo >> $LOGS_DIR/  
logDeviceSDKClientDemo_logs.txt &
```

to

```
./MyFirstApplication >> $LOGS_DIR/MyFirstApplication_logs.txtt &
```

Référence de l'API du client Hub SDK

Le client Hub SDK (SDKClient classe Device) fournit une interface permettant à votre application personnalisée d'interagir avec le SDK Device des intégrations gérées. Avec ce client, vous pouvez effectuer les opérations suivantes :

- Abonnez-vous aux tâches liées à l'approvisionnement et au contrôle à partir des composants d'intégrations gérées.
- Publiez les demandes liées à l'approvisionnement et au contrôle dans les composants des intégrations gérées.

Pour plus d'informations sur les intégrations gérées pour AWS IoT Device Management APIs, consultez [Qu'est-ce AWS Lambda que](#).

Rubriques

- [Initialisation du client](#)
- [Abonnement aux tâches de provisionnement](#)
- [Publication des tâches de provisionnement](#)
- [Abonnement aux tâches de contrôle](#)
- [Publication des tâches de contrôle](#)
- [Fonctionnalités de journalisation](#)
- [Autres APIs](#)

Initialisation du client

Pour commencer à utiliser leDeviceSDKClient, initialisez-le avec un ID client.

```
iotmi_statusCode_t DeviceSDKClient(const std::string& clientId)
```

Cela crée une nouvelle DeviceSDKClient instance avec la valeur spécifiéeclientId. clientIdll doit correspondre à celui que vous enregistrez auprès des intégrations gérées.

Paramètres

`clientId(string)` - L'ID client de cette instance.

```
connect()
```

Connecte l'`DeviceSDKClient` instance aux intégrations gérées.

Renvoie

- `IOTMI_STATUS_OK`- La connexion a été établie.
- `IOTMI_STATUS_CUSTOM_PLUGIN_CONNECTION_ERROR`- Une erreur s'est produite lors de la connexion aux intégrations gérées.

Abonnement aux tâches de provisionnement

Utilisez ces méthodes pour vous abonner aux tâches liées à l'approvisionnement à partir des composants d'intégrations gérées.

```
iotmi_statusCode_t  
iotmi_provision_subscribe_to_tasks(DeviceSDKClient_SubscriberCallback callback, char*  
context)
```

S'abonne aux tâches liées à l'approvisionnement, telles que l'intégration et le déprovisionnement des appareils, à partir des composants d'intégration gérés.

Paramètres

- `callback(Device SDKClient _SubscriberCallback)` - Fonction de rappel qui s'exécute lorsqu'une tâche est reçue.
- `context(char*)` - Un contexte personnalisé transmis à la fonction de rappel.

Renvoie

- `IOTMI_STATUS_OK`- L'abonnement a été effectué avec succès.
- `IOTMI_STATUS_CUSTOM_PLUGIN_CLIENT_NOT_CONNECTED`- L'`SDKClient` instance Device n'est pas connectée aux intégrations gérées.
- `IOTMI_STATUS_CUSTOM_PLUGIN_SUBSCRIBE_ERROR`- Une erreur s'est produite lors de l'inscription aux tâches.

Publication des tâches de provisionnement

Utilisez ces méthodes pour publier les demandes relatives à l'approvisionnement auprès des composants d'intégrations gérées.

```
iotmi_statusCode_t iotmi_provision_publish_request(DataModel::iotmi_client_request_t request)
```

Publie une demande relative à la fourniture aux composants des intégrations gérées. Par exemple, un événement d'intégration d'un appareil ou un état de déprovisionnement

Paramètres

`request(DataModel: :iotmi_client_request_t)` - Un pointeur vers une structure de demande contenant les détails.

Renvoie

- `IOTMI_STATUS_OK`- La demande a été publiée avec succès.
- `IOTMI_STATUS_CUSTOM_PLUGIN_CLIENT_NOT_CONNECTED`- L'`DeviceSDKClientInstance` n'est pas connectée aux intégrations gérées.
- `IOTMI_STATUS_INVALID_PARAMETER`- Un ou plusieurs paramètres de la demande ne sont pas valides.
- `IOTMI_STATUS_INVALID_JSON_OBJECT`- La charge utile de la demande n'est pas un objet JSON valide.
- `IOTMI_STATUS_NO_MEMORY`- Une erreur d'allocation de mémoire s'est produite.

Abonnement aux tâches de contrôle

Utilisez ces méthodes pour vous abonner aux tâches liées au contrôle à partir des composants d'intégrations gérées.

```
iotmi_statusCode_t iotmi_control_subscribe_to_tasks(DeviceSDKClient_SubscriberCallback callback, char context)
```

S'abonne aux tâches liées au contrôle (par exemple, les demandes de contrôle des appareils) à partir des composants d'intégrations gérées.

Paramètres

- `callback(Device SDKClient_SubscriberCallback)` - Fonction de rappel qui s'exécute lorsqu'une tâche est reçue.
- `context(char)` - Un contexte personnalisé transmis à la fonction de rappel.

Renvoie

- `IOTMI_STATUS_OK`- L'abonnement a été effectué avec succès.
- `IOTMI_STATUS_CUSTOM_PLUGIN_CLIENT_NOT_CONNECTED`- L'`DeviceSDKClientinstance` n'est pas connectée aux intégrations gérées.
- `IOTMI_STATUS_CUSTOM_PLUGIN_SUBSCRIBE_ERROR`- Une erreur s'est produite lors de l'inscription aux tâches.

Publication des tâches de contrôle

Utilisez ces méthodes pour publier les demandes liées au contrôle dans les composants des intégrations gérées.

```
iotmi_statusCode_t iotmi_control_publish_request(DataModel::iotmi_client_request_t request)
```

Publie une demande liée au contrôle auprès des composants des intégrations gérées. Par exemple, des événements non sollicités, des demandes de commande ou des requêtes sur l'état de l'appareil.

Paramètres

`request(DataModel: :iotmi_client_request_t)` - Un pointeur vers une structure de demande contenant les détails.

Renvoie

- `IOTMI_STATUS_OK`- La demande a été publiée avec succès.
- `IOTMI_STATUS_CUSTOM_PLUGIN_CLIENT_NOT_CONNECTED`- L'`SDKClient instance Device` n'est pas connectée aux intégrations gérées.
- `IOTMI_STATUS_INVALID_PARAMETER`- Un ou plusieurs paramètres de la demande ne sont pas valides.
- `IOTMI_STATUS_INVALID_JSON_OBJECT`- La charge utile de la demande n'est pas un objet JSON valide.

- `IOTMI_STATUS_NO_MEMORY`- Une erreur d'allocation de mémoire s'est produite.

Fonctionnalités de journalisation

Utilisez ces méthodes pour implémenter les fonctionnalités de journalisation fournies par les intégrations gérées.

Initialisation de l'enregistreur

```
void iotmi_devicesdk_log_init(const char* logger_name)
```

Vous devez initialiser l'enregistreur avant d'utiliser toute fonctionnalité de journalisation.

Paramètres

`logger_name`- Le nom de l'enregistreur que vous spécifiez. La valeur par défaut est : `MyApplication`

Macros d'enregistrement

`LOGGER_LOGD(...)`

Utilisez cette macro dans votre application pour la journalisation au niveau `DEBUG`.

`LOGGER_LOGI(...)`

Utilisez cette macro dans votre application pour la journalisation au niveau `INFO`.

`LOGGER_LOGW(...)`

Utilisez cette macro dans votre application pour la journalisation au niveau `WARN`.

`LOGGER_LOGE(...)`

Utilisez cette macro dans votre application pour la journalisation des niveaux d'erreur.

Note

Pour plus d'informations sur les fonctionnalités de journalisation, consultez la [documentation de journalisation du Hub](#). Les plugins de protocole personnalisés prennent entièrement en charge toutes les fonctionnalités de journalisation proposées par les intégrations gérées.

Autres APIs

```
std::string get_client_id()
```

Renvoie l'ID client associé à l'SDKClient instance Device.

Renvoie

ID client.

Types de données

Cette section définit les types de données utilisés pour le plug-in de protocole personnalisé.

iotmi_client_request_t

Représente une demande à publier dans les composants des intégrations gérées.

MessageType

Type du message (CommonTypes: :C2 MIMessage Type). La liste suivante indique les valeurs valides.

- C2MI_DEVICE_ONBOARDED: indique un message d'intégration de l'appareil avec la charge utile associée.
- C2MI_DE_PROVISIONING_PRE_ASSOCIATED_COMPLETE: indique une notification de fin de tâche de déprovisionnement pour un appareil pré-associé.
- C2MI_DE_PROVISIONING_ACTIVATED_COMPLETE: indique une notification de fin de tâche de déprovisionnement pour un appareil activé.
- C2MI_DE_PROVISIONING_COMPLETE_RESPONSE: indique une réponse complète à une tâche de déprovisionnement.
- C2MI_CONTROL_EVENT: indique un événement de contrôle susceptible de modifier l'état de l'appareil.
- C2MI_CONTROL_SEND_COMMAND: indique une commande de contrôle provenant d'un contrôleur local.
- C2MI_CONTROL_SEND_DEVICE_STATE_QUERY: indique une demande d'état du dispositif de contrôle provenant d'un contrôleur local.

Charge utile Req

La charge utile de la demande, généralement une chaîne au format JSON.

Charge utile RSP

La charge utile de réponse, alimentée par les composants des intégrations gérées.

iotmi_client_event_t

Représente un événement reçu des composants d'intégrations gérées.

identifiant_événement

Identifiant unique de l'événement.

longueur

Longueur des données de l'événement.

data

Un pointeur vers les données de l'événement, y compris le `messageType`. La liste suivante indique les valeurs possibles.

- `C2MI_PROVISION_UGS_TASK`: indique une tâche de provisionnement pour le flux UGS.
- `C2MI_PROVISION_SS_TASK`: indique une tâche de provisionnement pour le SimpleSetup flux.
- `C2MI_DE_PROVISION_PRE_ASSOCIATED_TASK`: indique une tâche de déprovisionnement pour un appareil pré-associé.
- `C2MI_DE_PROVISION_ACTIVATED_TASK`: indique une tâche de déprovisionnement pour un appareil activé.
- `C2MI_DEVICE_ONBOARDED_RESPONSE`: indique une réponse d'intégration de l'appareil.
- `C2MI_CONTROL_TASK`: indique une tâche de contrôle.
- `C2MI_CONTROL_EVENT_NOTIFICATION`: indique une notification d'événement de contrôle pour un contrôleur local.

ctx

Contexte personnalisé associé à l'événement.

Contrôle du hub

Le contrôle du hub est une extension du SDK pour appareils finaux des intégrations gérées qui lui permet de s'interfacer avec le MQTTProxy composant du SDK du hub. Avec le contrôle du hub, vous pouvez implémenter du code à l'aide du SDK pour appareils finaux et contrôler votre hub via le cloud d'intégrations gérées en tant qu'appareil distinct. Le SDK de contrôle du hub sera fourni sous forme de package distinct dans le SDK du hub, étiqueté comme `iot-managed-integrations-hub-control-x.x.x`

Rubriques

- [Prérequis](#)
- [Composants du SDK pour appareils finaux](#)
- [Intégrer au SDK pour appareils finaux](#)
- [Exemple : créer un contrôle du hub](#)
- [Exemples pris en charge](#)
- [Plateformes prises en charge](#)

Prérequis

Pour configurer le contrôle du hub, vous avez besoin des éléments suivants :

- Un hub intégré au [SDK du Hub](#), version 0.4.0 ou ultérieure.
- Téléchargez la dernière version du [SDK pour appareils finaux](#) à partir du AWS Management Console.
- Un composant [proxy MQTT](#) exécuté sur le hub, version 0.5.0 ou supérieure.

Composants du SDK pour appareils finaux

Utilisez les composants suivants du [SDK pour appareils finaux](#) :

- Générateur de code pour le modèle de données
- Gestionnaire de modèles de données

Étant donné que le SDK Hub dispose déjà d'un processus d'intégration et d'une connexion au cloud, vous n'avez pas besoin des composants suivants :

- Bénéficiaire
- Interface PKCS
- Gestionnaire de tâches
- Agent MQTT

Intégrer au SDK pour appareils finaux

1. Suivez les instructions de la section [Générateur de code pour le modèle de données](#) afin de générer le code C de bas niveau.
2. Suivez les instructions de la section [Intégration du SDK de l'appareil final](#) pour :
 - a. Configuration de l'environnement de construction

Créez le code sur Amazon Linux 2023/x86_64 en tant qu'hôte de développement. Installez les dépendances de build nécessaires :

```
dnf install make gcc gcc-c++ cmake
```

- b. Développer des fonctions de rappel matérielles

Avant d'implémenter les fonctions de rappel matériel, comprenez le fonctionnement de l'API. Cet exemple utilise le On/Off cluster et OnOff l'attribut pour contrôler la fonction d'un appareil. Pour plus de détails sur l'API, consultez [Fonction C de faible niveau APIs](#).

```
struct DeviceState
{
    struct iotmiDev_Agent *agent;
    struct iotmiDev_Endpoint *endpointLight;
    /* This simulates the HW state of OnOff */
    bool hwState;
};

/* This implementation for OnOff getter just reads
the state from the DeviceState */
iotmiDev_DMStatus exampleGetOnOff(bool *value, void *user)
{
    struct DeviceState *state = (struct DeviceState *)(user);
    *value = state->hwState;
    return iotmiDev_DMStatusOk;
```

```
}

```

c. Configurer les points de terminaison et connecter les fonctions de rappel du matériel

Après avoir implémenté les fonctions, créez des points de terminaison et enregistrez vos rappels. Effectuez les tâches suivantes :

- i. Création d'un agent de terminal
- ii. Remplissez les points de la fonction de rappel pour chaque structure de cluster que vous souhaitez prendre en charge
- iii. Configuration des points de terminaison et enregistrement des clusters pris en charge

```
struct DeviceState
{
    struct iotmiDev_Agent * agent;
    struct iotmiDev_Endpoint *endpoint1;

    /* OnOff cluster states*/
    bool hwState;
};

/* This implementation for OnOff getter just reads
the state from the DeviceState */
iotmiDev_DMStatus exampleGetOnOff( bool * value, void * user )
{
    struct DeviceState * state = ( struct DeviceState * ) ( user );
    *value = state->hwState;
    printf( "%s(): state->hwState: %d\n", __func__, state->hwState );
    return iotmiDev_DMStatusOk;
}

iotmiDev_DMStatus exampleGetOnTime( uint16_t * value, void * user )
{
    *value = 0;
    printf( "%s(): OnTime is %u\n", __func__, *value );
    return iotmiDev_DMStatusOk;
}

iotmiDev_DMStatus exampleGetStartUpOnOff( iotmiDev_OnOff_StartUpOnOffEnum *
value, void * user )
```



```

{
    *value = iotmiDev_OnOff_StartUpOnOffEnum_Off;
    printf( "%s(): StartUpOnOff is %d\n", __func__, *value );
    return iotmiDev_DMStatusOk;
}

void setupOnOff( struct DeviceState *state )
{
    struct iotmiDev_clusterOnOff clusterOnOff = {
        .getOnOff = exampleGetOnOff,
        .getOnTime = exampleGetOnTime,
        .getStartUpOnOff = exampleGetStartUpOnOff,
    };
    iotmiDev_OnOffRegisterCluster( state->endpoint1,
                                   &clusterOnOff,
                                   ( void * ) state);
}

/* Here is the sample setting up an endpoint 1 with OnOff
   cluster. Note all error handling code is omitted. */
void setupAgent(struct DeviceState *state)
{
    struct iotmiDev_Agent_Config config = {
        .thingId = IOTMI_DEVICE_MANAGED_THING_ID,
        .clientId = IOTMI_DEVICE_CLIENT_ID,
    };
    iotmiDev_Agent_InitDefaultConfig(&config);

    /* Create a device agent before calling other SDK APIs */
    state->agent = iotmiDev_Agent_new(&config);

    /* Create endpoint#1 */
    state->endpoint1 = iotmiDev_Agent_addEndpoint( state->agent,
                                                    1,
                                                    "Data Model Handler Test
Device",
                                                    (const char*[])
{ "Camera" },
                                                    1 );

    setupOnOff(state);
}

```

Exemple : créer un contrôle du hub

Le contrôle du hub est fourni dans le package du SDK du hub. Le sous-package de contrôle du hub est étiqueté `iot-managed-integrations-hub-control-x.x.x` et contient des bibliothèques différentes de celles du SDK de périphérique non modifié.

1. Déplacez les fichiers générés par le code vers le exemple dossier :

```
cp codegen/out/* example/dm
```

2. Pour créer le contrôle du hub, exécutez les commandes suivantes :

```
cd <hub-control-root-folder>
```

```
mkdir build
```

```
cd build
```

```
cmake -DBUILD_EXAMPLE_WITH_MQTT_PROXY=ON -  
DIOTMI_USE_MANAGED_INTEGRATIONS_DEVICE_LOG=ON ..
```

```
cmake -build .
```

3. Exécutez les exemples avec le MQTTProxy composant sur le hub, avec les MQTTProxy composants HubOnboarding et en cours d'exécution.

```
./examples/iotmi_device_sample_camera/iotmi_device_sample_camera
```

Voir [Modèle de données d'intégrations gérées](#) pour le modèle de données. Suivez l'étape 5 [Commencez avec le SDK pour appareils finaux](#) pour configurer les points de terminaison et gérer les communications entre l'utilisateur final et. `iot-managed-integrations`

Exemples pris en charge

Les exemples suivants ont été créés et testés :

- Démo du purificateur d'air `iotmi_device_dm_`

- `iotmi_device_basic_diagnostics`
- `iotmi_device_dm_camera_demo`

Plateformes prises en charge

Le tableau suivant indique les plateformes prises en charge pour le contrôle du hub.

Architecture	Système d'exploitation	Version GCC	Version binutils
X86_64	Linux	10.5.0	2,37
aarch64	Linux	10.5.0	2,37

Activer CloudWatch les journaux

Le SDK Hub fournit des fonctionnalités de journalisation complètes. Par défaut, le SDK du Hub écrit les journaux dans le système de fichiers local. Vous pouvez toutefois tirer parti de l'API cloud pour configurer le streaming CloudWatch des journaux vers Logs, qui propose :

- Surveillez les performances des appareils : enregistrez des journaux d'exécution détaillés pour une gestion proactive des appareils. Activez une analyse et une surveillance avancées des journaux sur l'ensemble de votre parc d'appareils
- Résoudre les problèmes : générez des entrées de journal détaillées pour une analyse diagnostique rapide. Enregistrez les événements au niveau du système et de l'application pour une enquête approfondie.
- Journalisation flexible et centralisée : gestion des journaux à distance sans accès direct aux appareils. Regroupez les journaux de plusieurs appareils dans un seul référentiel consultable.

Prérequis

- Intégrez l'appareil géré dans le cloud. Consultez [Configuration de l'intégration du hub](#) pour plus de détails.
- Vérifiez le démarrage et la réussite de l'initialisation de l'agent Hub. Consultez [Installation et validation du SDK Hub pour les intégrations gérées](#) pour plus de détails.

 Note

Pour créer des configurations de journalisation, consultez [PutRuntimeLogConfiguration l'API](#) pour plus de détails.

 Warning

L'activation des journaux est prise en compte dans le calcul des quotas par niveaux.
L'augmentation des niveaux de journalisation se traduira par une augmentation du volume de messages et des coûts supplémentaires.

Configuration des configurations du journal du SDK Hub

Configurez les paramètres du journal du SDK du hub en appelant l'API pour configurer la configuration du journal d'exécution.

Exemple exemple de demande d'API

```
aws iot-managed-integrations put-runtime-log-configuration \  
  --managed-thing-id MANAGED_THING_ID \  
  --runtime-log-configurations LogLevel=DEBUG,UploadLog=TRUE
```

RuntimeLogConfigurations attributs

Les attributs suivants sont facultatifs et peuvent être configurés dans l'RuntimeLogConfigurationsAPI.

LogLevel

Définit le niveau de gravité minimal pour les traces d'exécution. Valeurs: DEBUG, ERROR, INFO, WARN

Par défaut : WARN (version publiée)

LogFlushLevel

Détermine le niveau de gravité pour le transfert immédiat des données vers le stockage local.
Valeurs: DEBUG, ERROR, INFO, WARN

Par défaut : DISABLED

LocalStoreLocation

Spécifie l'emplacement de stockage pour les traces d'exécution. Par défaut : `/var/log/awsiotmi`

- Journal actif : `/var/log/awsiotmi/ManagedIntegrationsDeviceSdkHub.log`
- Journaux pivotés : `/var/log/awsiotmi/ManagedIntegrationsDeviceSdkHub.N.log`
(N indique l'ordre de rotation)

LocalStoreFileRotationMaxBytes

Déclenche la rotation du fichier lorsque le fichier actuel dépasse la taille spécifiée.

Important

Pour une efficacité optimale, maintenez la taille du fichier en dessous de 125 Ko. Les valeurs supérieures à 125 Ko seront automatiquement limitées.

LocalStoreFileRotationMaxFiles,

Définit le nombre maximum de fichiers de rotation autorisés par le démon log.

UploadLog

Contrôle le transfert des traces d'exécution vers le cloud. Les journaux sont stockés dans le groupe `/aws/iotmanagedintegration CloudWatch Logs`.

Par défaut: `false`.

UploadPeriodMinutes

Définit la fréquence des téléchargements des traces d'exécution. Par défaut : 5

DeleteLocalStoreAfterUpload

Contrôle la suppression des fichiers après le téléchargement. Par défaut : `true`

Note

S'il est défini sur `false`, les fichiers téléchargés sont renommés en : `/var/log/awsiotmi/ManagedIntegrationsDeviceSdkHub.uploaded.{uploaded_timestamp}`

Exemple de fichier journal

Voir l'exemple d'un fichier CloudWatch journal ci-dessous :

Types d'appareils Zigbee et Z-Wave pris en charge

Cette page répertorie les types d'appareils connectés au hub qui ont été testés avec des intégrations gérées et qui sont pris en charge. Les intégrations gérées prennent en charge les deux [Configuration simple \(SS\)](#) et [Configuration guidée par l'utilisateur \(UGS\)](#) pour ces appareils.

Ce tableau répertorie les appareils Zigbee pris en charge.

Type d'appareil Zigbee	Fonctionnalités prises en charge
Ampoule intelligente/Lumière à intensité variable/Lumière RGB	OnOff, LevelControl, ColorControl
Prise intelligente	OnOff
Interrupteur intelligent	OnOff
bande LED	OnOff, LevelControl, ColorControl
Valve d'eau	OnOff
Vanne de radiateur	Thermostat OnOff, minuterie
Thermostat	Thermostat FanControl OnOff, minuterie
Ouvre-porte de garage	WindowCovering, OnOff, LevelControl
Détecteur de fumée	BooleanState,, OnOff, Minuteur TemperatureMeasurement, Fumée COAlarm
Capteur de mouvement	BooleanState
Détecteur de présence humaine ou d'occupation	BooleanState, OccupancySensing
Détecteur de porte et de fenêtre	BooleanState

Type d'appareil Zigbee	Fonctionnalités prises en charge
Détecteur de fuite d'eau	BooleanState
Capteur de vibrations	BooleanState
Capteur de température et d'humidité	TemperatureMeasurement, RelativeHumidityMeasurement

Ce tableau répertorie les appareils Z-Wave pris en charge.

Type d'appareil Z-Wave	Fonctionnalités prises en charge
Ampoule intelligente/lumière à intensité variable	OnOff, LevelControl
Prise intelligente	OnOff
Contrôleur de porte de garage	OnOff, LevelControl
Compteur d'énergie	ElectricalEnergyMeasurement, ElectricalPowerMeasurement
Batterie	LevelControl
Sirène	LevelControl
Capteur de mouvement	BooleanState
Détecteur de porte et de fenêtre	BooleanState
Détecteur de fuite d'eau	BooleanState
Sonde de température	TemperatureMeasurement
Détecteur de CO	Fumée COAlarm
Détecteur de fumée	Fumée COAlarm

Exécutez des intégrations gérées sur Raspberry Pi

Note

Cette implémentation du SDK AWS IoT Hub sur Raspberry Pi est un projet de démonstration destiné uniquement à des fins d'apprentissage et de test et n'est pas destiné à être utilisé dans des environnements de production. Dans le cadre de cette démonstration, définissez les configurations suivantes pour faciliter le développement :

AWS stockage des informations d'identification : à des fins de démonstration uniquement, les informations d'identification et les certificats sont stockés dans un endroit accessible pour faciliter les tests et le développement. Les environnements de production doivent utiliser des solutions de stockage sécurisées telles que AWS Secrets Manager Systems Manager Parameter Store. Ils doivent implémenter le chiffrement au repos et suivre les consignes AWS IoT de sécurité.

Privilèges de conteneur : la démo s'exécute avec des privilèges élevés afin de permettre un accès illimité aux ressources de l'hôte et de simplifier les flux de développement. En production, les conteneurs doivent fonctionner avec les privilèges minimaux requis.

Configuration du pont réseau : La démonstration utilise une configuration de pont réseau qui expose le trafic réseau interne pour faciliter le débogage et la surveillance. Dans les environnements de production, mettez en œuvre une isolation et une segmentation appropriées du réseau pour empêcher tout accès non autorisé au trafic réseau interne.

Autorisations des périphériques USB : l'accès illimité aux périphériques USB est activé pour faciliter la connexion des périphériques de développement et des appareils de test. Pour la production, mettez en œuvre des contrôles et une validation stricts des périphériques USB afin de prévenir les attaques par usurpation d'appareil.

Ces configurations permettent des tests simples et ne doivent pas être utilisées dans des environnements de production. Lors du déploiement en production, veuillez suivre les meilleures pratiques en matière de sécurité afin d'éviter toute compromission du système hôte et tout accès non autorisé aux informations d'identification.

Comme condition préalable, vous devez configurer le dongle USB Sonoff Zigbee avant de configurer le Raspberry Pi.

Flasher le firmware sur le dongle USB Sonoff Zigbee

Prérequis

- [Dongle USB Sonoff Zigbee](#)
- Windows : installez le pilote [Windows universel CP21 0x](#)

Flasher le firmware

1. Téléchargez la [version 7.4.1.0 du micrologiciel du dongle Zigbee](#).
2. Ouvrez le logiciel [Silabs Microware Flasher](#).
3. Connectez le dongle USB Sonoff Zigbee à votre ordinateur.
4. Faites défiler la page et trouvez ZBDongle-E.
5. Choisissez Se connecter.
6. Attendez que l'appareil se connecte.
7. Choisissez Modifier le microprogramme.
8. Sélectionnez Téléverser votre propre microprogramme.
9. Trouvez l'emplacement du téléchargement du [firmware Zigbee Dongle Build 7.4.1.0](#) et sélectionnez-le.
10. Cliquez sur Installer.
11. Attendez que le microprogramme soit installé.
12. Choisissez Continuer lorsque l'installation est terminée.

Le dongle est maintenant prêt à être utilisé.

Choisissez l'une des options répertoriées ci-dessous pour exécuter le SDK du hub d'intégrations gérées sur votre Raspberry Pi. Les étapes de configuration et de validation pour les deux approches sont répertoriées ci-dessous.

Rubriques

- [Image du SDK Hub d'intégrations gérées sur Raspberry Pi](#)

- [Intégrations gérées Hub SDK, conteneur Docker sur Raspberry Pi](#)
- [Application de démonstration d'intégrations gérées](#)

Image du SDK Hub d'intégrations gérées sur Raspberry Pi

Note

Cette implémentation du SDK AWS IoT Hub sur Raspberry Pi est un projet de démonstration destiné uniquement à des fins d'apprentissage et de test et n'est pas destiné à être utilisé dans des environnements de production. Dans le cadre de cette démonstration, définissez les configurations suivantes pour faciliter le développement :

AWS stockage des informations d'identification : à des fins de démonstration uniquement, les informations d'identification et les certificats sont stockés dans un endroit accessible pour faciliter les tests et le développement. Les environnements de production doivent utiliser des solutions de stockage sécurisées telles que AWS Secrets Manager Systems Manager Parameter Store. Ils doivent implémenter le chiffrement au repos et suivre les consignes AWS IoT de sécurité.

Privilèges de conteneur : la démo s'exécute avec des privilèges élevés afin de permettre un accès illimité aux ressources de l'hôte et de simplifier les flux de développement. En production, les conteneurs doivent fonctionner avec les privilèges minimaux requis.

Configuration du pont réseau : La démonstration utilise une configuration de pont réseau qui expose le trafic réseau interne pour faciliter le débogage et la surveillance. Dans les environnements de production, mettez en œuvre une isolation et une segmentation appropriées du réseau pour empêcher tout accès non autorisé au trafic réseau interne.

Autorisations des périphériques USB : l'accès illimité aux périphériques USB est activé pour faciliter la connexion des périphériques de développement et des appareils de test. Pour la production, mettez en œuvre des contrôles et une validation stricts des périphériques USB afin de prévenir les attaques par usurpation d'appareil.

Ces configurations permettent des tests simples et ne doivent pas être utilisées dans des environnements de production. Lors du déploiement en production, veuillez suivre les meilleures pratiques en matière de sécurité afin d'éviter toute compromission du système hôte et tout accès non autorisé aux informations d'identification.

Prérequis

Répondez aux exigences suivantes avant de déployer l'image du Raspberry Pi :

- Téléchargez et installez l'[imageur Raspberry Pi](#).
- Procurez-vous une [carte SD](#).
- Configurez un [Raspberry Pi 5 avec un processeur quadricœur 64 bits 2,4 GHz \(8 Go de RAM\)](#).
- Connectez un dongle [USB Sonoff Zigbee](#).
- [Flasher le firmware sur le dongle USB Sonoff Zigbee](#).
- Connectez un dongle [Silicon Labs SLUSB001A](#).
- [Ouvrez un AWS compte](#).
- Installez la dernière version de [AWS CLI depuis le manuel Managed Integrations AWS CLI Command Reference](#).

Flasher une image du Raspberry Pi sur une nouvelle carte SD

Flashez l'image des intégrations gérées sur votre carte SD en procédant comme suit :

1. Téléchargez l'image du [SDK Raspberry Pi Hub pour les intégrations gérées](#).
2. Lancez Raspberry Pi Imager sur votre bureau.
3. Insérez la carte SD dans le lecteur de carte SD intégré à votre ordinateur ou dans un lecteur de carte USB externe.
4. Sélectionnez Choisir un appareil → Raspberry Pi 5.
5. Sélectionnez Choisir le système d'exploitation → Utiliser la personnalisation → Rechercher le fichier `IoTMI-HubSDK-RPi-Image -v1.0.0.img.gz` → Ouvrir.
6. Sélectionnez Choisir le stockage → Sélectionnez votre lecteur de carte SD.
7. Vérifiez que votre configuration correspond à l'écran suivant :
8. Cliquez sur Next (Suivant).
9. Configurez les paramètres de personnalisation du système d'exploitation :
 - Nom d'hôte : sélectionnez raspberrypi.
 - Nom d'utilisateur et mot de passe :
 - Activez Définir le nom d'utilisateur et le mot de passe :
 - Pour Nom d'utilisateur :, entrez `hub123456`.
 - Pour le mot de passe :, entrez `sh123456`.
 - Réseau local sans fil :

- Activez Configurer le réseau sans fil.
- Entrez le SSID et le mot de passe de votre routeur.

Exemples de paramètres :

- SSID : `iotmi-tplink`
- Mot de passe : `*****` (8 caractères minimum)
- Définir le pays : àUS.
- Définissez les paramètres régionaux :
 - Régler le fuseau horaire : àAmerica/Los Angeles.
 - Régler la disposition du clavier : surUS.
- SSH :
 - Choisissez l'onglet Services.
 - Cochez Activer SSH.
 - Choisissez Utiliser l'authentification par mot de passe.

10. Confirmez toutes les fenêtres contextuelles pour la personnalisation du système d'exploitation et l'effacement des données.

11. Attendez que le processus d'écriture soit terminé.

12. Vérifiez que l'opération est terminée avec succès à l'aide de l'écran suivant :

13. Cliquez sur Continuer.

14. Retirez la carte SD et insérez-la dans votre Raspberry Pi.

Exécutez le SDK Hub sur le Raspberry Pi

Démarrez les services du SDK Hub sur votre Raspberry Pi configuré :

1. Insérez la carte SD préparée dans l'appareil Raspberry Pi 5.
2. Connectez le dongle USB Sonoff Zigbee et le dongle Silicon Labs SLUSB001A au Raspberry Pi.
3. Allumez le Raspberry Pi.
4. Assurez-vous que le Raspberry Pi et votre ordinateur (à partir duquel vous utilisez SSH) sont sur le même réseau.
5. Connectez-vous au Raspberry Pi en utilisant les informations d'identification que vous avez définies lors du déploiement de l'image.

```
ssh username@hostname
```

6. Accédez au répertoire du SDK du hub :

```
cd /data/aws/iotmi
```

7. Terminez la [configuration d'intégration du Hub](#) pour ajouter du matériel d'authentification et de configuration.

Note

Vous devez être connecté à YUL ou à une DUB région pour effectuer cette étape.

8. Exécutez le SDK du Hub :

```
cd /data/aws/iotmi
bash start_hub_sdk.sh
```

Le système affiche la réponse suivante pour un démarrage réussi du SDK Hub :

```
-----Stopping SDK running processes---
-----Starting Hub SDK-----
-----Creating logs directory-----
Logs directory created.
-----Verifying Middleware paths-----
All middleware libraries exist
-----Verifying Middleware pre reqs---
AIPC and KVstroage directories exist
-----Starting HubOnboarding-----
-----Starting MQTT Proxy-----
-----Staring Log Daemon---
-----Starting Event Manager-----
-----Starting Zigbee Service-----
--Checking Zigbee network information--
-----Starting Zwave Service-----
/data/aws/iotmi/middleware/AceZwave/bin /data/aws/iotmi
/data/aws/iotmi
-----Starting CDMB-----
-----Starting Agent-----
-----Starting Provisioner-----
```

```

-----Checking SDK status-----
hub1234+    1780  0.2  0.1 1093936 16368 pts/1    Sl+  16:34    0:00 ./iotmi_mqtt_proxy -
C /data/aws/iotmi/config/iotmi_config.json
Process 'iotmi_mqtt_proxy' is running.
hub1234+    1884  0.0  0.0 236272  2624 pts/1    Sl+  16:34    0:00 ./middleware/
AceCommon/bin/ace_eventmgr
Process 'ace_eventmgr' is running.
hub1234+    1892  9.1  0.1 393040  8352 pts/1    Sl+  16:34    0:04 ./middleware/
AceZigbee/bin/ace_zigbee_service
Process 'ace_zigbee_service' is running.
hub1234+    1923  0.0  0.1 1570736 12736 pts/1    Sl+  16:34    0:00 ./zwave_svc
Process 'zwave_svc' is running.
hub1234+    1958  0.0  0.0 1067632 5776 pts/1    Sl+  16:34    0:00 ./iotmi_cdmdb
Process 'iotmi_cdmdb' is running.
hub1234+    2001  0.2  0.2 2017712 21264 pts/1    Sl+  16:35    0:00 ./iotmi_device_agent
Process 'iotmi_device_agent' is running.
hub1234+    2045  0.0  0.1 1457824 12624 pts/1    Sl+  16:35    0:00 ./
iotmi_lpw_provisioner
Process 'iotmi_lpw_provisioner' is running.
hub1234+    1813  0.0  0.0 875152  6848 pts/1    Sl+  16:34    0:00 ./iotmi_log_daemon
Process 'iotmi_log_daemon' is running.
-----Successfully Started Hub SDK-----

```

Étapes suivantes

Après avoir démarré avec succès le SDK du Hub, procédez à l'intégration et à la gestion des appareils à l'adresse. [Configuration guidée par l'utilisateur pour embarquer et faire fonctionner les appareils](#)

Intégrations gérées Hub SDK, conteneur Docker sur Raspberry Pi

Note

Cette implémentation du SDK AWS IoT Hub sur Raspberry Pi est un projet de démonstration destiné uniquement à des fins d'apprentissage et de test et n'est pas destiné à être utilisé dans des environnements de production. Dans le cadre de cette démonstration, définissez les configurations suivantes pour faciliter le développement :

AWS stockage des informations d'identification : à des fins de démonstration uniquement, les informations d'identification et les certificats sont stockés dans un endroit accessible pour faciliter les tests et le développement. Les environnements de production doivent utiliser des solutions de stockage sécurisées telles que AWS Secrets Manager Systems Manager

Parameter Store. Ils doivent implémenter le chiffrement au repos et suivre les consignes AWS IoT de sécurité.

Privilèges de conteneur : la démo s'exécute avec des privilèges élevés afin de permettre un accès illimité aux ressources de l'hôte et de simplifier les flux de développement. En production, les conteneurs doivent fonctionner avec les privilèges minimaux requis.

Configuration du pont réseau : La démonstration utilise une configuration de pont réseau qui expose le trafic réseau interne pour faciliter le débogage et la surveillance. Dans les environnements de production, mettez en œuvre une isolation et une segmentation appropriées du réseau pour empêcher tout accès non autorisé au trafic réseau interne.

Autorisations des périphériques USB : l'accès illimité aux périphériques USB est activé pour faciliter la connexion des périphériques de développement et des appareils de test. Pour la production, mettez en œuvre des contrôles et une validation stricts des périphériques USB afin de prévenir les attaques par usurpation d'appareil.

Ces configurations permettent des tests simples et ne doivent pas être utilisées dans des environnements de production. Lors du déploiement en production, veuillez suivre les meilleures pratiques en matière de sécurité afin d'éviter toute compromission du système hôte et tout accès non autorisé aux informations d'identification.

Prérequis

Les conditions préalables suivantes sont requises pour le conteneur docker.

- Téléchargez et installez l'[imageur Raspberry Pi](#).
- Procurez-vous une [carte SD](#).
- Configurez un [Raspberry Pi 5 avec un processeur quadricœur 64 bits 2,4 GHz \(8 Go de RAM\)](#).
- Connectez un dongle [USB Sonoff Zigbee](#).
- [Flasher le firmware sur le dongle USB Sonoff Zigbee](#).
- Connectez un dongle [Silicon Labs SLUSB001A](#).
- [Ouvrez un AWS compte](#).
- Installez la dernière version de [AWS CLI depuis le manuel Managed Integrations AWS CLI Command Reference](#).
- Accès SSH au Raspberry Pi avec adresse IP ou nom d'hôte.

Utiliser le conteneur Docker du SDK Hub des intégrations gérées sur Raspberry Pi

1. Téléchargez les [intégrations gérées Raspberry Pi Hub SDK Docker](#).

2. Copiez le fichier sur le Raspberry Pi en utilisant SCP :

```
scp ~/path/to/IotMI-HubSDK-Docker-v1.0.0.tar.gz [username]@raspberrypi.local:~
```

3. Connectez-vous au Raspberry Pi via SSH :

```
ssh hub123456@raspberrypi.local
```

4. Installez Docker s'il n'est pas présent :

```
# Install Docker
cd
curl -fsSL https://get.docker.com | sudo sh

# Add your user to docker group
sudo usermod -aG docker $USER
exit # exit ssh

# Log in again
```

5. Installez Docker Compose s'il n'est pas présent :

```
# Install Docker Compose
sudo apt-get update
sudo apt-get install -y docker-compose-plugin
```

6. Extrayez les fichiers du SDK du Hub :

```
# Navigate to the home directory
cd

# Extract the hub-docker.tar.gz file
tar -xzf IotMI-HubSDK-Docker-v1.0.0.tar.gz
```

7. Accédez au répertoire hub-docker :

```
cd IotMI-HubSDK-Docker
```

8. Complétez la [configuration d'intégration du Hub](#) pour configurer l'authentification et les paramètres.

 Note

Vous devez être connecté à YUL ou à une DUB région pour effectuer cette étape.

9. Démarrez le conteneur Docker :

```
# The first time it's called, it will build the container
docker compose up -d
docker compose logs -f
```

Résultat attendu :

[illegible]

[illegible]

Après avoir démarré avec succès le SDK du Hub, procédez à l'intégration et à la gestion des appareils à l'adresse. [Configuration guidée par l'utilisateur pour embarquer et faire fonctionner les appareils](#)

 Note

- Pour accéder au shell bash du conteneur Docker, exécutez la commande suivante :

```
docker compose exec hubsdk bash
```

- Pour redémarrer le conteneur après le redémarrage, exécutez la commande suivante :

```
docker compose up -d
```

- Pour mettre à jour le SDK du Hub, remplacez les fichiers binaires dans le dossier suivant :

```
hub-docker/iotmi
```

- Pour redémarrer le conteneur en toute sécurité tout en préservant les données, procédez comme suit :

```
docker compose down  
docker compose up -d  
docker compose logs -f
```

Application de démonstration d'intégrations gérées

Note

Cette implémentation du SDK AWS IoT Hub sur Raspberry Pi est un projet de démonstration destiné uniquement à des fins d'apprentissage et de test et n'est pas destiné à être utilisé dans des environnements de production. Dans le cadre de cette démonstration, définissez les configurations suivantes pour faciliter le développement :

AWS stockage des informations d'identification : à des fins de démonstration uniquement, les informations d'identification et les certificats sont stockés dans un endroit accessible pour faciliter les tests et le développement. Les environnements de production doivent utiliser des solutions de stockage sécurisées telles que AWS Secrets Manager Systems Manager Parameter Store. Ils doivent implémenter le chiffrement au repos et suivre les consignes AWS IoT de sécurité.

Privilèges de conteneur : la démo s'exécute avec des privilèges élevés afin de permettre un accès illimité aux ressources de l'hôte et de simplifier les flux de développement. En production, les conteneurs doivent fonctionner avec les privilèges minimaux requis.

Configuration du pont réseau : La démonstration utilise une configuration de pont réseau qui expose le trafic réseau interne pour faciliter le débogage et la surveillance. Dans les environnements de production, mettez en œuvre une isolation et une segmentation appropriées du réseau pour empêcher tout accès non autorisé au trafic réseau interne.

Autorisations des périphériques USB : l'accès illimité aux périphériques USB est activé pour faciliter la connexion des périphériques de développement et des appareils de test. Pour la production, mettez en œuvre des contrôles et une validation stricts des périphériques USB afin de prévenir les attaques par usurpation d'appareil.

Ces configurations permettent des tests simples et ne doivent pas être utilisées dans des environnements de production. Lors du déploiement en production, veuillez suivre les meilleures pratiques en matière de sécurité afin d'éviter toute compromission du système hôte et tout accès non autorisé aux informations d'identification.

L'application de démonstration est une application de démonstration basée sur React présentant les capacités d'intégration gérées pour la gestion des appareils domestiques intelligents. Cette application explique l'intégration, le contrôle et la surveillance des appareils Z-Wave et Zigbee via une interface Web moderne.

Prérequis

- [Ouvrez un AWS compte.](#)
- [Créez un casier d'identifiants et ajoutez-le à votre hub.](#)
- [Configuration complète de l'intégration au Hub.](#)
- [Node.js 18+ et npm.](#)
- Installez la dernière version de [AWS CLI depuis le manuel Managed Integrations AWS CLI Command Reference.](#)
- Navigateur Web moderne (Chrome, Firefox, Safari, Edge)

Installation et configuration de l'application

1. Téléchargez l'[application de démonstration des intégrations gérées](#).
2. Extrayez le package :

```
cd ~/Downloads
tar -xzf IotMI-HubSDK-DemoApp-v1.0.0.tar.gz
cd IotManagedIntegrations-DemoApp
```

3. Installez les dépendances:

```
npm install
```

4. Créez un `.env` fichier dans le répertoire racine :

```
# AWS Configuration
```

```
REACT_APP_AWS_REGION=your_region
REACT_APP_AWS_ACCESS_KEY_ID=your_access_key
REACT_APP_AWS_SECRET_ACCESS_KEY=your_secret_key
REACT_APP_AWS_SESSION_TOKEN=your_session_token

# IoT Managed Integrations Endpoint
REACT_APP_IOT_ENDPOINT=https://your-iot-endpoint.amazonaws.com

# Hub Configuration
REACT_APP_HUB_MANAGED_THING_ID=your_hub_id
REACT_APP_CREDENTIAL_LOCKER_ID=your_credential_locker_id
```

5. Créez et démarrez l'application :

```
npm start
```

6. Accédez à l'application à l'adresse suivante :

```
http://localhost:3000
```

Pour plus d'informations sur les tarifs, reportez-vous à la [section Intégrations gérées sur la page de tarification de la gestion des AWS IoT appareils](#).

Hub d'intégrations gérées hors bord

Présentation du processus de déconnexion du SDK Hub

Le processus de désintégration du hub supprime un hub du système de AWS Cloud gestion. Lorsque le cloud envoie une [DeleteManagedThing](#) demande, le processus atteint deux objectifs principaux :

Actions côté appareil :

- Réinitialisez l'état interne du hub
- Supprimer toutes les données enregistrées localement
- Préparez l'appareil en vue d'un éventuel futur réembarquement

Actions côté cloud :

- Supprimer toutes les ressources cloud associées au hub

- Déconnexion complète du compte précédent

Les clients commencent généralement à quitter le hub lorsque :

- Modifier le compte associé au hub
- Remplacement d'un hub existant par un nouvel appareil

Le processus garantit une transition propre et sécurisée entre les configurations du hub, permettant une gestion fluide des appareils et une flexibilité des comptes.

Prérequis

- Vous devez disposer d'un hub intégré. Pour obtenir des instructions, consultez la section [Configuration de l'intégration au Hub](#).
- Dans le `iotmi_config.json` fichier situé à l'adresse `/data/aws/iotmi/config/`, vérifiez que cela s'`iot_provisioning_state` affiche `PROVISIONED`.
- Vérifiez que les certificats permanents et les clés auxquels il est fait référence `iotmi_config.json` existent dans les chemins spécifiés.
- Assurez-vous que l'agent HubOnboarding, le provisionneur et le proxy MQTT sont correctement configurés et exécutés.
- Vérifiez qu'aucun appareil enfant n'est installé sur le hub. Utilisez l' [DeleteManagedThing](#) API pour supprimer tous les appareils enfants avant de continuer.

Processus de déconnexion du SDK Hub

Pour quitter le hub, procédez comme suit :

Récupère l'ID `hub_managed_thing`

Le `iotmi_config.json` fichier est utilisé pour stocker l'ID d'objet géré pour un hub d'intégrations gérées. Cet identifiant est une information essentielle qui permet au hub de communiquer avec le service d'intégrations AWS IoT gérées. L'ID d'objet géré est stocké dans la section `rw` (lecture-écriture) du fichier JSON, sous le champ `managed_thing_id` Cela se voit dans l'exemple de configuration suivant :

```
{
  "ro": {
    "iot_provisioning_method": "FLEET_PROVISIONING",
    "iot_claim_cert_path": "PATH",
    "iot_claim_pk_path": "PATH",
    "UPC": "UPC",
    "sh_endpoint_url": "ENDPOINT_URL",
    "SN": "SN",
    "fp_template_name": "TEMPLATENAME"
  },
  "rw": {
    "iot_provisioning_state": "PROVISIONED",
    "client_id": "ID",
    "managed_thing_id": "ID",
    "iot_permanent_cert_path": "CERT_PATH",
    "iot_permanent_pk_path": "KEY",
    "metadata": {
      "last_updated_epoch_time": 1747766125
    }
  }
}
```

Envoyer la commande au hub externe

Utilisez les informations d'identification de votre compte et exécutez la commande avec les informations `managed_thing_id` récupérées dans la section précédente :

```
aws iot-managed-integrations delete-managed-thing \
  --identifiant HUB_MANAGED_THING_ID
```

Vérifiez que le hub a été déconnecté

Utilisez les informations d'identification de votre compte et exécutez la commande avec les informations `managed_thing_id` récupérées dans la section précédente :

```
aws iot-managed-integrations get-managed-thing \
  --identifiant HUB_MANAGED_THING_ID
```

Scénarios de réussite et d'échec

Scénario de réussite

Si la commande de déconnexion du hub a été exécutée avec succès, l'exemple de réponse suivant est attendu :

```
{
  "Message" : "Managed Thing resource not found."
}
```

En outre, l'exemple suivant `iotmi_config.json` serait observé si la commande de déchargement du hub était réussie. Vérifiez que la section `rw` contient uniquement `iot_provisioning_state` et éventuellement des métadonnées. L'absence de métadonnées est acceptable. `iot_provisioning_state` doit être `NOT_PROVISIONED`.

```
{
  "ro": {
    "iot_provisioning_method": "FLEET_PROVISIONING",
    "iot_claim_cert_path": "PATH",
    "iot_claim_pk_path": "PATH",
    "UPC": "1234567890101",
    "sh_endpoint_url": "ENDPOINT_URL",
    "SN": "1234567890101",
    "fp_template_name": "test-template"
  },
  "rw": {
    "iot_provisioning_state": "NOT_PROVISIONED",
    "metadata": {
      "last_updated_epoch_time": 1747766125
    }
  }
}
```

Scénario de défaillance

Si la commande de déconnexion du hub a échoué, l'exemple de réponse suivant est attendu :

```
{
  "Arn" : "ARN",
  "CreatedAt" : 1.748968266655E9,
```



```

    "Id" : "ID",
    "ProvisioningStatus" : "DELETE_IN_PROGRESS",
    "Role" : "CONTROLLER",
    "SerialNumber" : "SERIAL_NO",
    "Tags" : { },
    "UniversalProductCode" : "UPC",
    "UpdatedAt" : 1.748968272107E9
  }

```

- Si tel ProvisioningStatus est le cas DELETE_IN_PROGRESS, suivez les instructions de la section de [restauration du Hub](#).
- Si ProvisioningStatus n'est pas le cas DELETE_IN_PROGRESS, la commande de déconnexion du hub a échoué dans le cloud d'intégrations gérées ou n'a pas été reçue par le cloud d'intégrations gérées. Suivez les instructions de la [section Hub Recovery](#).
- Si le téléchargement a échoué, votre `iotmi_config.json` fichier ressemblera au fichier d'exemple ci-dessous.

```

{
  "ro": {
    "iot_provisioning_method": "FLEET_PROVISIONING",
    "iot_claim_cert_path": "PATH",
    "iot_claim_pk_path": "PATH",
    "UPC": "123456789101",
    "sh_endpoint_url": "ENDPOINT_URL",
    "SN": "123456789101",
    "fp_template_name": "test-template"
  },
  "rw": {
    "iot_provisioning_state": "PROVISIONED",
    "client_id": "ID",
    "managed_thing_id": "ID",
    "iot_permanent_cert_path": "PATH",
    "iot_permanent_pk_path": "PATH",
    "metadata": {
      "last_updated_epoch_time": 1747766125
    }
  }
}

```

(Facultatif) Après avoir quitté le SDK Hub

Important

Les scénarios suivants répertorient les actions facultatives à effectuer en cas d'échec du SDK du Hub ou si vous souhaitez réintégrer votre hub après le déchargement.

Réembarquement

Si l'offboarding a été un succès, intégrez votre SDK Hub en suivant l'[étape 3 : créer un objet géré \(approvisionnement de flotte\)](#) et suivre le reste du processus d'intégration.

Restauration du hub

Succès de l'offboarding du Device Hub et échec de l'offboarding du Cloud

Si [GetManagedThing](#) l'appel d'API ne renvoie pas de Managed Thing resource not found message, mais que le fichier `iotmi_config.json` est déconnecté. Voir [Scénario de réussite](#) pour un exemple de fichier json.

Pour sortir de ce scénario, voir [Forcer la suppression](#).

Le déchargement du hub de périphériques échoue

Ce scénario se produit lorsque le fichier `n'iotmi_config.json` est pas correctement déconnecté. Voir [Scénario d'échec](#) pour un exemple de fichier JSON.

Pour sortir de ce scénario, voir [Forcer la suppression](#). S'il `n'iotmi_config.json` est toujours pas déconnecté, le hub doit être réinitialisé aux paramètres d'usine.

Le déchargement du hub d'appareils et le déchargement du cloud échouent

Dans ce scénario, `n'iotmi_config.json` est toujours pas déconnecté et le statut du hub est soit `ACTIVATED`, soit `DISCOVERED`

Pour sortir de ce scénario, voir [Forcer la suppression](#). Si la suppression forcée échoue ou `n'iotmi_config.json` est toujours pas déconnectée, le hub doit être réinitialisé aux paramètres d'usine.

Le hub est hors ligne et son statut est `DELETE_IN_PROGRESS`

Dans ce scénario, le hub est hors ligne et le cloud reçoit une commande de déconnexion.

Pour sortir de ce scénario, consultez la section [Forcer la suppression](#).

Forcer la suppression

Pour supprimer des ressources cloud sans que le terminal n'ait été déconnecté correctement, procédez comme suit. Cette opération peut entraîner une incohérence entre l'état du cloud et celui de l'appareil, ce qui peut entraîner des problèmes lors des opérations futures.

Appelez [DeleteManagedThing](#) l'API avec le paramètre du hub `managed_thing_id` et le paramètre `force` :

```
aws iot-managed-integrations delete-managed-thing \  
  --identifiant HUB_MANAGED_THING_ID \  
  --force
```

Ensuite, appelez l' [GetManagedThing](#) API et vérifiez qu'elle est renvoyée `Managed Thing resource not found`. Cela confirme que les ressources du cloud sont supprimées.

Note

Cette approche n'est pas recommandée, car elle peut entraîner des incohérences entre l'état du cloud et celui de l'appareil. Il est généralement préférable de s'assurer que le terminal a été déconnecté correctement avant de tenter de supprimer les ressources du cloud.

Intergiciel spécifique au protocole

Important

La documentation et le code fournis ici décrivent une implémentation de référence du middleware. Il ne vous est pas fourni dans le cadre du SDK.

L'intergiciel spécifique au protocole joue un rôle essentiel dans l'interaction avec les piles de protocoles sous-jacentes. Les composants d'intégration et de contrôle des appareils du SDK Managed Integrations Hub l'utilisent pour interagir avec l'appareil final.

Le middleware exécute les fonctions suivantes.

- Extrait les piles de protocoles APIs de l'appareil provenant de différents fournisseurs en fournissant un ensemble commun de APIs
- Assure la gestion de l'exécution des logiciels, notamment le planificateur de threads, la gestion des files d'événements et le cache de données.

Architecture intergicielle

Le schéma fonctionnel ci-dessous représente l'architecture du middleware Zigbee. L'architecture des intergiciels d'autres protocoles tels que Z-Wave est également similaire.

L'intergiciel spécifique au protocole comporte trois composants principaux.

- ACS Zigbee DPK : Le kit de portage de périphériques (DPK) Zigbee est utilisé pour fournir une abstraction du matériel et du système d'exploitation sous-jacents, permettant ainsi la portabilité. En gros, cela peut être considéré comme la couche d'abstraction matérielle (HAL), qui fournit un ensemble commun APIs pour contrôler et communiquer avec les radios Zigbee de différents fournisseurs. Le middleware Zigbee contient l'implémentation de l'API DPK pour le framework d'applications Zigbee de Silicon Labs.
- Service ACS Zigbee : Le service Zigbee fonctionne comme un démon dédié. Il inclut un gestionnaire d'API servant les appels d'API des applications clientes via les canaux IPC. L'AIPC est utilisé comme canal IPC entre l'adaptateur Zigbee et le service Zigbee. Il fournit d'autres fonctionnalités telles que la gestion des deux async/sync commandes, la gestion des événements de la HAL et l'utilisation d'ACS Event Manager pour l'enregistrement/la publication d'événements.
- Adaptateur Zigbee ACS : L'adaptateur Zigbee est une bibliothèque exécutée dans le cadre du processus d'application (dans ce cas, l'application est le plugin CDMB). L'adaptateur Zigbee fournit un ensemble de APIs composants utilisés par les applications clientes, tels que des plugins de CDMB/Provisioner protocole pour contrôler et communiquer avec l'appareil final.

End-to-end exemple de flux de commande intergiciel

Voici un exemple du flux de commandes via le middleware Zigbee.

Voici un exemple du flux de commandes via le middleware Z-Wave.

Organisation du code intergiciel spécifique au protocole

Cette section contient des informations sur l'emplacement du code de chaque composant dans le IotManagedIntegrationsDeviceSDK-Middleware référentiel. Voici un exemple de la structure de dossiers de ce référentiel.

```
./IotManagedIntegrationsDeviceSDK-Middleware
|- greengrass
|- example-iot-ace-dpk
|- example-iot-ace-general
|- example-iot-ace-project
|- example-iot-ace-z3-gateway
|- example-iot-ace-zware
|- example-iot-ace-zwave-mw
```

Rubriques

- [Organisation du code du middleware Zigbee](#)
- [Organisation du code du middleware Z-Wave](#)

Organisation du code du middleware Zigbee

Ce qui suit montre l'organisation du code du middleware de référence Zigbee.

Rubriques

- [ACS Zigbee DPK](#)
- [Kit de développement logiciel Zigbee de Silicon Labs](#)
- [Service ACS Zigbee](#)
- [Adaptateur ACS Zigbee](#)

ACS Zigbee DPK

Le code de Zigbee DPK se trouve dans le répertoire indiqué dans l'exemple ci-dessous :

```
./IotManagedIntegrationsDeviceSDK-Middleware/example-iot-ace-dpk/example/dpk/ace_hal/
|- common
|-   |- fxnDBusClient
```

```

|-   |- include
|- kvs
|- log
|- wifi
|-   |- include
|-   |- src
|-   |- wifid
|-       |- fxnWifiClient
|-       |- include
|- zigbee
|-   |- include
|-   |- src
|-   |- zigbeed
|-       |- ember
|-       |- include
|- zwave
|-   |- include
|-   |- src
|-   |- zwaved
|-       |- fxnZwaveClient
|-       |- include
|-       |- zwave

```

Kit de développement logiciel Zigbee de Silicon Labs

Le SDK Silicon Labs est présenté dans le IotManagedIntegrationsDeviceSDK-Middleware/*example*-iot-ace-z3-gateway dossier. Cette couche DPK ACS Zigbee est implémentée pour ce SDK Silicon Labs.

```

./IotManagedIntegrationsDeviceSDK-Middleware/example-iot-ace-zz3-gateway/
|- autogen
|- config
|- gecko_sdk_4.3.2
|-   |- platform
|-   |- protocol
|-   |- util

```

Service ACS Zigbee

Le code du service Zigbee se trouve dans le IotManagedIntegrationsDeviceSDK-Middleware/*example*-iot-ace-general/middleware/zigbee/ dossier. Les include sous-dossiers src et situés à cet emplacement contiennent tous les fichiers liés au service ACS Zigbee.

```
IotManagedIntegrationsDeviceSDK-Middleware/example-iot-ace-general/middleware/zigbee/  
src/  
|- zb_alloc.c  
|- zb_callbacks.c  
|- zb_database.c  
|- zb_discovery.c  
|- zb_log.c  
|- zb_main.c  
|- zb_region_info.c  
|- zb_server.c  
|- zb_svc.c  
|- zb_svc_pwr.c  
|- zb_timer.c  
|- zb_util.c  
|- zb_zdo.c  
|- zb_zts.c  
IotManagedIntegrationsDeviceSDK-Middleware/example-iot-ace-general/middleware/zigbee/  
include/  
|- init.zigbeeservice.rc  
|- zb_ace_log_uhl.h  
|- zb_alloc.h  
|- zb_callbacks.h  
|- zb_client_aipc.h  
|- zb_client_event_handler.h  
|- zb_database.h  
|- zb_discovery.h  
|- zb_log.h  
|- zb_region_info.h  
|- zb_server.h  
|- zb_svc.h  
|- zb_svc_pwr.h  
|- zb_timer.h  
|- zb_util.h  
|- zb_zdo.h  
|- zb_zts.h
```

Adaptateur ACS Zigbee

Le code de l'adaptateur ACS Zigbee se trouve dans le IotManagedIntegrationsDeviceSDK-Middleware/*example*-iot-ace-general/middleware/zigbee/api dossier. Les include sous-dossiers src et situés à cet emplacement contiennent tous les fichiers relatifs à la bibliothèque ACS Zigbee Adaptor.

```

IotManagedIntegrationsDeviceSDK-Middleware/example-iot-ace-general/middleware/zigbee/
api/src/
|- zb_client_aipc.c
|- zb_client_api.c
|- zb_client_event_handler.c
|- zb_client_zcl.c
IotManagedIntegrationsDeviceSDK-Middleware/example-iot-ace-general/middleware/zigbee/
api/include/
|- ace
|-   |- zb_adapter.h
|-   |- zb_command.h
|-   |- zb_network.h
|-   |- zb_types.h
|-   |- zb_zcl.h
|-   |- zb_zcl_cmd.h
|-   |- zb_zcl_color_control.h
|-   |- zb_zcl_hvac.h
|-   |- zb_zcl_id.h
|-   |- zb_zcl_identify.h
|-   |- zb_zcl_level.h
|-   |- zb_zcl_measure_and_sensing.h
|-   |- zb_zcl_onoff.h
|-   |- zb_zcl_power.h

```

Organisation du code du middleware Z-Wave

Ce qui suit montre l'organisation du code du middleware de référence Z-wave.

Rubriques

- [ACS Z-Wave DPK](#)
- [Silicon Labs ZWare et Zip Gateway](#)
- [Service ACS Z-Wave](#)
- [Adaptateur ACS Z-Wave](#)

ACS Z-Wave DPK

Le code de Z-Wave DPK se trouve dans le IotManagedIntegrationsDeviceSDK-Middleware/*example*-iot-ace-dpk/*example*/dpk/ace_hal/zwave dossier.

```

./IotManagedIntegrationsDeviceSDK-Middleware/example-iot-ace-dpk/example/dpk/ace_hal/

```



```
| - common
| -   | - fxnDBusClient
| -   | - include
| - kvs
| - log
| - wifi
| -   | - include
| -   | - src
| -   | - wifid
| -       | - fxnWifiClient
| -       | - include
| - zigbee
| -   | - include
| -   | - src
| -   | - zigbeed
| -       | - ember
| -       | - include
| - zwave
| -   | - include
| -   | - src
| -   | - zwaved
| -       | - fxnZwaveClient
| -       | - include
| -       | - zware
```

Silicon Labs ZWAre et Zip Gateway

Le code pour les laboratoires Silicon ZWAre et Zip Gateway se trouve dans le `IotManagedIntegrationsDeviceSDK-Middleware/example-iot-ace-z3-gateway` dossier. Cette couche DPK ACS Z-Wave est implémentée pour les passerelles Z-Wave C APIs et Zip.

```
./IotManagedIntegrationsDeviceSDK-Middleware/example-iot-ace-z3-gateway/
| - autogen
| - config
| - gecko_sdk_4.3.2
| -   | - platform
| -   | - protocol
| -   | - util
```

Service ACS Z-Wave

Le code du service Z-Wave se trouve dans le dossier répertorié dans le `IotManagedIntegrationsMiddlewares/exampleiot-ace-zwave-mw/` dossier. Les include dossiers `src` et situés à cet emplacement contiennent tous les fichiers relatifs au service ACS Z-Wave.

```
IotManagedIntegrationsDeviceSDK-Middleware/example-iot-ace-zwave-mw/src/  
|- zwave_mgr.c  
|- zwave_mgr_cc.c  
|- zwave_mgr_ipc_aipc.c  
|- zwave_svc.c  
|- zwave_svc_dispatcher.c  
|- zwave_svc_hsm.c  
|- zwave_svc_ipc_aipc.c  
|- zwave_svc_main.c  
|- zwave_svc_publish.c  
IotManagedIntegrationsDeviceSDK-Middleware/example-iot-ace-zwave-mw/include/  
|- ace  
|-   |- zwave_common_cc.h  
|-   |- zwave_common_cc_battery.h  
|-   |- zwave_common_cc_doorlock.h  
|-   |- zwave_common_cc_firmware.h  
|-   |- zwave_common_cc_meter.h  
|-   |- zwave_common_cc_notification.h  
|-   |- zwave_common_cc_sensor.h  
|-   |- zwave_common_cc_switch.h  
|-   |- zwave_common_cc_thermostat.h  
|-   |- zwave_common_cc_version.h  
|-   |- zwave_common_types.h  
|-   |- zwave_mgr.h  
|-   |- zwave_mgr_cc.h  
|- zwave_log.h  
|- zwave_mgr_internal.h  
|- zwave_mgr_ipc.h  
|- zwave_svc_hsm.h  
|- zwave_svc_internal.h  
|- zwave_utils.h
```

Adaptateur ACS Z-Wave

Le code de l'adaptateur ACS Zigbee se trouve dans le `IotManagedIntegrationsDeviceSDK-Middleware/example-iot-ace-zwave-mw/cli/` dossier. Le `include` dossier `src` and situé à cet emplacement contient tous les fichiers relatifs à la bibliothèque ACS Z-Wave Adaptor.

```
IotManagedIntegrationsDeviceSDK-Middleware/example-iot-ace-zwave-mw/cli/  
|- include  
|-   |- zwave_cli.h  
|- src  
|-   |- zwave_cli.yaml  
|-   |- zwave_cli_cc.c  
|-   |- zwave_cli_event_monitor.c  
|-   |- zwave_cli_main.c  
|-   |- zwave_cli_net.c
```

Intégrer le middleware au SDK

L'intégration du middleware sur le nouveau hub est abordée dans les sections suivantes.

Rubriques

- [Intégration de l'API du kit de portage de périphériques \(DPK\)](#)
- [Implémentation des références et organisation du code](#)

Intégration de l'API du kit de portage de périphériques (DPK)

Pour intégrer le SDK de n'importe quel fournisseur de chipsets au middleware, une interface API standard est fournie par la couche DPK (Device Porting Kit) située au milieu. Les fournisseurs de services d'intégrations gérées ODMs doivent ou doivent les implémenter sur la APIs base du SDK du fournisseur pris en charge par les chipsets Zigbee/Z-wave/Wi-Fi utilisés sur leurs hubs IoT.

Implémentation des références et organisation du code

À l'exception du middleware, tous les autres composants du Device SDK, tels que les intégrations gérées Device Agent et Common Data Model Bridge (CDMB), peuvent être utilisés sans aucune modification et doivent uniquement être compilés de manière croisée.

L'implémentation du middleware est basée sur le SDK Silicon Labs pour Zigbee et Z-Wave. Si les chipsets Z-Wave et Zigbee utilisés dans le nouveau hub sont pris en charge par le SDK Silicon Labs

présent dans le middleware, le middleware de référence peut être utilisé sans aucune modification. Il vous suffit de procéder à une compilation croisée du middleware pour qu'il puisse ensuite être exécuté sur le nouveau hub.

Le DPK (Device Porting Kit) APIs pour Zigbee se trouve dans `acehal_zigbee.c`, et l'implémentation de référence du DPK APIs est présente dans le dossier. `zigbee`

```
IotManagedIntegrationsDeviceSDK-Middleware/example-iot-ace-dpk/example/dpk/ace_hal/
zigbee/
|- CMakeLists.txt
|- include
|-   |- zigbee_log.h
|- src
|-   |- acehal_zigbee.c
|- zigbeed
|-   |- CMakeLists.txt
|-   |- ember
|-   |-   |- ace_ember_common.c
|-   |-   |- ace_ember_ctrl.c
|-   |-   |- ace_ember_hal_callbacks.c
|-   |-   |- ace_ember_network_creator.c
|-   |-   |- ace_ember_power_settings.c
|-   |-   |- ace_ember_zts.c
|-   |- include
|-   |-   |- zbd_api.h
|-   |-   |- zbd_callbacks.h
|-   |-   |- zbd_common.h
|-   |-   |- zbd_network_creator.h
|-   |-   |- zbd_power_settings.h
|-   |-   |- zbd_zts.h
```

Le DPK APIs pour Z-Wave se trouve dans le dossier `acehal_zwave.c` et l'implémentation de référence du DPK APIs est présente dans le dossier. `zwaved`

```
IotManagedIntegrationsDeviceSDK-Middleware/example-iot-ace-dpk/example/dpk/ace_hal/
zwave/
|- CMakeLists.txt
|- include
|-   |- zwave_log.h
|- src
|-   |- acehal_zwave.c
|- zwaved
```

```
|-  |- CMakeLists.txt
|-  |- fxnZwaveClient
|-  |-   |- zwave_client.c
|-  |-   |- zwave_client.h
|-  |- include
|-  |-   |- zwaved_cc_intf_api.h
|-  |-   |- zwaved_common_utils.h
|-  |-   |- zwaved_ctrl_api.h
|-  |- zware
|-  |-   |- ace_zware_cc_intf.c
|-  |-   |- ace_zware_common_utils.c
|-  |-   |- ace_zware_ctrl.c
|-  |-   |- ace_zware_debug.c
|-  |-   |- ace_zware_debug.h
|-  |-   |- ace_zware_internal.h
```

Comme point de départ pour implémenter la couche DPK pour un SDK d'un autre fournisseur, l'implémentation de référence peut être utilisée et modifiée. Les deux modifications suivantes seront nécessaires pour prendre en charge le SDK d'un autre fournisseur :

1. Remplacez le SDK du fournisseur actuel par le nouveau SDK du fournisseur dans le référentiel.
2. Implémentez le DDleware DPK (Device Porting Kit) APIs conformément au SDK du nouveau fournisseur.

Intégrations gérées SDK pour appareils finaux

Créez une plateforme IoT qui connecte les appareils intelligents aux intégrations gérées et traite les commandes via une interface de contrôle unifiée. Le SDK pour appareils finaux s'intègre au microprogramme de votre appareil et fournit une configuration simplifiée grâce aux composants périphériques du SDK, ainsi qu'une connectivité et une gestion sécurisées AWS IoT Core des AWS IoT appareils. Téléchargez la dernière version du SDK pour appareils finaux à partir du AWS Management Console

Ce guide explique comment implémenter le SDK de l'appareil final dans votre microprogramme. Passez en revue l'architecture, les composants et les étapes d'intégration pour commencer à créer votre implémentation.

Rubriques

- [Qu'est-ce que le SDK pour appareils finaux ?](#)
- [Architecture et composants du SDK pour appareils finaux](#)
- [Bénéficiaire](#)
- [Over-the-Air mises à jour](#)
- [Générateur de code de modèle de données](#)
- [Fonction C de faible niveau APIs](#)
- [Interactions entre les fonctionnalités et les appareils dans les intégrations gérées](#)
- [Commencez avec le SDK pour appareils finaux](#)

Qu'est-ce que le SDK pour appareils finaux ?

Qu'est-ce que le SDK pour appareils finaux ?

Le SDK pour appareils finaux est un ensemble de code source, de bibliothèques et d'outils fournis par AWS IoT. Conçu pour les environnements aux ressources limitées, le SDK prend en charge les appareils ne disposant que de 512 Ko de RAM et de 4 Mo de mémoire flash, tels que les appareils photo et les purificateurs d'air fonctionnant sous Linux intégré et des systèmes d'exploitation en temps réel (RTOS). Téléchargez la dernière version du SDK pour appareils finaux depuis la [console de AWS IoT gestion](#).

Composants de base

Le SDK combine un agent MQTT pour la communication dans le cloud, un gestionnaire de tâches pour la gestion des tâches et un gestionnaire d'intégrations gérées, le Data Model Handler. Ces composants fonctionnent ensemble pour fournir une connectivité sécurisée et une traduction automatique des données entre vos appareils et les intégrations gérées.

Pour connaître les exigences techniques détaillées, consultez le [Référence technique](#).

Architecture et composants du SDK pour appareils finaux

Cette section décrit l'architecture du SDK pour appareils finaux et la manière dont ses composants interagissent avec vos fonctions C de bas niveau. Le schéma suivant illustre les composants principaux et leurs relations dans le cadre du SDK.

Composants du SDK pour appareils finaux

L'architecture du SDK pour appareils finaux contient les composants suivants pour l'intégration des fonctionnalités d'intégration gérée :

Bénéficiaire

Crée des ressources d'appareils dans le cloud d'intégrations gérées, notamment des certificats d'appareils et des clés privées pour une communication MQTT sécurisée. Ces informations d'identification établissent des connexions fiables entre votre appareil et les intégrations gérées.

Agent MQTT

Gère les connexions MQTT via une bibliothèque cliente C adaptée aux threads. Ce processus d'arrière-plan gère les files de commandes dans les environnements multithread, avec des tailles de file d'attente configurables pour les appareils à mémoire limitée. Les messages sont acheminés via des intégrations gérées pour être traités.

Gestionnaire de tâches

Traite les mises à jour over-the-air (OTA) du microprogramme de l'appareil, des correctifs de sécurité et de la livraison des fichiers. Ce service intégré gère les mises à jour logicielles pour tous les appareils enregistrés.

Gestionnaire de modèles de données

Traduit les opérations entre les intégrations gérées et vos fonctions C de bas niveau en utilisant la AWS« mise en œuvre » du modèle de données Matter. Pour plus d'informations, consultez la [documentation Matter](#) sur GitHub.

Clés et certificats

[Gère les opérations cryptographiques via l'API PKCS #11, en prenant en charge à la fois les modules de sécurité matériels et les implémentations logicielles telles que Core. PKCS11](#) Cette API gère les opérations de certification pour les composants tels que le Provisionee et l'agent MQTT lors des connexions TLS.

Bénéficiaire

Le provisionee est un composant des intégrations gérées qui permet le provisionnement de la flotte par réclamation. Avec le fournisseur, vous approvisionnez vos appareils en toute sécurité. Le SDK crée les ressources nécessaires au provisionnement des appareils, notamment le certificat de l'appareil et les clés privées obtenus à partir du cloud d'intégrations gérées. Lorsque vous souhaitez approvisionner vos appareils, ou si des modifications peuvent vous obliger à reprovisionner vos appareils, vous pouvez faire appel au fournisseur.

Rubriques

- [Flux de travail Provisionee](#)
- [Définir les variables d'environnement](#)
- [Enregistrer un point de terminaison personnalisé](#)
- [Création d'un profil de provisionnement](#)
- [Création d'un objet géré](#)
- [Approvisionnement Wi-Fi des utilisateurs du SDK](#)
- [Approvisionnement de la flotte par sinistre](#)
- [Fonctionnalités de l'objet géré](#)

Flux de travail Provisionee

Le processus nécessite une configuration à la fois côté cloud et côté appareil. Les clients configurent les exigences du cloud, telles que les points de terminaison personnalisés, les profils de provisionnement et les objets gérés. Lors de la première mise sous tension de l'appareil, le fournisseur :

1. Se connecte au point de terminaison des intégrations gérées à l'aide d'un certificat de réclamation
2. Valide les paramètres de l'appareil par le biais de crochets de provisionnement du parc

3. Obtient et stocke un certificat permanent et une clé privée sur l'appareil
4. L'appareil utilise le certificat permanent pour se reconnecter
5. Découvre et télécharge les fonctionnalités de l'appareil vers des intégrations gérées

Une fois le provisionnement réussi, l'appareil communique directement avec les intégrations gérées. Le fournisseur s'active uniquement pour les tâches de réapprovisionnement.

Définir les variables d'environnement

Définissez les AWS informations d'identification suivantes dans votre environnement cloud :

```
$ export AWS_ACCESS_KEY_ID=YOUR-ACCOUNT-ACCESS-KEY-ID
$ export AWS_SECRET_ACCESS_KEY=YOUR-ACCOUNT-SECRET-ACCESS-KEY
$ export AWS_DEFAULT_REGION=YOUR-DEFAULT-REGION
```

Enregistrer un point de terminaison personnalisé

Utilisez la commande [RegisterCustomEndpoint](#) API dans votre environnement cloud pour créer un point de terminaison personnalisé pour device-to-cloud la communication.

```
aws iot-managed-integrations register-custom-endpoint
```

Exemple de réponse

```
{ "EndpointAddress": "[ACCOUNT-PREFIX]-ats.iot.AWS-REGION.amazonaws.com" }
```

Note

Stockez l'adresse du point de terminaison pour configurer les paramètres de provisionnement. Utilisez l'[GetCustomEndpoint](#) API pour renvoyer les informations du point de terminaison. Pour plus d'informations, consultez les sections [GetCustomEndpoint](#) API et [RegisterCustomEndpoint](#) API dans le Guide de référence de l'API Managed Integrations.

Création d'un profil de provisionnement

Créez un profil d'approvisionnement qui définit la méthode de provisionnement de votre flotte. Exécutez l'[CreateProvisioningProfile](#) API dans votre environnement cloud pour renvoyer un certificat de réclamation et une clé privée pour l'authentification de l'appareil :

```
aws iot-managed-integrations create-provisioning-profile \  
--provisioning-type "FLEET_PROVISIONING" \  
--name "PROVISIONING-PROFILE-NAME"
```

Exemple de réponse

```
{ "Arn": "arn:aws:iot-managed-integrations:AWS-REGION:YOUR-ACCOUNT-ID:provisioning-  
profile/PROFILE_NAME",  
  "ClaimCertificate": "string",  
  "ClaimCertificatePrivateKey": "string",  
  "Name": "ProfileName",  
  "ProvisioningType": "FLEET_PROVISIONING" }
```

Vous pouvez implémenter la bibliothèque d'abstraction de la PKCS11 plate-forme principale (PAL) pour qu'PKCS11 elle fonctionne avec votre appareil. Les ports PKCS11 PAL principaux doivent fournir un emplacement pour stocker le certificat de réclamation et la clé privée. Grâce à cette fonctionnalité, vous pouvez stocker en toute sécurité la clé privée et le certificat de l'appareil. Vous pouvez stocker la clé privée et le certificat sur un module de sécurité matériel (HSM) ou un module de plateforme sécurisée (TPM).

Création d'un objet géré

Enregistrez votre appareil auprès de Managed Integrations Cloud à l'aide de l'[CreateManagedThing](#) API. Indiquez le numéro de série (SN) et le code de produit universel (UPC) de votre appareil :

```
aws iot-managed-integrations create-managed-thing --role DEVICE \  
--authentication-material-type WIFI_SETUP_QR_BAR_CODE \  
--authentication-material "SN:DEVICE-SN;UPC:DEVICE-UPC;"
```

Voici un exemple de réponse d'API.

```
{
  "Arn": "arn:aws:iot-managed-integrations:AWS-REGION:ACCOUNT-ID:managed-
thing/59d3c90c55c4491192d841879192d33f",
  "CreatedAt": 1.730960226491E9,
  "Id": "59d3c90c55c4491192d841879192d33f"
}
```

L'API renvoie l'ID d'objet géré qui peut être utilisé pour la validation du provisionnement. Vous devrez fournir le numéro de série (SN) et le code de produit universel (UPC) de l'appareil, qui correspondent à l'objet géré approuvé lors de la transaction d'approvisionnement. La transaction renvoie un résultat similaire au suivant :

```
/**
 * @brief Device info structure.
 */
typedef struct iotmiDev_DeviceInfo
{
    char serialNumber[ IOTMI_DEVICE_MAX_SERIAL_NUMBER_LENGTH + 1U ];
    char universalProductCode[ IOTMI_DEVICE_MAX_UPC_LENGTH + 1U ];
    char internationalArticleNumber[ IOTMI_DEVICE_MAX_EAN_LENGTH + 1U ];
} iotmiDev_DeviceInfo_t;
```

Approvisionnement Wi-Fi des utilisateurs du SDK

Les fabricants d'appareils et les fournisseurs de solutions disposent de leur propre service de fourniture Wi-Fi propriétaire pour recevoir et configurer les informations d'identification Wi-Fi. Le service de fourniture Wi-Fi implique l'utilisation d'applications mobiles dédiées, de connexions Bluetooth Low Energy (BLE) et d'autres protocoles propriétaires pour transférer en toute sécurité les informations d'identification Wi-Fi lors du processus de configuration initiale.

L'utilisateur du SDK pour appareil final doit implémenter le service de provisionnement Wi-Fi et l'appareil peut se connecter à un réseau Wi-Fi.

Approvisionnement de la flotte par sinistre

À l'aide du fournisseur, l'utilisateur final peut fournir un certificat unique et l'enregistrer auprès d'intégrations gérées à l'aide du provisionnement par réclamation.

L'ID client peut être acquis soit à partir de la réponse du modèle d'approvisionnement, soit à partir du certificat de l'appareil `<common name>"_<serial number>`

Fonctionnalités de l'objet géré

Le fournisseur découvre les fonctionnalités des objets gérés, puis les télécharge vers les intégrations gérées. Il met les fonctionnalités à la disposition des applications et autres services. Les appareils, les autres clients Web et les services peuvent mettre à jour les fonctionnalités à l'aide de MQTT et de la rubrique MQTT réservée, ou du protocole HTTP à l'aide de l'API REST.

Over-the-Air mises à jour

Présentation de l'architecture OTA

Le processus de mise à jour Over-the-Air (OTA) implique la collaboration de plusieurs composants pour fournir des mises à jour du micrologiciel à vos appareils. Le schéma suivant illustre la façon dont une demande de mise à jour OTA est traitée par le biais de l'interaction entre le SDK du terminal, le SDK du Hub et la fonctionnalité.

L'architecture de mise à jour OTA comprend les composants suivants :

- Client : télécharge les documents de travail dans un compartiment S3 et lance les mises à jour via l'API
- Service OTA : gère la création, la validation et la gestion des emplois
- AWS IoT Tâches : gère l'exécution des tâches et leur diffusion sur les appareils
- Appareils : recevez et appliquez des mises à jour à l'aide du SDK Harmony

Conditions préalables

Avant de créer des tâches OTA, vous devez configurer les prérequis suivants :

Configuration de l'accès à Amazon S3

Pour activer les mises à jour OTA, vous devez télécharger vos documents de travail dans un compartiment Amazon S3 et configurer les autorisations d'accès appropriées :

1. Téléchargez votre document de travail OTA dans un compartiment S3
2. Ajoutez une politique de compartiment Amazon S3 qui accorde aux intégrations gérées l'accès à vos documents de travail :

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "PolicyForS3JobDocument",
      "Effect": "Allow",
      "Principal": {
        "Service": "iotmanagedintegrations.amazonaws.com"
      },
      "Action": "s3:GetObject",
      "Resource": [
        "arn:aws:s3:::YOUR_BUCKET/*",
        "arn:aws:s3:::YOUR_BUCKET/ota_job_document.json",
        "arn:aws:s3:::YOUR_BUCKET"
      ]
    }
  ]
}
```

Tâches de mise en œuvre Over-the-Air (OTA)

Vous pouvez créer des tâches OTA de deux manières, en fonction de vos exigences en matière de mise à jour et de votre stratégie de ciblage des appareils :

Mises à jour uniques des tâches OTA

Une tâche OTA unique contient une liste statique de cibles (ManagedThings) pour effectuer des mises à jour OTA. Vous pouvez ajouter jusqu'à 100 cibles à la fois. Le flux de travail utilise les AWS IoT tâches avec indexation de flotte tout en conservant la couche d'abstraction des intégrations gérées.

Utilisez l'exemple suivant pour créer une tâche OTA unique :

```
aws iotmanagedintegrations create-ota-task \
  --description "One-time OTA update" \
  --s3-url "s3://test-job-document-bucket/ota-job-document.json" \
  --protocol HTTP \
  --target ["arn:aws:iotmanagedintegrations:region:account id:managed-thing/managed
  thing id"] \
```

```
--ota-mechanism PUSH \  
--ota-type ONE_TIME \  
--client-token "foo" \  
--tags '{"key1":"foo","key2":"foo"}'
```

Mises à jour continues des tâches OTA

Le flux de travail de regroupement OTA (Over-the-Air) vous permet de déployer des mises à jour du microprogramme sur des groupes d'appareils en fonction d'attributs spécifiques, à l'aide de AWS IoT Jobs with Fleet Indexing tout en conservant la couche d'abstraction des intégrations gérées. Les tâches OTA continues utilisent une chaîne de requête au lieu de cibles spécifiques. Tous les appareils qui répondent aux critères de requête sont soumis à des mises à jour OTA, et les critères de requête sont continuellement réévalués. Les cibles correspondantes comporteront des déploiements d'emplois.

Configuration des prérequis

Avant de créer des tâches OTA continues, remplissez les conditions préalables suivantes :

1. Créez un objet géré en appelant l'[CreateManagedThing](#) API et effectuez le provisionnement de la flotte.
2. Ajoutez des attributs de métadonnées à vos objets gérés pour le ciblage des requêtes.

Ajoutez des attributs et des métadonnées à ManagedThing l'aide de l'[UpdateManagedThing](#) API :

```
aws iotmanagedintegrations update-managed-thing \  
--managed-thing-id "YOUR_MANAGED_THING_ID" \  
--meta-data '{"owner":"managedintegrations","version":"1.0"}'
```

Utilisez l'exemple suivant pour créer une tâche OTA continue :

```
aws iotmanagedintegrations create-ota-task \  
--description "Continuous OTA update" \  
--s3-url "s3://test-job-document-bucket/ota-job-document.json" \  
--protocol HTTP \  
--ota-mechanism PUSH \  
--ota-type CONTINUOUS \  
--client-token "foo" \  
--ota-target-query-string "attributes.owner=managedintegrations" \  

```

```
--tags '{"key1":"foo","key2":"foo"}'
```

Comprendre le flux de travail OTA continu

Le flux de mise à jour continue de l'OTA suit les étapes suivantes :

1. Vous mettez à jour les éléments gérés avec des attributs à l'aide de l'[UpdateManagedThing](#)API.
2. Créez une tâche OTA avec une chaîne de requête ciblant des attributs d'appareil spécifiques.
3. Le service OTA crée un groupe d'objets dynamique en AWS IoT Core fonction des attributs de requête.
4. IoT Jobs exécute des mises à jour sur les appareils correspondants
5. Vous surveillez la progression via l'[ListOtaTaskExecutions](#)API ou les notifications OTA via Kinesis Stream (si activé).

Différences entre les intégrations gérées (OTA) et les jobs IoT

La distinction fondamentale entre les intégrations gérées OTA et IoT Jobs réside dans l'orchestration et l'automatisation des services. Intégrations gérées OTA fournit une solution à service unique qui élimine la complexité de la coordination multiservices.

Ce que Managed Integrations OTA fait automatiquement :

- Création dynamique de groupes d'objets : génère automatiquement des AWS IoT Core groupes d'objets en fonction de vos critères de requête.
- Résolution cible : traduit les chaînes de requête (exemple :`attributes.owner=managedintegrations`) en cibles d'appareils réelles.
- Intégration des services : coordonne parfaitement AWS IoT Core les services IoT Jobs et d'indexation des flottes.
- Gestion du cycle de vie : gère l'ensemble du flux de travail OTA, de la création à la surveillance de l'exécution.

Ce que MI OTA élimine :

- Création de groupes d'objets dans AWS IoT Core.
- Ajouter des éléments à des groupes.
- Création d'emplois dans le domaine de l'IoT.

Intégrations gérées OTA gère les trois opérations en interne en fonction de votre chaîne de requête, en découvrant automatiquement les appareils qui correspondent à vos critères, en créant des jobs IoT en arrière-plan et en orchestrant le flux de travail OTA complet sans que vous ayez à interagir directement avec plusieurs AWS services.

Configuration des configurations de tâches OTA

Vous pouvez créer des configurations pour les mises à jour OTA afin de contrôler la manière dont les mises à jour sont déployées sur les appareils, de définir des conditions d'abandon et de configurer des délais d'expiration.

Exemple : CreateOtaTaskConfiguration

Utilisez l'exemple suivant pour créer une configuration de tâche OTA :

```
aws iotmanagedintegrations create-ota-task-configuration \  
  --description "OTA configuration" \  
  --name "MyOtaConfig" \  
  --push-config '{  
    "AbortConfig": {  
      "AbortConfigCriteriaList": [  
        {  
          "Action": "CANCEL",  
          "FailureType": "FAILED",  
          "MinNumberOfExecutedThings": 1,  
          "ThresholdPercentage": 90.0  
        }  
      ]  
    },  
    "RolloutConfig": {  
      "ExponentialRolloutRate": {  
        "BaseRatePerMinute": 1,  
        "IncrementFactor": 3.0,  
        "RateIncreaseCriteria": {  
          "numberOfNotifiedThings": 1  
        }  
      },  
      "MaximumPerMinute": 1  
    },  
    "TimeoutConfig": {  
      "InProgressTimeoutInMinutes": 100  
    }  
  }'
```



```
}' \  
--client-token "foo"
```

Appliquer les paramètres de configuration aux tâches OTA

Une fois la configuration créée, vous recevrez un élément ajouté à votre CreateOtaTask demande ainsi `taskConfigurationId` que des configurations supplémentaires :

```
aws iotmanagedintegrations create-ota-task \  
  --description "OTA with configuration" \  
  --s3-url "s3://test-job-document-bucket/ota-job-document.json" \  
  --protocol HTTP \  
  --target ["arn:aws:iotmanagedintegrations:region:account id:managed-thing/managed  
thing id"] \  
  --ota-mechanism PUSH \  
  --ota-type ONE_TIME \  
  --client-token "foo" \  
  --task-configuration-id "ae4f49352c5443369f43ad6c3a7f1580" \  
  --ota-scheduling-config '{  
    "EndBehavior": "STOP_ROLLOUT",  
    "EndTime": "2024-10-23T17:00",  
    "StartTime": "2024-10-20T17:00"  
  }' \  
  --ota-task-execution-retry-config '{  
    "RetryConfigCriteria": [  
      {  
        "FailureType": "FAILED",  
        "MinNumberOfRetries": 1  
      }  
    ]  
  }' \  
  --tags '{"key1":"foo","key2":"foo"}'
```

Surveillez les notifications OTA

Vous pouvez surveiller les mises à jour OTA à l'aide de deux méthodes différentes :

Notifications push via Kinesis Data Streams

Lorsque les notifications OTA sont activées, les événements relatifs à l'état des mises à jour sont automatiquement transmis à votre flux Kinesis. Cela fournit une visibilité en temps réel de la progression des mises à jour du micrologiciel sur tous les appareils.

Surveiller avec ListOtaTaskExecutions API

Vous pouvez utiliser l'[ListOtaTaskExecutions](#) API pour vérifier manuellement l'état des mises à jour OTA pour vos objets gérés :

```
aws iotmanagedintegrations list-ota-task-executions \  
  --task-id "task-123456789" \  
  --max-results 25
```

La réponse fournit un état d'exécution détaillé pour chaque objet géré :

```
{  
  "taskExecutionSummaries": [  
    {  
      "taskExecutionSummary": {  
        "executionNumber": 1,  
        "lastUpdatedAt": 1634567890,  
        "queuedAt": 1634567800,  
        "startedAt": 1634567830,  
        "status": "SUCCEEDED",  
        "retryAttempt": 0  
      },  
      "managedThingId": "device-001"  
    },  
    {  
      "taskExecutionSummary": {  
        "executionNumber": 1,  
        "lastUpdatedAt": 1634567920,  
        "queuedAt": 1634567800,  
        "startedAt": 1634567840,  
        "status": "IN_PROGRESS",  
        "retryAttempt": 0  
      },  
      "managedThingId": "device-002"  
    }  
  ],  
  "nextToken": "NEXT_TOKEN"  
}
```

Cette API vous permet de récupérer l'état d'exécution détaillé de chaque objet géré ciblé par une tâche OTA spécifique, y compris les horodatages et l'état actuel.

Traiter les documents de travail

Lorsque vous créez une tâche OTA, le gestionnaire de tâches exécute les étapes suivantes sur votre appareil. Lorsqu'une mise à jour est disponible, elle demande le document de travail via MQTT.

1. S'abonne aux rubriques de notification MQTT.
2. Appelle l'[StartNextPendingJobExecution](#) API pour les tâches en attente.
3. Reçoit les documents de travail disponibles.
4. Traite les mises à jour en fonction des délais que vous avez spécifiés.

À l'aide du gestionnaire de tâches, l'application peut déterminer s'il convient d'agir immédiatement ou d'attendre un délai d'expiration spécifié.

Implémenter un agent OTA

Lorsque vous recevez le document de tâche provenant d'intégrations gérées, vous devez disposer d'une implémentation de votre propre agent OTA qui traite le document de travail, télécharge les mises à jour et effectue toutes les opérations d'installation. L'agent OTA doit effectuer les étapes suivantes :

1. Analysez les documents de travail pour le microprogramme Amazon S3 URLs.
2. Téléchargez les mises à jour du microprogramme via HTTP.
3. Vérifiez les signatures numériques.
4. Installez les mises à jour validées.
5. Appelez `iotmi\_JobsHandler\_updateJobStatus` avec SUCCESS ou notre FAILED statut.

Lorsque votre appareil termine avec succès l'opération OTA, il doit appeler l'`iotmi\_JobsHandler\_updateJobStatus` API avec un statut égal `JobSucceeded` à pour signaler une tâche réussie.

```
/**
 * @brief Enumeration of possible job statuses.
 */
typedef enum{
    JobQueued,          /** The job is in the queue, waiting to be processed. */
    JobInProgress,      /** The job is currently being processed. */
    JobFailed,          /** The job processing failed. */
```

```
    JobSucceeded, /** The job processing succeeded. */
    JobRejected   /** The job was rejected, possibly due to an error or invalid
request. */
} iotmi_JobCurrentStatus_t;

/**
 * @brief Update the status of a job with optional status details.
 *
 * @param[in] pJobId Pointer to the job ID string.
 * @param[in] jobIdLength Length of the job ID string.
 * @param[in] status The new status of the job.
 * @param[in] statusDetails Pointer to a string containing additional details about the
job status.
 *
 * This can be a JSON-formatted string or NULL if no details
are needed.
 * @param[in] statusDetailsLength Length of the status details string. Set to 0 if
`statusDetails` is NULL.
 *
 * @return 0 on success, non-zero on failure.
 */
int iotmi_JobsHandler_updateJobStatus( const char * pJobId,
                                     size_t jobIdLength,
                                     iotmi_JobCurrentStatus_t status,
                                     const char * statusDetails,
                                     size_t statusDetailsLength );
```

Générateur de code de modèle de données

Découvrez comment utiliser le générateur de code pour le modèle de données. Le code généré peut être utilisé pour sérialiser et désérialiser les modèles de données échangés entre le cloud et l'appareil.

Le référentiel du projet contient un outil de génération de code permettant de créer des gestionnaires de modèles de données en code C. Les rubriques suivantes décrivent le générateur de code et le flux de travail.

Rubriques

- [Processus de génération de code](#)
- [Configuration de l'environnement](#)
- [Générer du code pour les appareils](#)

Processus de génération de code

Le générateur de code crée des fichiers source C à partir de trois entrées principales : AWS« l'implémentation du modèle de données Matter (fichier .matter) à partir de la plate-forme avancée Zigbee Cluster Library (ZCL), un plugin Python qui gère le prétraitement et des modèles Jinja2 qui définissent la structure du code. Pendant la génération, le plugin Python traite vos fichiers .matter en ajoutant des définitions de type globales, en organisant les types de données en fonction de leurs dépendances et en mettant en forme les informations pour le rendu du modèle.

L'image suivante décrit le générateur de code qui crée les fichiers source C.

Le SDK pour appareils finaux inclut des plugins Python et des modèles Jinja2 compatibles avec [codegen.py](#)le projet. [connectedhomeip](#) Cette combinaison génère plusieurs fichiers C pour chaque cluster en fonction de l'entrée de votre fichier .matter.

Les sous-rubriques suivantes décrivent ces fichiers.

- [Plug-in Python](#)
- [Modèles Jinja2](#)
- [\(Facultatif\) Schéma personnalisé](#)

Plug-in Python

Le générateur de code analyse `codegen.py` les fichiers .matter et envoie les informations sous forme d'objets Python au plugin. Le fichier du plugin `iotmi_data_model.py` prétraite ces données et affiche les sources à l'aide des modèles fournis. Le prétraitement inclut :

1. Ajout d'informations non disponibles `codegen.py`, telles que les types globaux
2. Exécution d'un tri topologique sur les types de données pour établir un ordre de définition correct

Note

Le tri topologique garantit que les types dépendants sont définis après leurs dépendances, quel que soit leur ordre d'origine.

Modèles Jinja2

Le SDK pour appareils finaux fournit des modèles Jinja2 adaptés aux gestionnaires de modèles de données et aux fonctions C de bas niveau.

Modèles Jinja2

Modèle	Source générée	Remarques
<code>cluster.h.jinja</code>	<code>iotmi_device_<cluster>.h</code>	Crée des fichiers d'en-tête de fonction C de bas niveau.
<code>cluster.c.jinja</code>	<code>iotmi_device_<cluster>.c</code>	Implémentez et enregistrez les pointeurs de la fonction de rappel avec le gestionnaire de modèle de données.
<code>cluster_type_helpers.h.jinja</code>	<code>iotmi_device_type_helpers_<cluster>.h</code>	Définit les prototypes de fonctions pour les types de données.
<code>cluster_type_helpers.c.jinja</code>	<code>iotmi_device_type_helpers_<cluster>.c</code>	Génère des prototypes de fonctions de type de données pour les énumérations, les bitmaps, les listes et les structures spécifiques aux clusters.
<code>iot_device_dm_types.h.jinja</code>	<code>iotmi_device_dm_types.h</code>	Définit les types de données C pour les types de données globaux.
<code>iot_device_type_helpers_global.h.jinja</code>	<code>iotmi_device_type_helpers_global.h</code>	Définit les types de données C pour les opérations globales.
<code>iot_device_type_helpers_global.c.jinja</code>	<code>iotmi_device_type_helpers_global.c</code>	Déclare les types de données standard, notamment les booléens, les entiers, les virgules flottantes, les chaînes,

Modèle	Source générée	Remarques
		les bitmaps, les listes et les structures.

(Facultatif) Schéma personnalisé

Le SDK pour appareils finaux combine le processus de génération de code standardisé avec un schéma personnalisé. Cela permet d'étendre le modèle de données de matière à vos appareils et à leurs logiciels. Les schémas personnalisés peuvent aider à décrire les capacités de device-to-cloud communication des appareils.

Pour obtenir des informations détaillées sur les modèles de données d'intégrations gérées, notamment le format, la structure et les exigences, consultez [Modèle de données d'intégrations gérées](#).

Utilisez l'outil `codegen.py` pour générer des fichiers source C pour un schéma personnalisé, comme suit :

Note

Chaque cluster personnalisé nécessite le même ID de cluster pour les trois fichiers suivants.

- Créez un schéma personnalisé dans un JSON format qui fournit une représentation des clusters pour les rapports de capacité afin de créer de nouveaux clusters personnalisés dans le cloud. Un fichier d'exemple se trouve à l'adresse `codegen/custom_schemas/custom.SimpleLighting@1.0`.
- Créez un fichier de définition ZCL (Zigbee Cluster Library) dans un XML format contenant les mêmes informations que le schéma personnalisé. Utilisez l'outil ZAP pour générer vos fichiers Matter IDL à partir du XML ZCL. Un fichier d'exemple se trouve à l'adresse `codegen/zcl/custom.SimpleLighting.xml`.
- La sortie de l'outil ZAP est Matter IDL File (`.matter`) et définit les clusters de matières correspondant à votre schéma personnalisé. Il s'agit de l'entrée de l'outil `codegen.py` permettant de générer des fichiers source C pour le SDK de l'appareil final. Un fichier d'exemple se trouve à l'adresse `codegen/matter_files/custom-light.matter`.

Pour obtenir des instructions détaillées sur la façon d'intégrer des modèles de données d'intégrations gérées personnalisés dans votre flux de travail de génération de code, consultez [Générer du code pour les appareils](#).

Configuration de l'environnement

Découvrez comment configurer votre environnement pour utiliser le générateur de code `codegen.py`.

Rubriques

- [Conditions préalables](#)
- [Configurez votre environnement](#)

Conditions préalables

Installez les éléments suivants avant de configurer votre environnement :

- Git
- Python 3.10 ou supérieur
- Poetry 1.2.0 ou supérieur

Configurez votre environnement

Utilisez la procédure suivante pour configurer votre environnement afin d'utiliser le générateur de code `codegen.py`.

1. Téléchargez la dernière version du [SDK pour appareils finaux](#) à partir du AWS Management Console.
2. Configurez l'environnement Python. Le projet `codegen` est basé sur Python et utilise Poetry pour la gestion des dépendances.
 - Installez les dépendances du projet en utilisant de la poésie dans le `codegen` répertoire :

```
poetry run poetry install --no-root
```

3. Configurez votre référentiel.

- a. Clonez le `connectedhomeip` dépôt. Il utilise le `codegen.py` script situé dans le `connectedhomeip/scripts/` dossier pour générer le code. Pour plus d'informations, consultez [connectedhomeip](#) on. GitHub

```
git clone -b v1.4.0.0 https://github.com/project-chip/connectedhomeip.git
```

- b. Clonez-le au même niveau que votre dossier `IoT-managed-integrations-End-Device-SDK` racine. La structure de votre dossier doit correspondre à ce qui suit :

```
| -connectedhomeip  
| -IoT-managed-integrations-End-Device-SDK
```

Note

Il n'est pas nécessaire de cloner les sous-modules de manière récursive.

Générer du code pour les appareils

Créez du code C personnalisé pour vos appareils à l'aide des outils de génération de code d'intégration gérés. Cette section décrit comment générer du code à partir de fichiers d'exemple inclus dans le SDK ou à partir de vos propres spécifications. Apprenez à utiliser les scripts de génération, à comprendre le processus de flux de travail et à créer du code adapté aux exigences de votre appareil.

Rubriques

- [Conditions préalables](#)
- [Générer du code pour les fichiers `.matter` personnalisés](#)
- [Flux de travail de génération de code](#)

Conditions préalables

1. Python 3.10 ou supérieur.
2. Commencez par un fichier `.matter` pour générer du code. Le SDK pour appareils finaux fournit deux exemples de fichiers dans le `codgen/matter_files` dossier :

- `custom-air-purifier.matter`
- `aws_camera.matter`

Note

Ces fichiers d'exemple génèrent du code pour les clusters d'applications de démonstration.

Générer du code

Exécutez cette commande pour générer du code dans le dossier de sortie :

```
bash ./gen-data-model-api.sh
```

Générer du code pour les fichiers `.matter` personnalisés

Pour générer le code d'un `.matter` fichier spécifique ou fournir votre propre `.matter` fichier, effectuez les tâches suivantes.

Pour générer le code des fichiers `.matter` personnalisés

1. Préparez votre fichier `.matter`
2. Exécutez la commande de génération :

```
./codegen.sh [--format] configs/dm_basic.json path-to-matter-file output-directory
```

(Facultatif) Pour générer du code avec un schéma personnalisé

1. Préparez votre schéma personnalisé au JSON format
2. Exécutez la commande de génération :

```
./codegen.sh [--format] configs/dm_basic.json path-to-matter-file output-directory  
--custom-schemas-dir path-to-custom-schema-directory
```

Les commandes ci-dessus utilisent plusieurs composants pour transformer votre `.matter` fichier en C code :

- `codegen.py` du projet `ConnectedHomeIP`
- Plugin Python situé à `codegen/py_scripts/iotmi_data_model.py`
- Modèles Jinja2 à partir du dossier `codegen/py_scripts/templates`

Le plugin définit les variables à transmettre aux modèles Jinja2, qui sont ensuite utilisés pour générer la sortie finale du code C. L'ajout du `--format drapeau` applique le format Clang au code généré.

Flux de travail de génération de code

Le processus de génération de code organise les structures de données de vos fichiers `.matter` à l'aide de fonctions utilitaires et d'un tri topologique. `topsort.py` Cela garantit un ordre correct des types de données et de leurs dépendances.

Le script combine ensuite les spécifications de votre fichier `.matter` avec le traitement du plugin Python pour extraire et formater les informations nécessaires. Enfin, il applique le formatage du modèle Jinja2 pour créer la sortie finale du code C.

Ce flux de travail garantit que les exigences spécifiques à votre appareil issues du fichier `.matter` sont traduites avec précision en code C fonctionnel qui s'intègre au système d'intégrations gérées.

Fonction C de faible niveau APIs

Intégrez le code spécifique à votre appareil aux intégrations gérées à l'aide de la fonction C de bas niveau fournie. APIs Cette section décrit les opérations d'API disponibles pour chaque cluster du modèle de AWS données pour des interactions efficaces entre l'appareil et le cloud. Découvrez comment implémenter des fonctions de rappel, émettre des événements, notifier les modifications d'attributs et enregistrer des clusters pour les points de terminaison de votre appareil.

Les principaux composants de l'API sont les suivants :

1. Structure de pointeur de fonction de rappel pour les attributs et les commandes
2. Fonctions d'émission d'événements
3. Fonctions de notification de modification d'attribut
4. Fonctions d'enregistrement des clusters

En les mettant en œuvre APIs, vous créez un pont entre les opérations physiques de votre appareil et les fonctionnalités cloud des intégrations gérées, garantissant ainsi une communication et un contrôle sans faille.

La section suivante illustre l'API du [OnOffcluster](#).

OnOff API de cluster

Le [OnOff.xml](#)cluster prend en charge les attributs et commandes suivants :

- Attributs :
 - OnOff (boolean)
 - GlobalSceneControl (boolean)
 - OnTime (int16u)
 - OffWaitTime (int16u)
 - StartUpOnOff (StartUpOnOffEnum)
- Commandes :
 - Off : () -> Status
 - On : () -> Status
 - Toggle : () -> Status
 - OffWithEffect : (EffectIdentifier: EffectIdentifierEnum, EffectVariant: enum8) -> Status
 - OnWithRecallGlobalScene : () -> Status
 - OnWithTimedOff : (OnOffControl: OnOffControlBitmap, OnTime: int16u, OffWaitTime: int16u) -> Status

Pour chaque commande, nous fournissons le pointeur de fonction mappé 1:1 que vous pouvez utiliser pour connecter votre implémentation.

Tous les rappels pour les attributs et les commandes sont définis dans une structure C nommée d'après le cluster.

Exemple de structure C

```
struct iotmiDev_clusterOnOff
{
```

```

/*
- Each attribute has a getter callback if it's readable

- Each attribute has a setter callback if it's writable

- The type of `value` are derived according to the data type of
  the attribute.

- `user` is the pointer passed during an endpoint setup

- The callback should return iotmiDev_DMStatus to report success or not.

- For unsupported attributes, just leave them as NULL.
*/
iotmiDev_DMStatus (*getOnTime)(uint16_t *value, void *user);
iotmiDev_DMStatus (*setOnTime)(uint16_t value, void *user);
/*
- Each command has a command callback

- If a command takes parameters, the parameters will be defined in a struct
  such as `iotmiDev_OnOff_OnWithTimedOffRequest` below.

- `user` is the pointer passed during an endpoint setup

- The callback should return iotmiDev_DMStatus to report success or not.

- For unsupported commands, just leave them as NULL.
*/
iotmiDev_DMStatus (*cmdOff)(void *user);
iotmiDev_DMStatus (*cmdOnWithTimedOff)(const iotmiDev_OnOff_OnWithTimedOffRequest
*request, void *user);
};

```

Outre la structure C, des fonctions de signalement des modifications d'attributs sont définies pour tous les attributs.

```

/* Each attribute has a report function for the customer to report
  an attribute change. An attribute report function is thread-safe.
*/
void iotmiDev_OnOff_OnTime_report_attr(struct iotmiDev_Endpoint *endpoint, uint16_t
newValue, bool immediate);

```

Les fonctions de reporting d'événements sont définies pour tous les événements spécifiques au cluster. Étant donné que le OnOff cluster ne définit aucun événement, voici un exemple tiré du CameraAvStreamManagement cluster.

```
/* Each event has a report function for the customer to report
   an event. An event report function is thread-safe.
   The iotmiDev_CameraAvStreamManagement_VideoStreamChangedEvent struct is
   derived from the event definition in the cluster.
*/
void iotmiDev_CameraAvStreamManagement_VideoStreamChanged_report_event(struct
    iotmiDev_Endpoint *endpoint, const
    iotmiDev_CameraAvStreamManagement_VideoStreamChangedEvent *event, bool immediate);
```

Chaque cluster possède également une fonction de registre.

```
iotmiDev_DMStatus iotmiDev_OnOffRegisterCluster(struct iotmiDev_Endpoint *endpoint,
    const struct iotmiDev_clusterOnOff *cluster, void *user);
```

Le pointeur utilisateur passé à la fonction de registre sera transmis aux fonctions de rappel.

Interactions entre les fonctionnalités et les appareils dans les intégrations gérées

Cette section décrit le rôle de l'implémentation de la fonction C et l'interaction entre l'appareil et la fonctionnalité du dispositif d'intégrations gérées.

Rubriques

- [Gestion des commandes à distance](#)
- [Gestion des événements non sollicités](#)

Gestion des commandes à distance

Les commandes à distance sont gérées par l'interaction entre le SDK du terminal et la fonctionnalité. Les actions suivantes décrivent un exemple de la manière dont vous pouvez allumer une ampoule à l'aide de cette interaction.

Le client MQTT reçoit la charge utile et la transmet au gestionnaire de modèles de données

Lorsque vous envoyez une commande à distance, le client MQTT reçoit le message des intégrations gérées au format JSON. Il transmet ensuite la charge utile au gestionnaire du modèle de données. Supposons, par exemple, que vous souhaitiez utiliser des intégrations gérées pour allumer une ampoule. L'ampoule possède un point de terminaison #1 qui supporte le OnOff cluster. Dans ce cas, lorsque vous envoyez la commande pour allumer l'ampoule, Managed Integrations envoie une demande via MQTT à l'appareil, indiquant qu'il souhaite invoquer la commande On sur le point de terminaison #1.

Le gestionnaire de modèle de données vérifie les fonctions de rappel et les invoque

Le gestionnaire de modèle de données analyse la requête JSON. Si la demande contient des propriétés ou des actions, le gestionnaire de modèle de données trouve les points de terminaison et appelle séquentiellement les fonctions de rappel correspondantes. Par exemple, dans le cas de l'ampoule, lorsque le Data Model Handler reçoit le message MQTT, il vérifie si la fonction de rappel correspondant à la commande On définie dans le OnOff cluster est enregistrée sur le point de terminaison #1.

L'implémentation du gestionnaire et de la fonction C exécute la commande

Le Data Model Handler appelle les fonctions de rappel appropriées qu'il a trouvées et les invoque. L'implémentation de la fonction C appelle ensuite les fonctions matérielles correspondantes pour contrôler le matériel physique et renvoie le résultat de l'exécution. Par exemple, dans le cas de l'ampoule, le Data Model Handler appelle la fonction de rappel et stocke le résultat de l'exécution. La fonction de rappel allume ensuite l'ampoule en conséquence.

Le gestionnaire de modèle de données renvoie le résultat de l'exécution

Une fois que toutes les fonctions de rappel ont été appelées, le Data Model Handler combine tous les résultats. Il emballe ensuite la réponse au format JSON et publie le résultat sur le cloud d'intégrations gérées à l'aide du client MQTT. Dans le cas de l'ampoule, le message MQTT contenu dans la réponse indiquera que l'ampoule a été allumée par la fonction de rappel.

Gestion des événements non sollicités

Les événements non sollicités sont également gérés par l'interaction entre le SDK du terminal et la fonctionnalité. Les actions suivantes décrivent comment procéder.

L'appareil envoie une notification au Data Model Handler

Lorsqu'un changement de propriété ou un événement se produit, par exemple lorsqu'un bouton physique a été enfoncé sur l'appareil, l'implémentation de la fonction C génère une notification d'événement non sollicitée et appelle la fonction de notification correspondante pour envoyer la notification au gestionnaire de modèle de données.

Le gestionnaire de modèle de données traduit les notifications

Le gestionnaire de modèle de données gère la notification reçue et la traduit dans le modèle de AWS données.

Data Model Handler publie une notification dans le cloud

Le Data Model Handler publie ensuite un événement non sollicité sur le cloud d'intégrations gérées à l'aide du client MQTT.

Commencez avec le SDK pour appareils finaux

Procédez comme suit pour exécuter le SDK du terminal sur un appareil Linux. Cette section vous guide tout au long de la configuration de l'environnement, de la configuration du réseau, de la mise en œuvre des fonctions matérielles et de la configuration des terminaux.

Important

Les applications de démonstration figurant dans le exemples répertoire et leur implémentation de la couche d'abstraction de plate-forme (PAL) ne platform/posix sont fournies qu'à titre de référence. Ne les utilisez pas dans des environnements de production.

Passez en revue attentivement chaque étape de la procédure suivante afin de garantir une intégration correcte des appareils avec les intégrations gérées.

Intégrer le SDK pour appareils finaux

1. Configuration de l' EC2 instance Amazon

Connectez-vous à l' EC2 instance Amazon AWS Management Console et lancez-la à l'aide d'une AMI Amazon Linux. Consultez [Commencer avec Amazon EC2](#) dans le [guide de l'utilisateur d'Amazon Elastic Container Registry](#).

2. Configuration de l'environnement de construction

Créez le code sur Amazon Linux 2023/x86_64 en tant qu'hôte de développement. Installez les dépendances de build nécessaires :

```
dnf install make gcc gcc-c++ cmake
```

3. (Facultatif) Configurer le réseau

Il est préférable d'utiliser le SDK de l'appareil final avec du matériel physique. Si vous utilisez Amazon EC2, ne suivez pas cette étape.

Si vous n'utilisiez pas Amazon EC2 avant d'utiliser l'exemple d'application, initialisez le réseau et connectez votre appareil à un réseau Wi-Fi disponible. Terminez la configuration du réseau avant le provisionnement des appareils :

```
/* Provisioning the device PKCS11 with claim credential. */  
status = deviceCredentialProvisioning();
```

4. Configuration des paramètres de provisionnement

Note

Suivez [Provisionnee](#) pour obtenir le certificat de réclamation et la clé privée avant de poursuivre.

Modifiez le fichier de configuration `example/project_name/device_config.sh` avec les paramètres de provisionnement suivants :

Paramètres de provisionnement

Paramètres des macros	Description	Comment obtenir ces informations
IOTMI_R00 T_CA_PATH	Le fichier de certificat racine de l'autorité de certification.	Vous pouvez télécharger ce fichier depuis la section Télécharger le certificat Amazon Root CA du guide du AWS IoT Core développeur.

Paramètres des macros	Description	Comment obtenir ces informations
IOTMI_CLAIM_CERTIFICATE_PATH	Le chemin d'accès au fichier de certificat de réclamation.	Pour obtenir le certificat de réclamation et la clé privée, créez un profil de provisionnement à l'aide de l' CreateProvisioningProfileAPI . Pour obtenir des instructions, veuillez consulter Création d'un profil de provisionnement .
IOTMI_CLAIM_PRIVATE_KEY_PATH	Le chemin d'accès au fichier de clé privée réclamé.	
IOTMI_MANAGED_INTEGRATIONS_ENDPOINT	URL du point de terminaison pour les intégrations gérées.	Pour obtenir le point de terminaison des intégrations gérées, utilisez l' RegisterCustomEndpointAPI . Pour obtenir des instructions, veuillez consulter Enregistrer un point de terminaison personnalisé .
IOTMI_MANAGED_INTEGRATIONS_ENDPOINT_PORT	Le numéro de port du point de terminaison des intégrations gérées	Par défaut, le port 8883 est utilisé pour les opérations de publication et d'abonnement MQTT. Le port 443 est défini pour l'extension TLS ALPN (Application Layer Protocol Negotiation) utilisée par les appareils.

5. Créez et exécutez les applications de démonstration

Cette section présente deux applications de démonstration Linux : une simple caméra de sécurité et un purificateur d'air, tous deux CMake utilisés comme système de construction.

a. Application de caméra de sécurité simple

Pour créer et exécuter l'application, exécutez les commandes suivantes :

```
>cd <path-to-code-drop>
# If you didn't generate cluster code earlier
>(cd codegen && poetry run poetry install --no-root && ./gen-data-model-api.sh)
>mkdir build
>cd build
>cmake ..
>cmake --build .
```

```
> ./examples/iotmi_device_sample_camera/iotmi_device_sample_camera
```

Cette démonstration implémente des fonctions C de bas niveau pour une caméra simulée avec un contrôleur de session RTC et des clusters d'enregistrement. Complétez le flux mentionné dans [Flux de travail Provisionee](#) avant de lancer.

Exemple de sortie de l'application de démonstration :

```
[2406832727][MAIN][INFO] ===== Device initialization and WIFI provisioning
=====
[2406832728][MAIN][INFO] fleetProvisioningTemplateName: XXXXXXXXXXXX
[2406832728][MAIN][INFO] managedintegrationsEndpoint: XXXXXXXXXX.account-prefix-
ats.iot.region.amazonaws.com
[2406832728][MAIN][INFO] pDeviceSerialNumber: XXXXXXXXXXXX
[2406832728][MAIN][INFO] universalProductCode: XXXXXXXXXXXX
[2406832728][MAIN][INFO] rootCertificatePath: XXXXXXXXXX
[2406832728][MAIN][INFO] pClaimCertificatePath: XXXXXXXXXX
[2406832728][MAIN][INFO] pClaimKeyPath: XXXXXXXXXXXXXXXXXXXX
[2406832728][MAIN][INFO] deviceInfo.serialNumber XXXXXXXXXXXX
[2406832728][MAIN][INFO] deviceInfo.universalProductCode XXXXXXXXXXXXXXXXXXXX
[2406832728][PKCS11][INFO] PKCS #11 successfully initialized.
[2406832728][MAIN][INFO] ===== Start certificate provisioning
=====
[2406832728][PKCS11][INFO] ===== Loading Root CA and claim credentials
through PKCS#11 interface =====
[2406832728][PKCS11][INFO] Writing certificate into label "Root Cert".
[2406832728][PKCS11][INFO] Creating a 0x1 type object.
[2406832728][PKCS11][INFO] Writing certificate into label "Claim Cert".
[2406832728][PKCS11][INFO] Creating a 0x1 type object.
[2406832728][PKCS11][INFO] Creating a 0x3 type object.
[2406832728][MAIN][INFO] ===== Fleet-provisioning-by-Claim =====
[2025-01-02 01:43:11.404995144][iotmi_device_sdkLog][INFO] [2406832728]
[MQTT_AGENT][INFO]
[2025-01-02 01:43:11.405106991][iotmi_device_sdkLog][INFO] Establishing a TLS
session to XXXXXXXXXXXXXXXXXXXX.account-prefix-ats.iot.region.amazonaws.com
[2025-01-02 01:43:11.405119166][iotmi_device_sdkLog][INFO]
[2025-01-02 01:43:11.844812513][iotmi_device_sdkLog][INFO] [2406833168]
[MQTT_AGENT][INFO]
[2025-01-02 01:43:11.844842576][iotmi_device_sdkLog][INFO] TLS session
connected
[2025-01-02 01:43:11.844852105][iotmi_device_sdkLog][INFO]
[2025-01-02 01:43:12.296421687][iotmi_device_sdkLog][INFO] [2406833620]
[MQTT_AGENT][INFO]
```

```

[2025-01-02 01:43:12.296449663][iotmi_device_sdkLog][INFO] Session present: 0.
[2025-01-02 01:43:12.296458997][iotmi_device_sdkLog][INFO]
[2025-01-02 01:43:12.296467793][iotmi_device_sdkLog][INFO] [2406833620]
[MQTT_AGENT][INFO]
[2025-01-02 01:43:12.296476275][iotmi_device_sdkLog][INFO] MQTT connect with
clean session.
[2025-01-02 01:43:12.296484350][iotmi_device_sdkLog][INFO]
[2025-01-02 01:43:13.171056119][iotmi_device_sdkLog][INFO] [2406834494]
[FLEET_PROVISIONING][INFO]
[2025-01-02 01:43:13.171082442][iotmi_device_sdkLog][INFO] Received accepted
response from Fleet Provisioning CreateKeysAndCertificate API.
[2025-01-02 01:43:13.171092740][iotmi_device_sdkLog][INFO]
[2025-01-02 01:43:13.171122834][iotmi_device_sdkLog][INFO] [2406834494]
[FLEET_PROVISIONING][INFO]
[2025-01-02 01:43:13.171132400][iotmi_device_sdkLog][INFO] Received privatekey
and certificate with Id: XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
[2025-01-02 01:43:13.171141107][iotmi_device_sdkLog][INFO]
[2406834494][PKCS11][INFO] Creating a 0x3 type object.
[2406834494][PKCS11][INFO] Writing certificate into label "Device Cert".
[2406834494][PKCS11][INFO] Creating a 0x1 type object.
[2025-01-02 01:43:18.584615126][iotmi_device_sdkLog][INFO] [2406839908]
[FLEET_PROVISIONING][INFO]
[2025-01-02 01:43:18.584662031][iotmi_device_sdkLog][INFO] Received accepted
response from Fleet Provisioning RegisterThing API.
[2025-01-02 01:43:18.584671912][iotmi_device_sdkLog][INFO]
[2025-01-02 01:43:19.100030237][iotmi_device_sdkLog][INFO] [2406840423]
[FLEET_PROVISIONING][INFO]
[2025-01-02 01:43:19.100061720][iotmi_device_sdkLog][INFO] Fleet-provisioning
iteration 1 is successful.
[2025-01-02 01:43:19.100072401][iotmi_device_sdkLog][INFO]
[2406840423][MQTT][ERROR] MQTT Connection Disconnected Successfully
[2025-01-02 01:43:19.216938181][iotmi_device_sdkLog][INFO] [2406840540]
[MQTT_AGENT][INFO]
[2025-01-02 01:43:19.216963713][iotmi_device_sdkLog][INFO] MQTT agent thread
leaves thread loop for iotmiDev_MQTTAgentStop.
[2025-01-02 01:43:19.216973740][iotmi_device_sdkLog][INFO]
[2406840540][MAIN][INFO] iotmiDev_MQTTAgentStop is called to break thread loop
function.
[2406840540][MAIN][INFO] Successfully provision the device.
[2406840540][MAIN][INFO] Client ID :
XXXXXXXXXXXXXXXXXXXXX_XXXXXXXXXXXXXXXXXXXXX
[2406840540][MAIN][INFO] Managed thing ID : XXXXXXXXXXXXXXXXXXXXXXXX
[2406840540][MAIN][INFO] ===== application loop
=====

```

```
[2025-01-02 01:43:19.217094828][iotmi_device_sdkLog][INFO] [2406840540]
[MQTT_AGENT][INFO]
[2025-01-02 01:43:19.217124600][iotmi_device_sdkLog][INFO] Establishing a TLS
session to XXXXXXXXXX.account-prefix-ats.iot.region.amazonaws.com:8883
[2025-01-02 01:43:19.217138724][iotmi_device_sdkLog][INFO]
[2406840540][Cluster OnOff][INFO] exampleOnOffInitCluster() for endpoint#1
[2406840540][MAIN][INFO] Press Ctrl+C when you finish testing...
[2406840540][Cluster ActivatedCarbonFilterMonitoring][INFO]
exampleActivatedCarbonFilterMonitoringInitCluster() for endpoint#1
[2406840540][Cluster AirQuality][INFO] exampleAirQualityInitCluster() for
endpoint#1
[2406840540][Cluster CarbonDioxideConcentrationMeasurement][INFO]
exampleCarbonDioxideConcentrationMeasurementInitCluster() for endpoint#1
[2406840540][Cluster FanControl][INFO] exampleFanControlInitCluster() for
endpoint#1
[2406840540][Cluster HepaFilterMonitoring][INFO]
exampleHepaFilterMonitoringInitCluster() for endpoint#1
[2406840540][Cluster Pm1ConcentrationMeasurement][INFO]
examplePm1ConcentrationMeasurementInitCluster() for endpoint#1
[2406840540][Cluster Pm25ConcentrationMeasurement][INFO]
examplePm25ConcentrationMeasurementInitCluster() for endpoint#1
[2406840540][Cluster TotalVolatileOrganicCompoundsConcentrationMeasurement]
[INFO]
exampleTotalVolatileOrganicCompoundsConcentrationMeasurementInitCluster() for
endpoint#1
[2025-01-02 01:43:19.648185488][iotmi_device_sdkLog][INFO] [2406840971]
[MQTT_AGENT][INFO]
[2025-01-02 01:43:19.648211988][iotmi_device_sdkLog][INFO] TLS session
connected
[2025-01-02 01:43:19.648225583][iotmi_device_sdkLog][INFO]

[2025-01-02 01:43:19.938281231][iotmi_device_sdkLog][INFO] [2406841261]
[MQTT_AGENT][INFO]
[2025-01-02 01:43:19.938304799][iotmi_device_sdkLog][INFO] Session present: 0.
[2025-01-02 01:43:19.938317404][iotmi_device_sdkLog][INFO]
```

b. Application simple du purificateur d'air

Pour créer et exécuter l'application, exécutez les commandes suivantes :

```
>cd <path-to-code-drop>
# If you didn't generate cluster code earlier
>(cd codegen && poetry run poetry install --no-root && ./gen-data-model-api.sh)
>mkdir build
```

```
>cd build
>cmake ..
>cmake --build .
>./examples/iotmi_device_dm_air_purifier/iotmi_device_dm_air_purifier_demo
```

Cette démonstration implémente des fonctions C de bas niveau pour un purificateur d'air simulé avec 2 terminaux et les clusters pris en charge suivants :

Clusters pris en charge pour le terminal du purificateur d'air

Endpoint	Clusters
Point final #1 : Purificateur d'air	OnOff
	Contrôle du ventilateur
	Surveillance des filtres HEPA
	Surveillance du filtre à charbon actif
Point final #2 : Capteur de qualité de l'air	Qualité de l'air
	Mesure de la concentration de dioxyde de carbone
	Mesure de la concentration de formaldéhyde
	Mesure de la concentration de Pm25
	Mesure de la concentration de Pm1
	Mesure de la concentration totale de composés organiques volatils

Le résultat est similaire à celui de l'application de démonstration de caméra, avec différents clusters pris en charge.

6. Étapes suivantes :

Le SDK pour appareils finaux et les applications de démonstration des intégrations gérées s'exécutent désormais sur vos instances Amazon EC2 . Cela vous permet de développer et de

tester vos applications sur votre propre matériel physique. Avec cette configuration, vous pouvez tirer parti du service d'intégrations gérées pour contrôler vos AWS IoT appareils.

a. Développer des fonctions de rappel matérielles

Avant d'implémenter les fonctions de rappel matériel, comprenez le fonctionnement de l'API. Cet exemple utilise le On/Off cluster et OnOff l'attribut pour contrôler la fonction d'un appareil. Pour plus de détails sur l'API, consultez [Fonction C de faible niveau APIs](#).

```
struct DeviceState
{
    struct iotmiDev_Agent *agent;
    struct iotmiDev_Endpoint *endpointLight;
    /* This simulates the HW state of OnOff */
    bool hwState;
};

/* This implementation for OnOff getter just reads
the state from the DeviceState */
iotmiDev_DMStatus exampleGetOnOff(bool *value, void *user)
{
    struct DeviceState *state = (struct DeviceState *)(user);
    *value = state->hwState;
    return iotmiDev_DMStatusOk;
}
```

b. Configuration des points de terminaison et des fonctions de rappel du matériel

Après avoir implémenté les fonctions, créez des points de terminaison et enregistrez vos rappels. Réalisez les tâches suivantes :

- i. Créez un agent de périphérique.
 - A. Créez un agent de périphérique `iotmiDev_Agent_new()` avant d'appeler toute autre fonction du SDK.
 - B. Au minimum, votre configuration doit inclure les paramètres `ThingID` et `ClientID`.
 - C. Utilisez cette `iotmiDev_Agent_initDefaultConfig()` fonction pour définir des valeurs par défaut raisonnables pour des paramètres tels que la taille des files d'attente et le nombre maximal de points de terminaison.

- D. Lorsque vous avez fini d'utiliser les ressources, libérez-les avec la `iotmiDev_Agent_free()` fonction. Cela permet d'éviter les fuites de mémoire et de garantir une gestion appropriée des ressources de votre application.
- ii. Remplissez les pointeurs de la fonction de rappel pour chaque structure de cluster que vous souhaitez prendre en charge.
- iii. Configurez les points de terminaison et enregistrez les clusters pris en charge.

Créez des points de terminaison avec `iotmiDev_Agent_addEndpoint()`, ce qui nécessite :

- A. Un identifiant de point de terminaison unique.
- B. Un nom de point de terminaison descriptif
- C. Un ou plusieurs types de périphériques conformes aux définitions AWS des modèles de données.
- D. Après avoir créé les points de terminaison, enregistrez les clusters à l'aide des fonctions d'enregistrement spécifiques aux clusters appropriées.
- E. Chaque enregistrement de cluster nécessite des fonctions de rappel pour les attributs et les commandes. Le système transmet votre pointeur de contexte utilisateur aux rappels pour maintenir l'état entre les appels.

```
struct DeviceState
{
    struct iotmiDev_Agent * agent;
    struct iotmiDev_Endpoint *endpoint1;

    /* OnOff cluster states*/
    bool hwState;
};

/* This implementation for OnOff getter just reads
the state from the DeviceState */
iotmiDev_DMStatus exampleGetOnOff( bool * value, void * user )
{
    struct DeviceState * state = ( struct DeviceState * ) ( user );
    *value = state->hwState;
    printf( "%s(): state->hwState: %d\n", __func__, state->hwState );
    return iotmiDev_DMStatusOk;
```



```

}

iotmiDev_DMStatus exampleGetOnTime( uint16_t * value, void * user )
{
    *value = 0;
    printf( "%s(): OnTime is %u\n", __func__, *value );
    return iotmiDev_DMStatusOk;
}

iotmiDev_DMStatus exampleGetStartupOnOff( iotmiDev_OnOff_StartUpOnOffEnum *
value, void * user )
{
    *value = iotmiDev_OnOff_StartUpOnOffEnum_Off;
    printf( "%s(): StartupOnOff is %d\n", __func__, *value );
    return iotmiDev_DMStatusOk;
}

void setupOnOff( struct DeviceState *state )
{
    struct iotmiDev_clusterOnOff clusterOnOff = {
        .getOnOff = exampleGetOnOff,
        .getOnTime = exampleGetOnTime,
        .getStartupOnOff = exampleGetStartupOnOff,
    };
    iotmiDev_OnOffRegisterCluster( state->endpoint1,
                                &clusterOnOff,
                                ( void * ) state);
}

/* Here is the sample setting up an endpoint 1 with OnOff
cluster. Note all error handling code is omitted. */
void setupAgent(struct DeviceState *state)
{
    struct iotmiDev_Agent_Config config = {
        .thingId = IOTMI_DEVICE_MANAGED_THING_ID,
        .clientId = IOTMI_DEVICE_CLIENT_ID,
    };
    iotmiDev_Agent_InitDefaultConfig(&config);

    /* Create a device agent before calling other SDK APIs */
    state->agent = iotmiDev_Agent_new(&config);

    /* Create endpoint#1 */

```

```

state->endpoint1 = iotmiDev_Agent_addEndpoint( state->agent,
                                                1,
                                                "Data Model Handler Test
Device",
                                                (const char*[])
{ "Camera" },
                                                1 );

setupOnOff(state);
}

```

c. Utilisez le gestionnaire de tâches pour obtenir le document de tâche

i. Lancez un appel vers votre application OTA :

```

static iotmi_JobCurrentStatus_t processOTA( iotmi_JobData_t * pJobData )
{
    iotmi_JobCurrentStatus_t jobCurrentStatus = JobSucceeded;

    ...
    // This function should create OTA tasks
    jobCurrentStatus = YOUR_OTA_FUNCTION(iotmi_JobData_t * pJobData);
    ...

    return jobCurrentStatus;
}

```

ii. Appelez `iotmi_JobsHandler_start` pour initialiser le gestionnaire de tâches.

iii. Appelez `iotmi_JobsHandler_getJobDocument` pour récupérer le document de travail à partir des intégrations gérées.

iv. Lorsque le document de tâches est obtenu avec succès, écrivez votre opération OTA personnalisée dans la `processOTA` fonction et renvoyez un `JobSucceeded` statut.

```

static void prvJobsHandlerThread( void * pParam )
{
    JobsHandlerStatus_t status = JobsHandlerSuccess;
    iotmi_JobData_t jobDocument;
    iotmiDev_DeviceRecord_t * pThreadParams = ( iotmiDev_DeviceRecord_t * )
pParam;
    iotmi_JobsHandler_config_t config = { .pManagedThingID = pThreadParams-
>pManagedThingID, .jobsQueueSize = 10 };

    status = iotmi_JobsHandler_start( &config );
}

```

```
if( status != JobsHandlerSuccess )
{
    LogError( ( "Failed to start Jobs Handler." ) );
    return;
}

while( !bExit )
{
    status = iotmi_JobsHandler_getJobDocument( &jobDocument, 30000 );

    switch( status )
    {
        case JobsHandlerSuccess:
        {
            LogInfo( ( "Job document received." ) );
            LogInfo( ( "Job ID: %.*s", ( int ) jobDocument.jobIdLength,
jobDocument.pJobId ) );
            LogInfo( ( "Job document: %.*s", ( int )
jobDocument.jobDocumentLength, jobDocument.pJobDocument ) );

            /* Process the job document */
            iotmi_JobCurrentStatus_t jobStatus =
processOTA( &jobDocument );

            iotmi_JobsHandler_updateJobStatus( jobDocument.pJobId,
jobDocument.jobIdLength, jobStatus, NULL, 0 );

            iotmiJobsHandler_destroyJobDocument(&jobDocument);

            break;
        }
        case JobsHandlerTimeout:
        {
            LogInfo( ( "No job document available. Polling for job
document." ) );

            iotmi_JobsHandler_pollJobDocument();

            break;
        }
        default:
        {
            LogError( ( "Failed to get job document." ) );
        }
    }
}
```

```
        break;
    }
}

while( iotmi_JobsHandler_getJobDocument( &jobDocument, 0 ) ==
JobsHandlerSuccess )
{
    /* Before stopping the Jobs Handler, process all the remaining
jobs. */

    LogInfo( ( "Job document received before stopping." ) );
    LogInfo( ( "Job ID: %.*s", ( int ) jobDocument.jobIdLength,
jobDocument.pJobId ) );
    LogInfo( ( "Job document: %.*s", ( int )
jobDocument.jobDocumentLength, jobDocument.pJobDocument ) );

    storeJobs( &jobDocument );

    iotmiJobsHandler_destroyJobDocument(&jobDocument);
}

iotmi_JobsHandler_stop();

LogInfo( ( "Job handler thread end." ) );
}
```

Portez le SDK de l'appareil final sur votre appareil

Portez le SDK de l'appareil final sur la plate-forme de votre appareil. Suivez ces étapes pour connecter vos appareils à la gestion des AWS IoT appareils.

Téléchargez et vérifiez le SDK de l'appareil final

1. Téléchargez la dernière version du SDK pour appareils finaux depuis la console des [intégrations gérées](#).
2. Vérifiez que votre plateforme figure dans la liste des plateformes prises en charge dans [Référence : Plateformes prises en charge](#).

 Note

Le SDK du terminal a été testé sur les plateformes spécifiées. D'autres plateformes peuvent fonctionner, mais elles n'ont pas été testées.

3. Extrayez (décompressez) les fichiers du SDK dans votre espace de travail.
4. Configurez votre environnement de construction avec les paramètres suivants :
 - Chemins du fichier source
 - Répertoires de fichiers d'en-tête
 - Bibliothèques requises
 - Drapeaux du compilateur et de l'éditeur de liens
5. Avant de porter la couche d'abstraction de plate-forme (PAL), assurez-vous que les fonctionnalités de base de votre plate-forme sont initialisées. Les fonctionnalités incluent :
 - Tâches du système d'exploitation
 - Périphériques
 - Interfaces réseau
 - Exigences spécifiques à la plate-forme

Portez le PAL sur votre appareil

1. Créez un nouveau répertoire pour les implémentations spécifiques à votre plate-forme dans le répertoire de plate-forme existant. Par exemple, si vous utilisez FreeRTOS, créez un répertoire dans `platform/freertos`

Exemple Structure du répertoire du SDK

```
### <SDK_ROOT_FOLDER>
#   ### CMakeLists.txt
#   ### LICENSE.txt
#   ### cmake
#   ### commonDependencies
#   ### components
#   ### docs
#   ### examples
```

```
#   ### include
#   ### lib
#   ### platform
#   ### test
#   ### tools
```

2. Copiez les fichiers d'implémentation de référence POSIX (.c et .h) du dossier posix vers le nouveau répertoire de votre plateforme. Ces fichiers fournissent un modèle pour les fonctions que vous devrez implémenter.
 - Gestion de la mémoire flash pour le stockage des informations d'identification
 - Implémentation du PKCS #11
 - Interface de transport réseau
 - Synchronisation du temps
 - Fonctions de redémarrage et de réinitialisation du système
 - Mécanismes de journalisation
 - Configurations spécifiques à l'appareil
3. Configurez l'authentification TLS (Transport Layer Security) avec MBed TLS.
 - Utilisez l'implémentation POSIX fournie si vous disposez déjà d'une version MBed TLS qui correspond à la version du SDK sur votre plate-forme.
 - Avec une version TLS différente, vous implémentez les crochets de transport pour votre pile TLS avec TCP/IP stack.
4. Comparez la configuration MbedTLS de votre plateforme avec les exigences du SDK dans. `platform/posix/mbedtls/mbedtls_config.h` Assurez-vous que toutes les options requises sont activées.
5. Le SDK s'appuie sur le CoreMQTT pour interagir avec le cloud. Par conséquent, vous devez implémenter une couche de transport réseau qui utilise la structure suivante :

```
typedef struct TransportInterface
{
    TransportRecv_t recv;
    TransportSend_t send;
    NetworkContext_t * pNetworkContext;
} TransportInterface_t;
```

Pour plus d'informations, consultez la [documentation de l'interface de transport](#) sur le site Web de FreeRTOS.

6. (Facultatif) Le SDK utilise l'API PKCS #11 pour gérer les opérations de certification. CorePKCS est une implémentation PKCS #11 non spécifique au matériel pour le prototypage. Nous vous recommandons d'utiliser des cryptoprocresseurs sécurisés tels que le Trusted Platform Module (TPM), le Hardware Security Module (HSM) ou le Secure Element dans votre environnement de production :

- Passez en revue l'exemple d'implémentation PKCS #11 qui utilise le système de fichiers Linux pour la gestion des informations d'identification sur. `platform/posix/corePKCS11-mbedtls`
- Implémentez la couche PAL PKCS #11 sur `commonDependencies/core_pkcs11/corePKCS11/source/include/core_pkcs11.h`.
- Implémentez le système de fichiers Linux à l'adresse `platform/posix/corePKCS11-mbedtls/source/iotmi_pal_Pkcs11Operations.c`.
- Implémentez la fonction de stockage et de chargement de votre type de stockage sur `platform/include/iotmi_pal_Nvm.h`.
- Implémentez l'accès aux fichiers standard à l'adresse `platform/posix/source/iotmi_pal_Nvm.c`.

Pour des instructions de portage détaillées, voir [Portage de la PKCS11 bibliothèque principale](#) dans le Guide de l'utilisateur de FreeRTOS.

7. Ajoutez les bibliothèques statiques du SDK à votre environnement de construction :
 - Configurez les chemins de bibliothèque pour résoudre les problèmes liés à l'éditeur de liens ou les conflits de symboles
 - Vérifiez que toutes les dépendances sont correctement liées

Testez votre port

Vous pouvez utiliser l'exemple d'application existant pour tester votre port. La compilation doit être terminée sans erreur ni avertissement.

Note

Nous vous recommandons de commencer par l'application multitâche la plus simple possible. L'exemple d'application fournit un équivalent multitâche.

1. Trouvez l'exemple d'application dans `examples/[device_type_sample]`.
2. Convertissez le `main.c` fichier en projet et ajoutez une entrée pour appeler la fonction `main()` existante.
3. Vérifiez que vous pouvez compiler correctement l'application de démonstration.

Référence technique

Rubriques

- [Référence : Plateformes prises en charge](#)
- [Référence : Exigences techniques](#)
- [Référence : Common API](#)

Référence : Plateformes prises en charge

Le tableau suivant indique les plateformes prises en charge par le SDK.

Plateformes prises en charge

Plateforme	Architecture	Système d'exploitation
Linux x86_64	x86_64	Linux
Ambarella	Bras (8) AArch64	Linux
Amebad	Armv8-M 32 bits	FreeRTOS
ESP32S3	Xtensa 32 bits LX7	FreeRTOS

Référence : Exigences techniques

Le tableau suivant indique les exigences techniques du SDK, y compris l'espace RAM. Le SDK de l'appareil final lui-même nécessite environ 5 à 10 Mo d'espace ROM lorsque vous utilisez la même configuration.

Espace RAM

SDK et composants	Espace requis (octets utilisés)
SDK de l'appareil final lui-même	180 KO
File d'attente de commandes par défaut de l'agent MQTT	480 octets (peut être configuré)
File d'attente entrante par défaut de l'agent MQTT	320 octets (peut être configuré)

Référence : Common API

Cette section répertorie les opérations d'API qui ne sont pas spécifiques à un cluster.

```

/* return code for data model related API */
enum iotmiDev_DMStatus
{
    /* The operation succeeded */
    iotmiDev_DMStatusOk = 0,
    /* The operation failed without additional information */
    iotmiDev_DMStatusFail = 1,
    /* The operation has not been implemented yet. */
    iotmiDev_DMStatusNotImplement = 2,
    /* The operation is to create a resource, but the resource already exists. */
    iotmiDev_DMStatusExist = 3,
}

/* The opaque type to represent a instance of device agent. */
struct iotmiDev_Agent;

/* The opaque type to represent an endpoint. */
struct iotmiDev_Endpoint;

```

```
/* A device agent should be created before calling other API */
struct iotmiDev_Agent* iotmiDev_create_agent();

/* Destroy the agent and free all occupied resources */
void iotmiDev_destroy_agent(struct iotmiDev_Agent *agent);

/* Add an endpoint, which starts with empty capabilities */
struct iotmiDev_Endpoint* iotmiDev_addEndpoint(struct iotmiDev_Agent *handle, uint16
    id, const char *name);

/* Test all clusters registered within an endpoint.
    Note: this API might exist only for early drop. */
void iotmiDev_testEndpoint(struct iotmiDev_Endpoint *endpoint);
```

Sécurité dans les intégrations gérées pour AWS IoT Device Management

La sécurité du cloud AWS est la priorité absolue. En tant que AWS client, vous bénéficiez de centres de données et d'architectures 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 la sécurité du cloud et la sécurité dans le cloud comme suit :

- 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 aux intégrations gérées, voir [AWS Services concernés par programme de conformitéAWS](#) .
- Sécurité dans le cloud — Votre responsabilité est déterminée par le AWS service que vous utilisez. Vous êtes également responsable d'autres facteurs, y compris de la sensibilité de vos données, des exigences de votre entreprise, ainsi que de la législation et de la réglementation applicables.

Cette documentation vous aide à comprendre comment appliquer le modèle de responsabilité partagée lors de l'utilisation d'intégrations gérées. Les rubriques suivantes expliquent comment configurer les intégrations gérées pour répondre à 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 ressources d'intégrations gérées.

Rubriques

- [Protection des données dans les intégrations gérées](#)
- [Gestion des identités et des accès pour les intégrations gérées](#)
- [Utilisation à des AWS Secrets Manager fins de protection des données pour les flux de travail C2C](#)
- [Validation de conformité pour les intégrations gérées](#)
- [Utilisez des intégrations gérées avec les points de terminaison VPC d'interface](#)
- [Connectez-vous à des intégrations gérées pour les points de terminaison AWS IoT Device Management FIPS](#)

Protection des données dans les intégrations gérées

Le [modèle de responsabilité AWS partagée](#) s'applique à la protection des données dans les intégrations gérées pour AWS IoT Device Management. 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 AWS et RGPD \(Règlement général sur la protection des données\)](#) sur le AWS Blog de sécurité.

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 des intégrations gérées pour AWS IoT

Device Management ou autre à Services AWS l'aide 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.

Chiffrement des données au repos pour les intégrations gérées

Les intégrations gérées AWS IoT Device Management chiffrent par défaut les données sensibles des clients au repos à l'aide de clés de chiffrement.

Deux types de clés de chiffrement sont utilisés pour protéger les données sensibles des clients utilisant des intégrations gérées :

Clés gérées par le client (CMK)

Les intégrations gérées prennent en charge l'utilisation d'une clé symétrique gérée par le client que vous pouvez créer, posséder et gérer. Vous disposez d'un contrôle total sur ces clés KMS, y compris établir et maintenir leurs politiques de clé, les politiques IAM et les octrois, leur activation et leur désactivation, la rotation de leurs éléments de chiffrement, l'ajout de balises, la création d'alias qui font référence aux clés KMS, et la planification des clés KMS en vue de leur suppression.

AWS clés possédées

Les intégrations gérées utilisent ces clés par défaut pour chiffrer automatiquement les données sensibles des clients. Vous ne pouvez ni consulter, ni gérer, ni auditer leur utilisation. Il n'est pas nécessaire de prendre des mesures ni de modifier de programme pour protéger les clés qui chiffrent vos données. Le chiffrement des données au repos par défaut permet de réduire les frais opérationnels et la complexité liés à la protection des données sensibles. Dans le même temps, il vous permet de créer des applications sécurisées qui répondent aux exigences réglementaires et de conformité strictes en matière de chiffrement.

La clé de chiffrement utilisée par défaut est celle AWS des clés détenues. Sinon, l'API facultative pour mettre à jour votre clé de chiffrement est [PutDefaultEncryptionConfiguration](#).

Pour plus d'informations sur les types de clés de AWS KMS chiffrement, consultez la section [AWS KMS clés](#).

AWS KMS utilisation pour les intégrations gérées

Les intégrations gérées cryptent et décryptent toutes les données des clients à l'aide du chiffrement d'enveloppe. Ce type de chiffrement prend vos données en texte brut et les chiffre à l'aide d'une clé de données. Ensuite, une clé de chiffrement appelée clé d'encapsulation chiffrera la clé de données d'origine utilisée pour chiffrer vos données en texte brut. Dans le cadre du chiffrement d'enveloppe, des clés d'encapsulation supplémentaires peuvent être utilisées pour chiffrer des clés d'encapsulation existantes dont le degré de séparation est plus proche de celui de la clé de données d'origine. La clé de données d'origine étant chiffrée par une clé d'encapsulation stockée séparément, vous pouvez stocker la clé de données d'origine et les données en texte brut cryptées au même endroit. Un trousseau de clés est utilisé pour générer, chiffrer et déchiffrer des clés de données, en plus de la clé d'encapsulation utilisée pour chiffrer et déchiffrer la clé de données.

Note

Le SDK AWS Database Encryption fournit un chiffrement d'enveloppe pour votre implémentation de chiffrement côté client. Pour plus d'informations sur le SDK de chiffrement AWS de base de données, voir [Qu'est-ce que le SDK de chiffrement AWS de base de données ?](#)

Pour plus d'informations sur le chiffrement des enveloppes, les clés de données, les clés d'encapsulation et les trousseaux de clés, voir [Chiffrement des enveloppes](#), [clé de données](#), [clé d'encapsulation](#) et [porte-clés](#).

Les intégrations gérées nécessitent que les services utilisent votre clé gérée par le client pour les opérations internes suivantes :

- Envoyez `DescribeKey` des demandes AWS KMS à pour vérifier que l'identifiant de clé symétrique géré par le client a été fourni lors de la rotation des clés de données.
- Envoyez `GenerateDataKeyWithoutPlaintext` des demandes AWS KMS à pour générer des clés de données chiffrées par votre clé gérée par le client.
- Envoyez `ReEncrypt*` des demandes AWS KMS pour rechiffrer les clés de données à l'aide de votre clé gérée par le client.
- Envoyez `Decrypt` des demandes pour déchiffrer AWS KMS les données à l'aide de la clé gérée par votre client.

Types de données chiffrées à l'aide de clés de chiffrement

Les intégrations gérées utilisent des clés de chiffrement pour chiffrer plusieurs types de données stockées au repos. La liste suivante décrit les types de données chiffrées au repos à l'aide de clés de chiffrement :

- Événements liés au connecteur Cloud-to-Cloud (C2C) tels que la découverte d'appareils et la mise à jour de l'état des appareils.
- Création d'un objet géré qui représente le périphérique physique et d'un profil de périphérique contenant les fonctionnalités d'un type d'appareil spécifique. Pour plus d'informations sur un appareil et son profil, reportez-vous aux sections [Appareil](#) et [Appareil](#).
- Notifications d'intégrations gérées sur divers aspects de la mise en œuvre de votre appareil. Pour plus d'informations sur les notifications d'intégrations gérées, consultez [Configurer les notifications d'intégrations gérées](#).
- Informations personnelles identifiables (PII) d'un utilisateur final, telles que le matériel d'authentification de l'appareil, le numéro de série de l'appareil, le nom de l'utilisateur final, l'identifiant de l'appareil et le nom de ressource Amazon (arn) de l'appareil.

Comment les intégrations gérées utilisent les politiques clés dans AWS KMS

Pour la rotation des clés de branche et les appels asynchrones, les intégrations gérées nécessitent une politique de clé pour utiliser votre clé de chiffrement. Une politique clé est utilisée pour les raisons suivantes :

- Autorisez par programmation l'utilisation d'une clé de chiffrement aux autres AWS principaux.

Pour un exemple de politique de clé utilisée pour gérer l'accès à votre clé de chiffrement dans les intégrations gérées, voir [Création d'une clé de chiffrement](#)

Note

Pour une clé AWS détenue, aucune politique de clé n'est requise car la clé AWS détenue appartient à AWS et vous ne pouvez pas la consulter, la gérer ou l'utiliser. Les intégrations gérées utilisent la clé AWS détenue par défaut pour chiffrer automatiquement les données sensibles de vos clients.

Outre l'utilisation de politiques clés pour gérer votre configuration de chiffrement à l'aide de AWS KMS clés, les intégrations gérées utilisent des politiques IAM. Pour plus d'informations sur les politiques IAM, consultez la section [Politiques et autorisations dans Gestion des identités et des accès AWS](#).

Création d'une clé de chiffrement

Vous pouvez créer une clé de chiffrement à l'aide du AWS Management Console ou du AWS KMS APIs.

Pour créer une clé de chiffrement

Suivez les étapes de [création d'une clé KMS](#) dans le guide du AWS Key Management Service développeur.

Stratégie de clé

Une déclaration de politique clé contrôle l'accès à une AWS KMS clé. Chaque AWS KMS clé contiendra une seule politique clé. Cette politique clé détermine quels AWS principaux peuvent utiliser la clé et comment ils peuvent l'utiliser. Pour plus d'informations sur la gestion de l'accès et de l'utilisation des AWS KMS clés à l'aide de déclarations de politique clés, consultez [la section Gestion de l'accès à l'aide de politiques](#).

Voici un exemple de déclaration de politique clé que vous pouvez utiliser pour gérer l'accès et l'utilisation des AWS KMS clés stockées dans vos Compte AWS intégrations gérées :

```
{
  "Statement" : [
    {
      "Sid" : "Allow access to principals authorized to use managed integrations",
      "Effect" : "Allow",
      "Principal" : {
        //Note: Both role and user are acceptable.
        "AWS" : "arn:aws:iam::111122223333:user/username",
        "AWS" : "arn:aws:iam::111122223333:role/roleName"
      },
      "Action" : [
        "kms:GenerateDataKeyWithoutPlaintext",
        "kms:Decrypt",
        "kms:ReEncrypt*"
      ],
      "Resource" : "arn:aws:kms:region:111122223333:key/key_ID",
```



```

"Condition" : {
  "StringEquals" : {
    "kms:ViaService" : "iotmanagedintegrations.amazonaws.com"
  },
  "ForAnyValue:StringEquals": {
    "kms:EncryptionContext:aws-crypto-ec:iotmanagedintegrations": "111122223333"
  },
  "ArnLike": {
    "aws:SourceArn": [
      "arn:aws:iotmanagedintegrations:<region>:<accountId>:managed-thing/
<managedThingId>",
      "arn:aws:iotmanagedintegrations:<region>:<accountId>:credential-locker/
<credentialLockerId>",
      "arn:aws:iotmanagedintegrations:<region>:<accountId>:provisioning-profile/
<provisioningProfileId>",
      "arn:aws:iotmanagedintegrations:<region>:<accountId>:ota-task/<otaTaskId>"
    ]
  }
},
{
  "Sid" : "Allow access to principals authorized to use managed integrations for
async flow",
  "Effect" : "Allow",
  "Principal" : {
    "Service": "iotmanagedintegrations.amazonaws.com"
  },
  "Action" : [
    "kms:GenerateDataKeyWithoutPlaintext",
    "kms:Decrypt",
    "kms:ReEncrypt*"
  ],
  "Resource" : "arn:aws:kms:region:111122223333:key/key_ID",
  "Condition" : {
    "ForAnyValue:StringEquals": {
      "kms:EncryptionContext:aws-crypto-ec:iotmanagedintegrations": "111122223333"
    },
    "ArnLike": {
      "aws:SourceArn": [
        "arn:aws:iotmanagedintegrations:<region>:<accountId>:managed-thing/
<managedThingId>",
        "arn:aws:iotmanagedintegrations:<region>:<accountId>:credential-locker/
<credentialLockerId>",

```

```

        "arn:aws:iotmanagedintegrations:<region>:<accountId>:provisioning-profile/
<provisioningProfileId>",
        "arn:aws:iotmanagedintegrations:<region>:<accountId>:ota-task/<otaTaskId>"
    ]
}
},
{
    "Sid" : "Allow access to principals authorized to use managed integrations for
describe key",
    "Effect" : "Allow",
    "Principal" : {
        "AWS": "arn:aws:iam::111122223333:user/username"
    },
    "Action" : [
        "kms:DescribeKey",
    ],
    "Resource" : "arn:aws:kms:region:111122223333:key/key_ID",
    "Condition" : {
        "StringEquals" : {
            "kms:ViaService" : "iotmanagedintegrations.amazonaws.com"
        }
    }
},
{
    "Sid": "Allow access for key administrators",
    "Effect": "Allow",
    "Principal": {
        "AWS": "arn:aws:iam::111122223333:root"
    },
    "Action" : [
        "kms:*"
    ],
    "Resource": "*"
}
]
}

```

Pour plus d'informations sur les magasins à clés, consultez la section [Magasins à clés](#).

Mettre à jour la configuration de chiffrement

La possibilité de mettre à jour facilement votre configuration de chiffrement est essentielle pour gérer la mise en œuvre du chiffrement des données pour les intégrations gérées. Lors de votre première intégration avec les intégrations gérées, vous serez invité à sélectionner votre configuration de chiffrement. Vos options seront soit les clés AWS détenues par défaut, soit la création de votre propre AWS KMS clé.

AWS Management Console

Pour mettre à jour votre configuration de chiffrement dans le AWS Management Console, ouvrez la page d'accueil du AWS IoT service, puis accédez à Managed Integration for Unified Control > Paramètres > Chiffrement. Dans la fenêtre Paramètres de chiffrement, vous pouvez mettre à jour votre configuration de chiffrement en sélectionnant une nouvelle AWS KMS clé pour une protection de chiffrement supplémentaire. Choisissez Personnaliser les paramètres de chiffrement (avancés) pour sélectionner une AWS KMS clé existante ou vous pouvez choisir Créer une AWS KMS clé pour créer votre propre clé gérée par le client.

Commandes d'API

Il en existe deux APIs pour gérer la configuration de chiffrement des AWS KMS clés dans les intégrations gérées : `PutDefaultEncryptionConfiguration` et `GetDefaultEncryptionConfiguration`.

Pour mettre à jour la configuration de chiffrement par défaut, appelez `PutDefaultEncryptionConfiguration`. Pour plus d'informations sur `PutDefaultEncryptionConfiguration`, consultez [PutDefaultEncryptionConfiguration](#).

Pour consulter la configuration de chiffrement par défaut, appelez `GetDefaultEncryptionConfiguration`. Pour plus d'informations sur `GetDefaultEncryptionConfiguration`, consultez [GetDefaultEncryptionConfiguration](#).

Gestion des identités et des accès pour les intégrations gérées

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 les ressources d'intégrations gérées. IAM est un Service AWS outil que vous pouvez utiliser sans frais supplémentaires.

Rubriques

- [Public ciblé](#)
- [Authentification par des identités](#)
- [Gestion de l'accès à l'aide de politiques](#)
- [AWS politiques gérées pour les intégrations gérées](#)
- [Comment fonctionnent les intégrations gérées avec IAM](#)
- [Exemples de politiques basées sur l'identité pour les intégrations gérées](#)
- [Résolution des problèmes liés à l'identité et à l'accès aux intégrations gérées](#)
- [Utilisation de rôles liés à un service pour les intégrations gérées](#)

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 liés à l'identité et à l'accès aux intégrations gérées](#))
- Administrateur du service : déterminez l'accès des utilisateurs et soumettez les demandes d'autorisation (voir [Comment fonctionnent les intégrations gérées avec IAM](#))
- Administrateur IAM : rédigez des politiques pour gérer l'accès (voir [Exemples de politiques basées sur l'identité pour les intégrations gérées](#))

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.

Identité fédérée

Il est recommandé d'obliger les utilisateurs humains à utiliser la fédération avec un fournisseur d'identité pour accéder à Services AWS l'aide d'informations d'identification temporaires.

Une identité fédérée est un utilisateur provenant de l'annuaire de votre entreprise, de votre fournisseur d'identité Web ou Directory Service qui y accède à Services AWS l'aide d'informations d'identification provenant d'une source d'identité. Les identités fédérées assument des rôles qui fournissent des informations d'identification temporaires.

Pour une gestion des accès centralisée, nous vous recommandons d'utiliser AWS IAM Identity Center. Pour plus d'informations, consultez [Qu'est-ce que IAM Identity Center ?](#) dans le Guide de l'utilisateur AWS IAM Identity Center .

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 entre comptes, les accès entre services et pour 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.

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.

AWS politiques gérées pour les intégrations gérées

Pour ajouter des autorisations aux utilisateurs, aux groupes et aux rôles, il est plus facile d'utiliser des politiques AWS gérées que de les rédiger vous-même. Il faut du temps et de l'expertise pour [créer des politiques gérées par le client IAM](#) qui ne fournissent à votre équipe que les autorisations dont elle a besoin. Pour démarrer rapidement, vous pouvez utiliser nos politiques AWS gérées. Ces politiques couvrent des cas d'utilisation courants et sont disponibles dans votre Compte AWS. Pour plus d'informations sur les politiques AWS gérées, voir les [politiques AWS gérées](#) dans le guide de l'utilisateur IAM.

AWS les services maintiennent et mettent à jour les politiques AWS gérées. Vous ne pouvez pas modifier les autorisations dans les politiques AWS gérées. Les services ajoutent occasionnellement des autorisations à une politique gérée par AWS pour prendre en charge de nouvelles fonctionnalités. Ce type de mise à jour affecte toutes les identités (utilisateurs, groupes et rôles) auxquelles la politique est attachée. Les services sont très susceptibles de mettre à jour une politique gérée par AWS quand une nouvelle fonctionnalité est lancée ou quand de nouvelles opérations sont disponibles. Les services ne suppriment pas les autorisations d'une politique AWS gérée. Les mises à jour des politiques n'endommageront donc pas vos autorisations existantes.

En outre, AWS prend en charge les politiques gérées pour les fonctions professionnelles qui couvrent plusieurs services. Par exemple, la politique ReadOnlyAccess AWS gérée fournit un accès en lecture seule à tous les AWS services et ressources. Lorsqu'un service lance une nouvelle fonctionnalité, il AWS ajoute des autorisations en lecture seule pour les nouvelles opérations et ressources. Pour obtenir la liste des politiques de fonctions professionnelles et leurs descriptions, consultez la page [politiques gérées par AWS pour les fonctions de tâche](#) dans le Guide de l'utilisateur IAM.

AWS politique gérée : AWSIoTManagedIntegrationsFullAccess

Vous pouvez associer la politique AWSIoTManagedIntegrationsFullAccess à vos identités IAM.

Cette politique accorde des autorisations d'accès complètes aux intégrations gérées et aux services associés. Pour consulter cette politique dans le AWS Management Console, voir [AWSIoTManagedIntegrationsFullAccess](#).

Détails de l'autorisation

Cette politique inclut les autorisations suivantes :

- **iotmanagedintegrations**— Fournit un accès complet aux intégrations gérées et aux services associés aux utilisateurs, groupes et rôles IAM auxquels vous ajoutez cette politique.
- **iam**— Permet aux utilisateurs, groupes et rôles IAM assignés de créer un rôle lié à un service dans un. Compte AWS

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iotmanagedintegrations:*",
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": "iam:CreateServiceLinkedRole",
      "Resource": "arn:aws:iam::*:role/aws-service-role/iotmanagedintegrations.amazonaws.com/AWSServiceRoleForIoTManagedIntegrations",
      "Condition": {
        "StringEquals": {
          "iam:AWSServiceName": "iotmanagedintegrations.amazonaws.com"
        }
      }
    }
  ]
}
```

AWS politique gérée : AWS IoTManagedIntegrationsRolePolicy

Vous pouvez associer la politique AWS IoTManagedIntegrationsRolePolicy à vos identités IAM.

Cette politique autorise les intégrations gérées à publier les CloudWatch journaux et les statistiques Amazon en votre nom.

Pour consulter cette politique dans le AWS Management Console, voir

[AWSIoTManagedIntegrationsRolePolicy](#).

Détails de l'autorisation

Cette politique inclut les autorisations suivantes.

- **logs**— Permet de créer des groupes de CloudWatch journaux Amazon et de diffuser des journaux vers les groupes.
- **cloudwatch**— Permet de publier des CloudWatch statistiques Amazon. Pour plus d'informations sur CloudWatch les métriques Amazon, consultez [Metrics in Amazon CloudWatch](#).

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "CloudWatchLogs",
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup"
      ],
      "Resource": [
        "arn:aws:logs:*:*:log-group:/aws/iotmanagedintegrations/*"
      ],
      "Condition": {
        "StringEquals": {
          "aws:PrincipalAccount": "${aws:ResourceAccount}"
        }
      }
    }
  ],
}
```

```

{
  "Sid": "CloudWatchStreams",
  "Effect": "Allow",
  "Action": [
    "logs:CreateLogStream",
    "logs:PutLogEvents"
  ],
  "Resource": [
    "arn:aws:logs:*:*:log-group:/aws/iotmanagedintegrations/*:log-stream:*"
  ],
  "Condition": {
    "StringEquals": {
      "aws:PrincipalAccount": "${aws:ResourceAccount}"
    }
  }
},
{
  "Sid": "CloudWatchMetrics",
  "Effect": "Allow",
  "Action": [
    "cloudwatch:PutMetricData"
  ],
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "cloudwatch:namespace": [
        "AWS/IoTManagedIntegrations",
        "AWS/Usage"
      ]
    }
  }
}
]
}

```

Intégrations gérées : mises à jour des politiques AWS gérées

Consultez les détails des mises à jour des politiques AWS gérées pour les intégrations gérées depuis que 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 des documents des intégrations gérées.

Modifier	Description	Date
Les intégrations gérées ont commencé à suivre les modifications	Managed Integrations a commencé à suivre les modifications apportées à ses politiques AWS gérées.	3 mars 2025

Comment fonctionnent les intégrations gérées avec IAM

Avant d'utiliser IAM pour gérer l'accès aux intégrations gérées, découvrez quelles fonctionnalités IAM peuvent être utilisées avec les intégrations gérées.

Fonctionnalités IAM que vous pouvez utiliser avec les intégrations gérées

Fonctionnalité IAM	Support des intégrations gérées
Politiques basées sur l'identité	Oui
Politiques basées sur les ressources	Non
Actions de politique	Oui
Ressources de politique	Oui
Clés de condition de politique	Oui
ACLs	Non
ABAC (étiquettes dans les politiques)	Non
Informations d'identification temporaires	Oui
Autorisations de principal	Oui
Rôles de service	Oui
Rôles liés à un service	Oui

Pour obtenir une vue d'ensemble du fonctionnement des intégrations gérées et des autres AWS services avec la plupart des fonctionnalités IAM, consultez la section [AWS Services compatibles avec IAM dans le Guide de l'utilisateur IAM](#).

Politiques basées sur l'identité pour les intégrations gérées

Prend en charge les politiques basées sur l'identité : oui

Les politiques basées sur l'identité sont des documents de politique d'autorisations JSON que vous pouvez attacher à une identité telle qu'un utilisateur, un groupe d'utilisateurs ou un rôle IAM. Ces politiques contrôlent quel type d'actions des utilisateurs et des rôles peuvent exécuter, sur quelles ressources et dans quelles conditions. Pour découvrir comment créer une politique basée sur l'identité, consultez [Définition d'autorisations IAM personnalisées avec des politiques gérées par le client](#) dans le Guide de l'utilisateur IAM.

Avec les politiques IAM basées sur l'identité, vous pouvez spécifier des actions et ressources autorisées ou refusées, ainsi que les conditions dans lesquelles les actions sont autorisées ou refusées. Pour découvrir tous les éléments que vous utilisez dans une politique JSON, consultez [Références des éléments de politique JSON IAM](#) dans le Guide de l'utilisateur IAM.

Exemples de politiques basées sur l'identité pour les intégrations gérées

Pour consulter des exemples d'intégrations gérées et de politiques basées sur l'identité, consultez. [Exemples de politiques basées sur l'identité pour les intégrations gérées](#)

Politiques basées sur les ressources dans le cadre des intégrations gérées

Prend en charge les politiques basées sur les ressources : non

Les politiques basées sur les ressources sont des documents de politique JSON que vous attachez à une ressource. Par exemple, les politiques de confiance de rôle IAM et les politiques de compartiment Amazon S3 sont des politiques basées sur les ressources. Dans les services qui sont compatibles avec les politiques basées sur les ressources, les administrateurs de service peuvent les utiliser pour contrôler l'accès à une ressource spécifique. Pour la ressource dans laquelle se trouve la politique, cette dernière définit quel type d'actions un principal spécifié peut effectuer sur cette ressource et dans quelles conditions. Vous devez [spécifier un principal](#) dans une politique basée sur les ressources. Les principaux peuvent inclure des comptes, des utilisateurs, des rôles, des utilisateurs fédérés ou. Services AWS

Pour permettre un accès intercompte, vous pouvez spécifier un compte entier ou des entités IAM dans un autre compte en tant que principal dans une politique basée sur les ressources. Pour plus d'informations, consultez [Accès intercompte aux ressources dans IAM](#) dans le Guide de l'utilisateur IAM.

Actions politiques pour les intégrations gérées

Prend en charge les actions de politique : oui

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.

Pour consulter la liste des actions d'intégrations gérées, consultez la section [Actions définies par les intégrations gérées](#) dans la référence d'autorisation de service.

Les actions de stratégie dans les intégrations gérées utilisent le préfixe suivant avant l'action :

```
iot-mi
```

Pour indiquer plusieurs actions dans une seule déclaration, séparez-les par des virgules.

```
"Action": [  
  "iot-mi:action1",  
  "iot-mi:action2"  
]
```

Pour consulter des exemples d'intégrations gérées et de politiques basées sur l'identité, consultez [Exemples de politiques basées sur l'identité pour les intégrations gérées](#)

Ressources relatives aux politiques pour les intégrations gérées

Prend en charge les ressources de politique : oui

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": "*" 
```

Pour consulter la liste des types de ressources des intégrations gérées et leurs caractéristiques ARNs, consultez la section [Ressources définies par les intégrations gérées](#) dans la référence d'autorisation de service. Pour savoir avec quelles actions vous pouvez spécifier l'ARN de chaque ressource, consultez la section [Actions définies par les intégrations gérées](#).

Pour consulter des exemples d'intégrations gérées et de politiques basées sur l'identité, consultez [Exemples de politiques basées sur l'identité pour les intégrations gérées](#)

Clés de conditions de politique pour les intégrations gérées

Prend en charge les clés de condition de politique spécifiques au service : oui

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 s'exécutent 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.

Pour consulter la liste des clés de condition des intégrations gérées, consultez la section [Clés de condition pour les intégrations gérées](#) dans la référence d'autorisation de service. Pour savoir avec quelles actions et ressources vous pouvez utiliser une clé de condition, consultez la section [Actions définies par les intégrations gérées](#).

Pour consulter des exemples d'intégrations gérées et de politiques basées sur l'identité, consultez.

[Exemples de politiques basées sur l'identité pour les intégrations gérées](#)

ACLs dans les intégrations gérées

Supports ACLs : Non

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.

ABAC avec intégrations gérées

Prend en charge ABAC (identifications dans les politiques) : partiellement

Le contrôle d'accès par attributs (ABAC) est une stratégie d'autorisation qui définit les autorisations en fonction des attributs nommés balise. Vous pouvez associer des balises aux entités et aux AWS ressources IAM, puis concevoir des politiques ABAC pour autoriser les opérations lorsque la balise du principal correspond à la balise de la ressource.

Pour contrôler l'accès basé sur des étiquettes, vous devez fournir les informations d'étiquette dans [l'élément de condition](#) d'une politique utilisant les clés de condition `aws:ResourceTag/key-name`, `aws:RequestTag/key-name` ou `aws:TagKeys`.

Si un service prend en charge les trois clés de condition pour tous les types de ressources, alors la valeur pour ce service est Oui. Si un service prend en charge les trois clés de condition pour certains types de ressources uniquement, la valeur est Partielle.

Pour plus d'informations sur ABAC, consultez [Définition d'autorisations avec l'autorisation ABAC](#) dans le Guide de l'utilisateur IAM. Pour accéder à un didacticiel décrivant les étapes de configuration de l'ABAC, consultez [Utilisation du contrôle d'accès par attributs \(ABAC\)](#) dans le Guide de l'utilisateur IAM.

Utilisation d'informations d'identification temporaires avec des intégrations gérées

Prend en charge les informations d'identification temporaires : oui

Les informations d'identification temporaires fournissent un accès à court terme aux AWS ressources et sont automatiquement créées lorsque vous utilisez la fédération ou que vous changez de rôle. AWS recommande de générer dynamiquement des informations d'identification temporaires au lieu d'utiliser des clés d'accès à long terme. Pour plus d'informations, consultez [Informations](#)

[d'identification de sécurité temporaires dans IAM](#) et [Services AWS compatibles avec IAM](#) dans le Guide de l'utilisateur IAM.

Autorisations principales interservices pour les intégrations gérées

Prend en charge les sessions d'accès direct (FAS) : oui

Les sessions d'accès direct (FAS) utilisent les autorisations du principal appelant et Service AWS, combinées Service AWS à la demande d'envoi de demandes aux services en aval. Pour plus de détails sur la politique relative à la transmission de demandes FAS, consultez la section [Sessions de transmission d'accès](#).

Rôles de service pour les intégrations gérées

Prend en charge les rôles de service : oui

Un rôle de service est un [rôle IAM](#) qu'un service endosse pour accomplir des actions en votre nom. Un administrateur IAM peut créer, modifier et supprimer un rôle de service à partir d'IAM. Pour plus d'informations, consultez [Création d'un rôle pour la délégation d'autorisations à un Service AWS](#) dans le Guide de l'utilisateur IAM.

Warning

La modification des autorisations associées à un rôle de service peut perturber les fonctionnalités des intégrations gérées. Modifiez les rôles de service uniquement lorsque les intégrations gérées fournissent des instructions à cet effet.

Rôles liés à un service pour les intégrations gérées

Prend en charge les rôles liés à un service : oui

Un rôle lié à un service est un type de rôle de service lié à un Service AWS. Le service peut endosser le rôle afin d'effectuer une action en votre nom. Les rôles liés à un service apparaissent dans votre Compte AWS répertoire et appartiennent au service. Un administrateur IAM peut consulter, mais ne peut pas modifier, les autorisations concernant les rôles liés à un service.

Pour plus d'informations sur la création ou la gestion des rôles liés à un service, consultez [Services AWS qui fonctionnent avec IAM](#). Recherchez un service dans le tableau qui inclut un Yes dans la colonne Rôle lié à un service. Choisissez le lien Oui pour consulter la documentation du rôle lié à ce service.

Exemples de politiques basées sur l'identité pour les intégrations gérées

Par défaut, les utilisateurs et les rôles ne sont pas autorisés à créer ou à modifier les ressources d'intégrations gérées. Pour octroyer aux utilisateurs des autorisations d'effectuer des actions sur les ressources dont ils ont besoin, un administrateur IAM peut créer des politiques IAM.

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 IAM \(console\)](#) dans le Guide de l'utilisateur IAM.

Pour plus de détails sur les actions et les types de ressources définis par les intégrations gérées, y compris le format de ARNs pour chacun des types de ressources, consultez la section [Actions, ressources et clés de condition pour les intégrations gérées](#) dans la référence d'autorisation de service.

Rubriques

- [Bonnes pratiques en matière de politiques](#)
- [Utilisation de la console d'intégrations gérées](#)
- [Autorisation accordée aux utilisateurs pour afficher leurs propres autorisations](#)

Bonnes pratiques en matière de politiques

Les politiques basées sur l'identité déterminent si quelqu'un peut créer, accéder ou supprimer des ressources d'intégrations gérées 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 les AWS clients 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.

Utilisation de la console d'intégrations gérées

Pour accéder à la console des intégrations gérées, vous devez disposer d'un ensemble minimal d'autorisations. Ces autorisations doivent vous permettre de répertorier et d'afficher les détails des ressources d'intégrations gérées dans votre Compte AWS. Si vous créez une politique basée sur l'identité qui est plus restrictive que l'ensemble minimum d'autorisations requis, la console ne fonctionnera pas comme prévu pour les entités (utilisateurs ou rôles) tributaires de cette politique.

Il n'est pas nécessaire d'accorder des autorisations de console minimales aux utilisateurs qui appellent uniquement l'API AWS CLI ou l' AWS API. Autorisez plutôt l'accès à uniquement aux actions qui correspondent à l'opération d'API qu'ils tentent d'effectuer.

Pour garantir que les utilisateurs et les rôles peuvent toujours utiliser la console des intégrations gérées, associez également les intégrations gérées *ConsoleAccess* ou la politique *ReadOnly* AWS gérée aux entités. Pour plus d'informations, consultez [Ajout d'autorisations à un utilisateur](#) 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:ListGroupsWithUser",
        "iam:ListAttachedUserPolicies",
        "iam:ListUserPolicies",
        "iam:GetUser"
      ],
      "Resource": ["arn:aws:iam::*:user/${aws:username}"]
    },
    {
      "Sid": "NavigateInConsole",
      "Effect": "Allow",
      "Action": [
        "iam:GetGroupPolicy",
        "iam:GetPolicyVersion",
        "iam:GetPolicy",
        "iam:ListAttachedGroupPolicies",
        "iam:ListGroupPolicies",
        "iam:ListPolicyVersions",
        "iam:ListPolicies",
        "iam:ListUsers"
      ],
      "Resource": "*"
    }
  ]
}
```

```
]
}
```

Résolution des problèmes liés à l'identité et à l'accès aux intégrations gérées

Utilisez les informations suivantes pour vous aider à diagnostiquer et à résoudre les problèmes courants que vous pouvez rencontrer lors de l'utilisation d'intégrations gérées et d'IAM.

Rubriques

- [Je ne suis pas autorisé à effectuer une action dans les intégrations gérées](#)
- [Je ne suis pas autorisé à effectuer iam : PassRole](#)
- [Je souhaite permettre à des personnes extérieures Compte AWS à moi d'accéder à mes ressources d'intégrations gérées](#)

Je ne suis pas autorisé à effectuer une action dans les intégrations gérées

Si vous recevez une erreur qui indique que vous n'êtes pas autorisé à effectuer une action, vos politiques doivent être mises à jour afin de vous permettre d'effectuer l'action.

L'exemple d'erreur suivant se produit quand l'utilisateur IAM `mateojackson` tente d'utiliser la console pour afficher des informations détaillées sur une ressource *my-example-widget* fictive, mais ne dispose pas des autorisations `iot-mi:GetWidget` fictives.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform: iot-mi:GetWidget on resource: my-example-widget
```

Dans ce cas, la politique qui s'applique à l'utilisateur `mateojackson` doit être mise à jour pour autoriser l'accès à la ressource *my-example-widget* à l'aide de l'action `iot-mi:GetWidget`.

Si vous avez besoin d'aide, contactez votre AWS administrateur. Votre administrateur vous a fourni vos informations d'identification de connexion.

Je ne suis pas autorisé à effectuer iam : PassRole

Si vous recevez un message d'erreur indiquant que vous n'êtes pas autorisé à effectuer l'`iam:PassRole`action, vos politiques doivent être mises à jour pour vous permettre de transmettre un rôle aux intégrations gérées.

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, vous devez 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 effectuer une action dans le cadre d'intégrations gérées. Toutefois, l'action nécessite que le service ait des autorisations accordées par un rôle de service. Mary n'est pas autorisée à transmettre 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 souhaite permettre à des personnes extérieures d'accéder à mes ressources d'intégrations gérées

Vous pouvez créer un rôle que les utilisateurs provenant d'autres comptes ou les personnes extérieures à votre organisation pourront utiliser pour accéder à vos ressources. Vous pouvez spécifier qui est autorisé à assumer le rôle. Pour les services qui prennent en charge les politiques basées sur les ressources ou les listes de contrôle d'accès (ACLs), vous pouvez utiliser ces politiques pour autoriser les utilisateurs à accéder à vos ressources.

Pour plus d'informations, consultez les éléments suivants :

- Pour savoir si les intégrations gérées prennent en charge ces fonctionnalités, consultez [Comment fonctionnent les intégrations gérées avec IAM](#).
- Pour savoir comment fournir l'accès à vos ressources sur celles d'un autre compte AWS que vous possédez, consultez la section [Fournir l'accès à un utilisateur IAM dans un autre utilisateur d'un compte AWS que vous possédez](#) dans le Guide de l'utilisateur IAM.

- Pour savoir comment fournir l'accès à vos ressources à des tiers Comptes AWS, consultez la section [Fournir un accès à des ressources Comptes AWS détenues par des tiers](#) dans le guide de l'utilisateur IAM.
- Pour savoir comment fournir un accès par le biais de la fédération d'identité, consultez [Fournir un accès à des utilisateurs authentifiés en externe \(fédération d'identité\)](#) dans le Guide de l'utilisateur IAM.
- Pour en savoir plus sur la différence entre l'utilisation des rôles et des politiques basées sur les ressources pour l'accès intercompte, consultez [Accès intercompte aux ressources dans IAM](#) dans le Guide de l'utilisateur IAM.

Utilisation de rôles liés à un service pour les intégrations gérées

Les intégrations gérées pour la gestion des AWS IoT appareils utilisent des rôles Gestion des identités et des accès AWS liés aux [services](#) (IAM). Un rôle lié à un service est un type unique de rôle IAM directement lié aux intégrations gérées. Les rôles liés aux services sont prédéfinis par les intégrations gérées et incluent toutes les autorisations dont le service a besoin pour appeler d'autres AWS services en votre nom.

Un rôle lié à un service facilite la configuration des intégrations gérées, car vous n'avez pas à ajouter manuellement les autorisations nécessaires. Les intégrations gérées pour AWS IoT Device Management définissent les autorisations associées à ses rôles liés aux services et, sauf indication contraire, seules les intégrations gérées peuvent assumer ces rôles. Les autorisations définies comprennent la politique de confiance et la politique d'autorisation. De plus, cette politique d'autorisation ne peut pas être attachée à une autre entité IAM.

Vous pouvez supprimer un rôle lié à un service uniquement après la suppression préalable de ses ressources connexes. Cela protège les ressources de vos intégrations gérées, car vous ne pouvez pas supprimer par inadvertance l'autorisation d'accès aux ressources.

Pour plus d'informations sur les autres services qui prennent en charge les rôles liés à un service, consultez la section [AWS Services qui fonctionnent avec IAM](#) et recherchez les services dont la valeur est Oui dans la colonne Rôles liés à un service. Sélectionnez un Oui ayant un lien pour consulter la documentation du rôle lié à un service, pour ce service.

Autorisations de rôle liées à un service pour les intégrations gérées

Les intégrations gérées pour la gestion des AWS IoT appareils utilisent le rôle lié au service nommé `AWSServiceRoleForIoTManagedIntegrations` : fournit des intégrations gérées pour autoriser la gestion des AWS IoT appareils à publier des journaux et des statistiques en votre nom.

Le rôle lié au service `AWSService RoleForIoT Managed Integrations` fait confiance aux services suivants pour assumer ce rôle :

- `iotmanagedintegrations.amazonaws.com`

La politique d'autorisations de rôle nommée `AWSIoTManagedIntegrationsServiceRolePolicy` permet aux intégrations gérées d'effectuer les actions suivantes sur les ressources spécifiées :

- Action : `logs:CreateLogGroup`, `logs:DescribeLogGroups`, `logs:CreateLogStream`, `logs:PutLogEvents`, `logs:DescribeLogStreams`, `cloudwatch:PutMetricData` on all of your managed integrations resources.

JSON

```
{
  "Version": "2012-10-17",
  "Statement" : [
    {
      "Sid" : "CloudWatchLogs",
      "Effect" : "Allow",
      "Action" : [
        "logs:CreateLogGroup",
        "logs:DescribeLogGroups"
      ],
      "Resource" : [
        "arn:aws:logs:*:*:log-group:/aws/iotmanagedintegrations/*"
      ]
    },
    {
      "Sid" : "CloudWatchStreams",
      "Effect" : "Allow",
      "Action" : [
        "logs:CreateLogStream",
        "logs:PutLogEvents",

```



```

    "logs:DescribeLogStreams"
  ],
  "Resource" : [
    "arn:aws:logs:*:*:log-group:/aws/iotmanagedintegrations/*:log-stream:*"
  ]
},
{
  "Sid" : "CloudWatchMetrics",
  "Effect" : "Allow",
  "Action" : [
    "cloudwatch:PutMetricData"
  ],
  "Resource" : "*",
  "Condition" : {
    "StringEquals" : {
      "cloudwatch:namespace" : [
        "AWS/IoTManagedIntegrations",
        "AWS/Usage"
      ]
    }
  }
}
]
}
}

```

Vous devez configurer les autorisations de manière à permettre à vos utilisateurs, groupes ou rôles de créer, modifier ou supprimer un rôle lié à un service. Pour plus d'informations, consultez [Autorisations de rôles liés à un service](#) dans le Guide de l'utilisateur IAM.

Création d'un rôle lié à un service pour les intégrations gérées

Vous n'avez pas besoin de créer manuellement un rôle lié à un service. Lorsque vous provoquez un type d'événement tel que l'`PutRuntimeLogConfiguration` appel de commandes ou d'`RegisterCustomEndpointAPI` dans le AWS Management Console, ou l' AWS API AWS CLI, les intégrations gérées créent le rôle lié au service pour vous. `CreateEventLogConfiguration` Pour plus d'informations sur `PutRuntimeLogConfiguration` `CreateEventLogConfiguration`, ou `RegisterCustomEndpoint`, voir [PutRuntimeLogConfigurationCreateEventLogConfiguration](#), ou [RegisterCustomEndpoint](#).

Si vous supprimez ce rôle lié à un service et que vous avez ensuite besoin de le recréer, vous pouvez utiliser la même procédure pour recréer le rôle dans votre compte. Lorsque vous provoquez

un type d'événement tel que l'appel de commandes ou d'`RegisterCustomEndpointAPI` `PutRuntimeLogConfigurationCreateEventLogConfiguration`, Managed Integrations crée à nouveau le rôle lié au service pour vous. Vous pouvez également contacter AWS le Support client via le AWS Support Center Console. Pour plus d'informations sur les plans de AWS support, consultez [Comparer les plans de support AWS](#).

Vous pouvez également utiliser la console IAM pour créer un rôle lié à un service avec le cas d'utilisation IoT ManagedIntegrations - Managed Role. Dans l'API AWS CLI ou dans l' AWS API, créez un rôle lié à un service avec le nom du `iotmanagedintegrations.amazonaws.com` service. Pour plus d'informations, consultez [Création d'un rôle lié à un service](#) dans le Guide de l'utilisateur IAM. Si vous supprimez ce rôle lié à un service, vous pouvez utiliser ce même processus pour créer le rôle à nouveau.

Modification d'un rôle lié à un service pour les intégrations gérées

les intégrations gérées ne vous permettent pas de modifier le rôle lié au service `AWSServiceRoleForIoTManagedIntegrations`. Après avoir créé un rôle lié à un service, vous ne pouvez pas changer le nom du rôle, car plusieurs entités peuvent faire référence à ce rôle. Néanmoins, vous pouvez modifier la description du rôle à l'aide d'IAM. Pour plus d'informations, consultez [Modification d'un rôle lié à un service](#) dans le Guide de l'utilisateur IAM.

Supprimer un rôle lié à un service pour les intégrations gérées

Si vous n'avez plus besoin d'utiliser une fonctionnalité ou un service qui nécessite un rôle lié à un service, nous vous recommandons de supprimer ce rôle. De cette façon, vous n'avez aucune entité inutilisée qui n'est pas surveillée ou gérée activement. Cependant, vous devez nettoyer les ressources de votre rôle lié à un service avant de pouvoir les supprimer manuellement.

Note

Si les intégrations gérées utilisent le rôle lorsque vous essayez de supprimer les ressources, la suppression risque d'échouer. Si cela se produit, patientez quelques minutes et réessayez.

Pour supprimer manuellement le rôle lié au service à l'aide d'IAM

Utilisez la console IAM AWS CLI, le ou l' AWS API pour supprimer le rôle lié au service `AWSServiceRoleForIoTManagedIntegrations`. Pour plus d'informations, consultez la section [Suppression d'un rôle lié à un service](#) dans le Guide de l'utilisateur IAM.

Régions prises en charge pour les intégrations gérées, les rôles liés aux services

Les intégrations gérées pour la gestion des AWS IoT appareils prennent en charge l'utilisation de rôles liés au service dans toutes les régions où le service est disponible. Pour plus d'informations, consultez [AWS Régions et points de terminaison](#).

Utilisation à des AWS Secrets Manager fins de protection des données pour les flux de travail C2C

AWS Secrets Manager est un service de stockage secret que vous pouvez utiliser pour protéger les informations d'identification de base de données, les clés d'API et d'autres informations secrètes. Ensuite, dans votre code, vous pouvez remplacer les informations d'identification codées en dur par un appel d'API à Secrets Manager. Cela permet de s'assurer que le secret ne peut pas être compromis par quelqu'un qui examine votre code, car le secret n'existe pas. Pour une présentation, consultez le [Guide de l'utilisateur AWS Secrets Manager](#).

Secrets Manager chiffre les secrets à l'aide de AWS Key Management Service clés. Pour plus d'informations, veuillez consulter [Chiffrement et déchiffrement de secret dans AWS Key Management Service](#).

Intégrations gérées AWS Secrets Manager pour les AWS IoT Device Management intégrations qui vous permettent de stocker vos données dans Secrets Manager et d'utiliser l'identifiant secret dans vos configurations.

Comment les intégrations gérées utilisent les secrets

Open Authorization (OAuth) est une norme ouverte d'autorisation d'accès déléguée, qui permet aux utilisateurs d'autoriser des sites Web ou des applications à accéder à leurs informations sur d'autres sites Web sans partager leurs mots de passe. Il s'agit d'un moyen sécurisé pour les applications tierces d'accéder aux données utilisateur au nom de l'utilisateur, offrant ainsi une alternative plus sûre au partage de mots de passe.

Dans OAuth, un identifiant client et un secret client sont des informations d'identification qui identifient et authentifient une application cliente lorsqu'elle demande un jeton d'accès.

Intégrations gérées permettant OAuth aux AWS IoT Device Management utilisateurs de communiquer avec les clients qui utilisent les flux de travail C2C. Les clients doivent fournir l'identifiant du client et le secret du client pour communiquer. Intégrations gérées Les clients

stockeront un identifiant client et un secret client dans leurs AWS comptes, et les intégrations gérées liront l'identifiant client et le secret client dans notre compte client.

Comment créer un secret

Pour créer un secret, suivez les étapes décrites dans [Créer un AWS Secrets Manager secret](#) dans le Guide de AWS Secrets Manager l'utilisateur.

Vous devez créer votre secret à l'aide d'une AWS KMS clé gérée par le client pour que les intégrations gérées puissent lire la valeur secrète. Pour plus d'informations, consultez la section [Autorisations relatives à la AWS KMS clé](#) dans le guide de AWS Secrets Manager l'utilisateur.

Vous devez également utiliser les politiques IAM décrites dans la section suivante.

Accordez l'accès aux intégrations gérées AWS IoT Device Management afin de récupérer le secret

Pour permettre aux intégrations gérées de récupérer la valeur secrète depuis Secrets Manager, incluez les autorisations suivantes dans la politique de ressources pour le secret lorsque vous le créez.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [ {
    "Effect": "Allow",
    "Principal": {
      "Service": "iotmanagedintegrations.amazonaws.com"
    },
    "Action": [ "secretsmanager:GetSecretValue" ],
    "Resource": "*",
    "Condition": {
      "StringEquals": {
        "aws:SourceArn": "arn:aws:iotmanagedintegrations:AWS Region:account-id:account-association:account-association-id"
      }
    }
  } ]
}
```

```
}
```

Ajoutez la déclaration suivante à la politique relative à votre clé gérée par le client AWS KMS .

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": [
      "kms:Decrypt",
      "kms:DescribeKey"
    ],
    "Principal": {
      "Service": [
        "iotmanagedintegrations.amazonaws.com"
      ]
    },
    "Resource": [
      "arn:aws:kms:us-east-1:123456789012:key/*"
    ]
  }
]
```

Validation de conformité pour les intégrations gérées

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](#).

Utilisez des intégrations gérées avec les points de terminaison VPC d'interface

Vous pouvez établir une connexion privée entre votre Amazon VPC et les intégrations AWS IoT gérées en créant une interface Amazon VPC endpoint. Les points de terminaison d'interface sont AWS PrivateLink alimentés par une technologie qui vous permet d'accéder à des services de manière privée en utilisant des adresses IP privées. AWS PrivateLink restreint tout le trafic réseau entre votre VPC et les intégrations gérées par l'IoT vers le réseau Amazon. Vous n'avez pas besoin d'une passerelle Internet, d'un appareil NAT ou d'une connexion VPN.

Vous n'êtes pas obligé de l'utiliser AWS PrivateLink, mais c'est recommandé. Pour plus d'informations sur AWS PrivateLink les points de terminaison VPC, consultez la section [Accès aux AWS services via le Guide AWS PrivateLink](#).AWS PrivateLink

Rubriques

- [Considérations relatives aux intégrations AWS IoT gérées \(points de terminaison VPC\)](#)
- [Création d'un point de terminaison VPC d'interface pour les intégrations gérées AWS IoT](#)
- [Test de votre point de terminaison VPC](#)
- [Contrôle de l'accès aux services via les points de terminaison VPC](#)
- [Tarification](#)
- [Limitations](#)

Considérations relatives aux intégrations AWS IoT gérées (points de terminaison VPC)

Avant de configurer un point de terminaison VPC d'interface pour les intégrations AWS IoT gérées, consultez les [propriétés et les limites du point de terminaison d'interface dans le Guide](#).AWS PrivateLink

AWS IoT Les intégrations gérées permettent d'appeler toutes ses actions d'API depuis votre VPC via les points de terminaison VPC de l'interface.

Points de terminaison pris en charge

AWS IoT Les intégrations gérées prennent en charge les points de terminaison VPC pour les interfaces de service suivantes :

- API du plan de contrôle : `com.amazonaws.region.iotmanagedintegrations.api`

Terminaux non pris en charge

Les points de terminaison d'intégrations AWS IoT gérées suivants ne prennent pas en charge les points de terminaison VPC :

- Points de terminaison MQTT : les appareils MQTT sont généralement déployés dans les environnements des utilisateurs finaux plutôt qu'à l'intérieur AWS VPCs, ce qui rend l'intégration inutile. AWS PrivateLink
- OAuth points de terminaison de rappel : de nombreuses plateformes tierces ne fonctionnent pas au sein de AWS l'infrastructure, ce qui réduit les avantages de la prise en AWS PrivateLink charge des OAuth flux.

Disponibilité

AWS IoT Les points de terminaison VPC des intégrations gérées sont disponibles dans les régions suivantes : AWS

- Canada (Centre) – ca-central-1
- Europe (Irlande) – eu-west-1

D'autres régions seront prises en charge à mesure que AWS IoT Managed Integrations étendra sa disponibilité.

Support à double pile

AWS IoT Intégrations gérées Les points de terminaison VPC prennent en charge IPv4 à la fois le trafic et le trafic. IPv6 Vous pouvez créer des points de terminaison VPC avec les types d'adresses IP suivants :

- IPv4: attribue des IPv4 adresses aux interfaces réseau des terminaux

- IPv6: attribue des IPv6 adresses aux interfaces réseau des terminaux (nécessite des sous-réseaux IPv6 uniquement)
- Dualstack : attribue à la fois des IPv6 adresses IPv4 et des adresses aux interfaces réseau des terminaux

Création d'un point de terminaison VPC d'interface pour les intégrations gérées AWS IoT

Vous pouvez créer un point de terminaison VPC pour le service d'intégrations AWS IoT gérées à l'aide de la console Amazon VPC ou de la (CLI). AWS CLI AWS

Pour créer un point de terminaison VPC d'interface pour les intégrations AWS IoT gérées (console)

1. Ouvrez la console Amazon VPC sur Amazon [VPC](#) Console.
2. Dans le panneau de navigation, choisissez Points de terminaison.
3. Choisissez Créer un point de terminaison.
4. Pour Service category (Catégorie de service), choisissez Services AWS .
5. Pour Nom du service, sélectionnez le nom du service correspondant à votre AWS région. Par exemple :
 - `com.amazonaws.ca-central-1.iotmanagedintegrations.api`
 - `com.amazonaws.eu-west-1.iotmanagedintegrations.api`
6. Pour le VPC, sélectionnez le VPC à partir duquel vous allez accéder aux intégrations gérées. AWS IoT
7. Pour les paramètres supplémentaires, l'option Activer le nom DNS est sélectionnée par défaut. Nous vous recommandons de conserver ce paramètre. Cela garantit que les demandes adressées aux points de terminaison du service public des intégrations AWS IoT gérées sont résolues vers votre point de terminaison Amazon VPC.
8. Pour les sous-réseaux, sélectionnez les sous-réseaux dans lesquels vous souhaitez créer les interfaces réseau des points de terminaison. Vous pouvez sélectionner un sous-réseau par zone de disponibilité.
9. Pour IP address type (Type d'adresse IP), choisissez l'une des options suivantes :
 - IPv4: Attribuer IPv4 des adresses aux interfaces réseau des terminaux

- IPv6: Attribuer IPv6 des adresses aux interfaces réseau des points de terminaison (prise en charge uniquement si tous les sous-réseaux sélectionnés sont IPv6 uniquement disponibles)
- Dualstack : Attribuez à la fois des IPv6 adresses IPv4 et des adresses aux interfaces réseau des terminaux

10 Pour Security groups (Groupes de sécurité), sélectionnez les groupes de sécurité à associer aux interfaces réseau du point de terminaison. Les règles du groupe de sécurité doivent autoriser la communication entre l'interface réseau du point de terminaison et les ressources de votre VPC qui communiquent avec le service.

11 Pour Policy, choisissez Accès complet pour autoriser toutes les opérations effectuées par tous les principaux sur toutes les ressources via le point de terminaison de l'interface. Pour restreindre l'accès, choisissez Personnalisé et spécifiez une politique.

12 (Facultatif) Pour ajouter une identification, choisissez Add new tag (Ajouter une identification) et saisissez la clé et la valeur de l'identification.

13 Choisissez Créer un point de terminaison.

Pour créer un point de terminaison VPC d'interface pour les intégrations gérées par l'IoT (AWS CLI)

Utilisez la [create-vpc-endpoint](#) commande et spécifiez l'ID du VPC, le type de point de terminaison du VPC (interface), le nom du service, les sous-réseaux qui utiliseront le point de terminaison et les groupes de sécurité à associer aux interfaces réseau du point de terminaison.

```
aws ec2 create-vpc-endpoint \  
  --vpc-id vpc-12345678 \  
  --route-table-ids rtb-12345678 \  
  --service-name com.amazonaws.ca-central-1.iotmanagedintegrations.api \  
  --vpc-endpoint-type Interface \  
  --subnet-ids subnet-12345678 subnet-87654321 \  
  --security-group-ids sg-12345678
```

Test de votre point de terminaison VPC

Après avoir créé votre point de terminaison VPC, vous pouvez tester la connexion en effectuant des appels d'API vers AWS IoT Managed Integrations depuis une EC2 instance de votre VPC.

Prérequis

- Une EC2 instance dans un sous-réseau privé au sein de votre VPC
- Autorisations IAM appropriées pour les opérations d'intégrations AWS IoT gérées
- Règles de groupe de sécurité qui autorisent le trafic HTTPS (port 443) vers le point de terminaison du VPC

Test de la connexion

1. Connectez-vous à votre EC2 instance Amazon dans le sous-réseau privé.
2. Vérifiez la résolution DNS pour le nom DNS privé :

```
dig api.iotmanagedintegrations.region.api.aws
```

3. Testez la connectivité HTTPS :

```
curl -v https://api.iotmanagedintegrations.region.api.aws
```

4. Passez un appel à l'API d'intégrations AWS IoT gérées :

```
aws iot-managed-integrations list-destinations \  
  --region region \  
  --endpoint-url https://api.iotmanagedintegrations.region.api.aws
```

Remplacez `region` par votre AWS région (par exemple, `ca-central-1`).

Contrôle de l'accès aux services via les points de terminaison VPC

Une politique de point de terminaison VPC est une politique de ressources IAM que vous attachez à un point de terminaison VPC d'interface lorsque vous créez ou modifiez le point de terminaison. Si vous ne définissez pas de politique lorsque vous créez un point de terminaison, nous définissons une politique par défaut pour vous, qui autorise un accès total au service. Une stratégie de point de terminaison n'annule pas et ne remplace pas les stratégies utilisateur IAM ou les stratégies propres au service. Il s'agit d'une politique distincte qui contrôle l'accès depuis le point de terminaison jusqu'au service spécifié.

Les politiques de point de terminaison doivent être écrites au format JSON. Pour de plus amples informations, veuillez consulter [Contrôle de l'accès aux services avec points de terminaison d'un VPC](#) dans le Amazon VPC Guide de l'utilisateur.

Exemple : politique de point de terminaison VPC pour les actions d'intégrations AWS IoT gérées

Voici un exemple de politique de point de terminaison pour les intégrations AWS IoT gérées. Cette politique permet aux utilisateurs qui se connectent à des intégrations AWS IoT gérées via le point de terminaison VPC d'accéder à des destinations, mais refuse l'accès aux casiers d'informations d'identification.

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": "*",
      "Action": [
        "iotmanagedintegrations:ListDestinations",
        "iotmanagedintegrations:GetDestination",
        "iotmanagedintegrations:CreateDestination",
        "iotmanagedintegrations:UpdateDestination",
        "iotmanagedintegrations>DeleteDestination"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Deny",
      "Principal": "*",
      "Action": [
        "iotmanagedintegrations:ListCredentialLockers",
        "iotmanagedintegrations:GetCredentialLocker",
        "iotmanagedintegrations:CreateCredentialLocker",
        "iotmanagedintegrations:UpdateCredentialLocker",
        "iotmanagedintegrations>DeleteCredentialLocker"
      ],
      "Resource": "*"
    }
  ]
}
```

Exemple : politique de point de terminaison VPC qui restreint l'accès à un rôle IAM spécifique

La politique de point de terminaison VPC suivante autorise l'accès aux intégrations AWS IoT gérées uniquement aux principaux IAM qui ont le rôle IAM spécifié dans leur chaîne de confiance. Tous les autres principaux IAM se voient refuser l'accès.

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": "*",
      "Action": "*",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:PrincipalArn": "arn:aws:iam::123456789012:role/IoTManagedIntegrationsVPCRole"
        }
      }
    }
  ]
}
```

Tarification

Des tarifs standard vous sont facturés pour la création et l'utilisation d'un point de terminaison VPC d'interface avec des intégrations AWS IoT gérées. Pour en savoir plus, consultez [Pricing AWS PrivateLink](#) (Tarification).

Limitations

- L'[CreateAccountAssociation](#) API est conçue pour fonctionner OAuth avec des services cloud tiers, ce qui nécessite que la demande quitte le réseau Amazon. Cela est important pour les clients qui AWS PrivateLink souhaitent contenir leur trafic au sein du VPC, car ils AWS PrivateLink ne peuvent pas fournir un end-to-end confinement complet pour cet appel d'API.
- Les points de terminaison VPC pour les intégrations AWS IoT gérées ne sont pas disponibles dans. AWS GovCloud (US) Regions

Pour connaître les limites générales des points de terminaison VPC, consultez la section [Propriétés et limites des points de terminaison d'interface](#) dans le guide de l'utilisateur Amazon VPC.

Connectez-vous à des intégrations gérées pour les points de terminaison AWS IoT Device Management FIPS

AWS IoT fournit un point de terminaison du plan de contrôle compatible avec la [norme fédérale de traitement de l'information \(FIPS\) 140-2](#). Les points de terminaison conformes à la norme FIPS sont différents des points de terminaison standard. AWS Pour interagir avec les intégrations gérées conformément AWS IoT Device Management à la norme FIPS, vous devez utiliser les points de terminaison décrits ci-dessous avec votre client conforme à la norme FIPS. La AWS IoT console n'est pas conforme à la norme FIPS.

Les sections suivantes décrivent comment accéder au point de AWS IoT terminaison conforme à la norme FIPS à l'aide de l'API REST, d'un SDK ou du AWS CLI

Points de terminaison du plan de contrôle

Les points de terminaison du plan de contrôle conformes à la norme FIPS qui prennent en charge les opérations d'intégration gérées et leurs AWS CLI commandes associées sont répertoriés dans la section Points de terminaison [FIPS](#) par service. Dans [FIPS Endpoints by Service](#), recherchez le service AWS IoT Device Management - Intégrations gérées et recherchez le point de terminaison correspondant à votre Région AWS

Pour utiliser le point de terminaison conforme à la norme FIPS lorsque vous accédez aux opérations d'intégration gérées, utilisez le AWS SDK ou l'API REST avec le point de terminaison qui vous convient. Région AWS

Pour utiliser le point de terminaison conforme à la norme FIPS lorsque vous exécutez des commandes CLI d'intégrations gérées, ajoutez le `--endpoint` paramètre avec le point de terminaison approprié pour vous Région AWS à la commande.

Surveillance des intégrations gérées

La surveillance joue un rôle important dans le maintien de la fiabilité, de la disponibilité et des performances des intégrations gérées et de vos autres solutions AWS. AWS fournit les outils de surveillance suivants pour surveiller les intégrations gérées, signaler un problème et prendre des mesures automatiques le cas échéant :

- AWS CloudTrail capture les appels d'API et les événements associés effectués par ou pour le compte de votre AWS compte et envoie les fichiers journaux dans un compartiment Amazon S3 que vous spécifiez. Vous pouvez identifier les utilisateurs et les comptes appelés AWS, l'adresse IP source à partir de laquelle les appels ont été effectués et la date des appels. Pour de plus amples informations, veuillez consulter le [Guide de l'utilisateur AWS CloudTrail](#).

Journalisation des appels d'API d'intégrations gérées à l'aide de AWS CloudTrail

Les intégrations gérées sont intégrées à [AWS CloudTrail](#) un service qui fournit un enregistrement des actions entreprises par un utilisateur, un rôle ou un Service AWS. CloudTrail capture tous les appels d'API pour les intégrations gérées sous forme d'événements. Les appels capturés incluent des appels provenant de la console des intégrations gérées et des appels de code vers les opérations de l'API des intégrations gérées. À l'aide des informations collectées par CloudTrail, vous pouvez déterminer la demande envoyée aux intégrations gérées, l'adresse IP à partir de laquelle la demande a été faite, la date à laquelle elle a été faite et des informations supplémentaires.

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 :

- Si la demande a été effectuée avec des informations d'identification d'utilisateur root ou d'utilisateur root.
- Si la demande a été faite au nom d'un utilisateur du centre d'identité 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 requête a été effectuée par un autre Service AWS.

CloudTrail est actif dans votre compte Compte AWS lorsque vous créez le compte et vous avez automatiquement accès à l'historique des CloudTrail événements. L'historique des CloudTrail événements fournit un enregistrement consultable, consultable, téléchargeable et immuable des 90 derniers jours des événements de gestion enregistrés dans un. Région AWS Pour plus d'informations, consultez la section [Utilisation de l'historique des CloudTrail événements](#) dans le guide de AWS CloudTrail l'utilisateur. La consultation de CloudTrail l'historique des événements est gratuite.

Pour un enregistrement continu des événements de vos 90 Compte AWS derniers jours, créez un magasin de données sur les événements de Trail ou [CloudTrailLake](#).

CloudTrail sentiers

Un suivi permet CloudTrail de fournir des fichiers journaux à un compartiment Amazon S3. Tous les sentiers créés à l'aide du AWS Management Console sont multirégionaux. Vous ne pouvez créer un journal de suivi en une ou plusieurs régions à l'aide de l' AWS CLI. Il est recommandé de créer un parcours multirégional, car vous capturez l'activité dans l'ensemble Régions AWS de votre compte. Si vous créez un journal de suivi pour une seule région, il convient de n'afficher que les événements enregistrés dans le journal de suivi pour une seule région Région AWS. Pour plus d'informations sur les journaux de suivi, consultez [Créez un journal de suivi dans vos Compte AWS](#) et [Création d'un journal de suivi pour une organisation](#) dans le AWS CloudTrail Guide de l'utilisateur.

Vous pouvez envoyer une copie de vos événements de gestion en cours dans votre compartiment Amazon S3 gratuitement CloudTrail en créant un journal. Toutefois, des frais de stockage Amazon S3 sont facturés. Pour plus d'informations sur la CloudTrail tarification, consultez la section [AWS CloudTrail Tarification](#). Pour obtenir des informations sur la tarification Amazon S3, consultez [Tarification Amazon S3](#).

CloudTrail Stockages de données sur les événements du lac

CloudTrail Lake vous permet d'exécuter des requêtes SQL sur vos événements. CloudTrail Lake convertit les événements existants au format JSON basé sur les lignes au format [Apache ORC](#). ORC est un format de stockage en colonnes qui est optimisé pour une récupération rapide des données. Les événements sont agrégés dans des magasins de données d'événement. Ceux-ci constituent des collections immuables d'événements basées sur des critères que vous sélectionnez en appliquant des [sélecteurs d'événements avancés](#). Les sélecteurs que vous appliquez à un magasin de données d'événement contrôlent les événements qui persistent et

que vous pouvez interroger. Pour plus d'informations sur CloudTrail Lake, consultez la section [Travailler avec AWS CloudTrail Lake](#) dans le guide de AWS CloudTrail l'utilisateur.

CloudTrail Les stockages et requêtes de données sur les événements de Lake entraînent des coûts. Lorsque vous créez un magasin de données d'événement, vous choisissez l'[option de tarification](#) que vous voulez utiliser pour le magasin de données d'événement. L'option de tarification détermine le coût d'ingestion et de stockage des événements, ainsi que les périodes de conservation par défaut et maximale pour le magasin de données d'événement. Pour plus d'informations sur la CloudTrail tarification, consultez la section [AWS CloudTrail Tarification](#).

Événements de gestion dans CloudTrail

[Les événements de gestion](#) fournissent des informations sur les opérations de gestion effectuées sur les ressources de votre Compte AWS. Ils sont également connus sous le nom opérations de plan de contrôle. Par défaut, CloudTrail enregistre les événements de gestion.

Managed Integrations enregistre les opérations du plan de contrôle des intégrations gérées suivantes CloudTrail sous forme d'événements de gestion.

- `CreateCloudConnector`
- `UpdateCloudConnector`
- `GetCloudConnector`
- `DeleteCloudConnector`
- `ListCloudConnectors`
- `CreateConnectorDestination`
- `UpdateConnectorDestination`
- `GetConnectorDestination`
- `DeleteConnectorDestination`
- `ListConnectorDestinations`
- `CreateAccountAssociation`
- `UpdateAccountAssociation`
- `GetAccountAssociation`
- `DeleteAccountAssociation`
- `ListAccountAssociations`

- StartAccountAssociationRefresh
- ListManagedThingAccountAssociations
- RegisterAccountAssociation
- DeregisterAccountAssociation
- SendConnectorEvent
- ListDeviceDiscoveries
- ListDiscoveredDevices

Exemples d'événements

Un événement représente une demande unique provenant de n'importe quelle source et inclut des informations sur l'opération d'API demandée, la date et l'heure de l'opération, les paramètres de la demande, etc. CloudTrail les fichiers journaux ne constituent pas une trace ordonnée des appels d'API publics. Les événements n'apparaissent donc pas dans un ordre spécifique.

L'exemple suivant montre un CloudTrail événement qui illustre le succès d'une opération `CreateCloudConnector` d'API.

CloudTrail Événement réussi avec l'opération de l'**CreateCloudConnector**API.

```
{
  "eventVersion": "1.09",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "EXAMPLE",
    "arn": "arn:aws:sts::111122223333:assumed-role/Admin/EXAMPLE",
    "accountId": "111122223333",
    "accessKeyId": "EXAMPLEKYSBQSCGRIC",
    "sessionContext": {
      "sessionIssuer": {
        "type": "Role",
        "principalId": "AR0AZ0ZQFKYSFZVB2J2GN",
        "arn": "arn:aws:iam::111122223333:role/Admin",
        "accountId": "111122223333",
        "userName": "Admin"
      },
      "attributes": {
        "creationDate": "2025-06-05T18:26:16Z",
        "mfaAuthenticated": "false"
      }
    }
  }
}
```

```

    }
  },
  "eventTime": "2025-06-05T18:30:40Z",
  "eventSource": "iotmanagedintegrations.amazonaws.com",
  "eventName": "CreateCloudConnector",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "192.0.2.0",
  "userAgent": "PostmanRuntime/7.44.0",
  "requestParameters": {
    "EndpointType": "LAMBDA",
    "Description": "Manual testing for C2C CT Validation",
    "ClientToken": "abc7460",
    "EndpointConfig": {
      "lambda": {
        "arn": "arn:aws:lambda:us-
east-1:111122223333:function:LightweightMockConnector7460"
      }
    },
    "Name": "EdenManualTestCloudConnector"
  },
  "responseElements": {
    "X-Frame-Options": "DENY",
    "Access-Control-Expose-Headers": "Content-Length,Content-Type,X-Amzn-
Errortype,X-Amzn-Requestid",
    "Strict-Transport-Security": "max-age:47304000; includeSubDomains",
    "Cache-Control": "no-store, no-cache",
    "X-Content-Type-Options": "nosniff",
    "Content-Security-Policy": "upgrade-insecure-requests; default-src 'none';
object-src 'none'; frame-ancestors 'none'; base-uri 'none'",
    "Pragma": "no-cache",
    "Id": "f7e633e719404c4a933596b4d0cc276e",
    "Arn": "arn:aws:iotmanagedintegrations:us-east-1:111122223333:cloud-connector/
EXAMPLE404c4a933596b4d0cc276e"
  },
  "requestID": "c0071fd1-b8e0-400a-bcc0-EXAMPLE9e4",
  "eventID": "95b318ea-2f63-4183-9c22-EXAMPLE3e",
  "readOnly": false,
  "eventType": "AwsApiCall",
  "managementEvent": true,
  "recipientAccountId": "111122223333",
  "eventCategory": "Management"
}

```

L'exemple suivant montre un CloudTrail événement qui illustre le succès d'une opération `ListDiscoveredDevices` d'API.

CloudTrail Événement réussi avec l'opération de l'**ListDiscoveredDevices**API.

```
{
  "eventVersion": "1.09",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "EZAMPLE",
    "arn": "arn:aws:sts::444455556666:assumed-role/Admin/EXAMPLE",
    "accountId": "444455556666",
    "accessKeyId": "EXAMPLERJ26PYMH",
    "sessionContext": {
      "sessionIssuer": {
        "type": "Role",
        "principalId": "EXAMPLE",
        "arn": "arn:aws:iam::444455556666:role/Admin",
        "accountId": "444455556666",
        "userName": "Admin"
      },
      "attributes": {
        "creationDate": "2025-06-10T23:37:31Z",
        "mfaAuthenticated": "false"
      }
    }
  },
  "eventTime": "2025-06-10T23:38:07Z",
  "eventSource": "iotmanagedintegrations.amazonaws.com",
  "eventName": "ListDiscoveredDevices",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "192.0.2.0",
  "userAgent": "EXAMPLE-runtime/2.4.0",
  "requestParameters": {
    "Identifier": "EXAMPLE4f268483a17d8060f014"
  },
  "responseElements": null,
  "requestID": "27ae1f61-e2e6-43e4-bf17-EXAMPLEa568",
  "eventID": "34734e81-76a8-49a4-9641-EXAMPLE28ed",
  "readOnly": true,
  "eventType": "AwsApiCall",
  "managementEvent": true,
  "recipientAccountId": "444455556666",
  "eventCategory": "Management"
}
```

```
}
```

Pour plus d'informations sur le contenu des CloudTrail enregistrements, voir [le contenu des CloudTrail enregistrements](#) dans le Guide de AWS CloudTrail l'utilisateur.

Historique des documents relatifs aux intégrations gérées — Guide du développeur

Le tableau suivant décrit les versions de documentation relatives aux intégrations gérées.

Modification	Description	Date
Version de disponibilité générale	Version de disponibilité générale du guide du développeur sur les intégrations gérées	25 juin 2025
Version préliminaire initiale	Version préliminaire initiale du guide du développeur sur les intégrations gérées	3 mars 2025

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.