デベロッパーガイド

のマネージド統合 AWS IoT Device Management

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

のマネージド統合 AWS IoT Device Management: デベロッパーガイド

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

Amazon の商標およびトレードドレスはAmazon 以外の製品およびサービスに使用することはできません。また、お客様に誤解を与える可能性がある形式で、または Amazon の信用を損なう形式で使用することもできません。Amazon が所有していないその他のすべての商標は Amazon との提携、関連、支援関係の有無にかかわらず、それら該当する所有者の資産です。

Table of Contents

... viii
マネージド統合とは ... 1
初めてのマネージド統合ユーザーですか？ ... 1
マネージド統合の概要 ... 1
マネージド型統合のお客様は誰ですか？ ... 2
マネージド統合の用語 ... 2
一般的なマネージド統合の用語 ... 2
デバイスタイプ ... 3
Cloud-to-cloud の用語 ... 4
データモデル用語 ... 4
設定 ... 6
にサインアップする AWS アカウント ... 6
管理アクセスを持つユーザーを作成する ... 6
入門 ... 9
直接接続されたデバイスのオンボーディング ... 9
暗号化キーの設定 (オプション) ... 9
カスタムエンドポイントを登録する (必須) ... 10
デバイスプロビジョニング (必須) ... 11
マネージド統合 エンドデバイス SDK (必須) ... 13
Device Pre-Association with Credential Locker (オプション) ... 14
デバイスの検出とオンボーディング (オプション) ... 14
Device Command and Control ... 15
API インデックス ... 16
ハブ接続デバイスのオンボーディング ... 16
モバイルアプリケーションの調整 (オプション) ... 16
暗号化キーの設定 (オプション) ... 17
カスタムエンドポイントを登録する (必須) ... 18
デバイスプロビジョニング (必須) ... 18
マネージドインテグレーション Hub SDK (必須) ... 20
認証情報ロッカーとのデバイスの事前関連付け (オプション) ... 21
デバイスの検出とオンボーディング (必須) ... 21
Device Command and Control ... 22
API インデックス ... 22
Cloud-to-cloudデバイスのオンボーディング ... 23

モバイルアプリケーションの調整 (必須) 23
暗号化キーの設定 (オプション) 24
アカウントのリンク (必須) 24
デバイス検出 (必須) 25
Device Command and Control 25
API インデックス 26
デバイスプロビジョニング 27
デバイスプロビジョニングとは 27
デバイスのライフサイクルとプロファイルを管理する 29
デバイス 29
デバイスプロファイル 30
データモデル 31
マネージド統合データモデル 31
AWS Matter データモデルの実装 33
デバイスコマンドとイベント 35
Device コマンド 35
デバイスイベント 37
通知の設定 39
マネージド統合通知の設定 39
ハブ SDK 45
Hub SDK アーキテクチャ 45
デバイスオンボーディング 45
デバイスオンボーディングコンポーネント 45
デバイスオンボーディングフロー 46
デバイスコントロール 47
SDK コンポーネント 48
デバイスコントロールフロー 49
ハブのオンボーディング 49
Hub オンボーディングサブシステム 50
オンボーディングのセットアップ 51
マネージド統合 Hub SDK をインストールして検証する 59
を使用して SDK をインストールする AWS IoT Greengrass 59
スクリプトを使用して Hub SDK をデプロイする 62
systemd を使用して Hub SDK をデプロイする 65
カスタム証明書ハンドラー 68
API 定義とコンポーネント 69

ビルド例 71
使用方法 75
プロセス間通信 (IPC) APIs 76
IPC クライアントのセットアップ 76
IPC インターフェイスの定義とペイロード 80
ハブコントロール 84
前提条件 85
エンドデバイス SDK コンポーネント 85
エンドデバイス SDK との統合 85
例: ハブコントロールを構築する 88
サポートされている例 89
サポートされているプラットフォーム 89
サポートされている Zigbee および Z-Wave デバイスタイプ 89
CloudWatch Logs を有効にする 91
前提条件 92
Setup Hub SDK ログ設定 92
エンドデバイス SDK 95
エンドデバイス SDK について 95
アーキテクチャとコンポーネント 96
プロビジョニング先 97
プロビジョンドワークフロー 97
環境変数を設定する 98
カスタムエンドポイントを作成する 98
プロビジョニングプロファイルを作成する 98
マネージド型モノを作成する 99
SDK ユーザーの Wi-Fi プロビジョニング 100
クレームによるフリートのプロビジョニング 100
モノのマネージド機能 100
ジョブハンドラー 100
ジョブハンドラーの仕組み 101
ジョブハンドラーの実装 101
データモデルコードジェネレーター 104
コード生成プロセス 105
環境設定 106
コードの生成 108
低レベルの C 機能 APIs 109

OnOff クラスター API ... 110
サービスとデバイスのやり取り ... 112
リモートコマンドの処理 ... 113
未承諾イベントの処理 ... 113
エンドデバイス SDK の統合 ... 114
エンドデバイス SDK を移植する ... 125
End Device SDK をダウンロードして検証する ... 125
PAL をデバイスに移植する ... 126
ポートをテストする ... 128
付録 ... 129
付録 A: サポートされているプラットフォーム ... 129
付録 B: 技術要件 ... 129
付録 C: 共通 API ... 130
プロトコルミドルウェアとは ... 132
ミドルウェアアーキテクチャ ... 132
End-to-endミドルウェアコマンドフローの例 ... 133
ミドルウェアコードの整理 ... 133
Zigbee ミドルウェアコードの整理 ... 134
Z-Wave ミドルウェアコードの整理 ... 137
SDK との統合によるミドルウェア ... 139
デバイス移植キット (DPK) API 統合 ... 140
リファレンス実装とコード編成 ... 140
セキュリティ ... 142
データ保護 ... 142
マネージド統合の保管時のデータ暗号化 ... 143
Identity and Access Management ... 149
対象者 ... 150
アイデンティティを使用した認証 ... 151
ポリシーを使用したアクセスの管理 ... 154
AWS マネージドポリシー ... 157
マネージド統合と IAM の連携方法 ... 161
アイデンティティベースのポリシーの例 ... 167
トラブルシューティング ... 170
サービスにリンクされたロールの使用 ... 172
コンプライアンス検証 ... 176
耐障害性 ... 177

モニタリング 179
CloudWatch によるモニタリング 179
イベントのモニタリング 180
eventName イベント 180
CloudTrail ログ 181
CloudTrail でイベントを管理する 182
イベント例 183
ドキュメント履歴 187

のマネージド統合 AWS IoT Device Management はプレビューリリースであり、変更される可能性があります。アクセスについては、 マネージド統合コンソールからお問い合わせください。

翻訳は機械翻訳により提供されています。提供された翻訳内容と英語版の間で齟齬、不一致または矛盾がある場合、英語版が優先します。

マネージド型統合とは AWS IoT Device Management

マネージド統合は の機能であるため AWS IoT Device Management、デベロッパーはデバイスベンダーや接続プロトコルに関係なく、デバイスセットアップワークフローを自動化し、多くのデバイス間の相互運用性をサポートできます。1 つのユーザーインターフェイスを使用して、さまざまなデバイスを制御、管理、運用できます。

トピック

- 初めてのマネージド統合ユーザーですか？
- マネージド統合の概要
- マネージド型統合のお客様は誰ですか？
- マネージド統合の用語

初めてのマネージド統合ユーザーですか？

マネージド統合を初めて使用する場合は、まず以下のセクションを読むことをお勧めします。

- マネージド統合のセットアップ
- の マネージド統合の開始方法 AWS IoT Device Management

マネージド統合の概要

次の図は、 マネージド統合機能の概要を示しています。

> **Note**
>
> の マネージド統合 AWS IoT Device Management は、現時点ではタグ付けをサポートしていません。つまり、この機能のリソースを組織のタグ付けポリシーに含めることはできません。詳細については、ホワイトペーパーの「ユースケースのタグ付け」を参照してください。 AWS

マネージド型統合のお客様は誰ですか？

マネージド統合のお客様は、 機能を使用してデバイスセットアッププロセスを自動化し、デバイスベンダーや接続プロトコルに関係なく、多くのデバイス間で相互運用性サポートを提供します。これらのソリューションプロバイダーは、デバイス向けの統合機能を提供し、ハードウェアメーカーと提携してサービスの範囲を拡張します。お客様は、 で定義されているデータモデルを使用してデバイスとやり取りできます AWS。

マネージド統合内のさまざまなロールについては、次の表を参照してください。

ロール	責任
Manufacturer	• 製造デバイス。 • マネージド統合へのデバイスプロファイルの登録。
エンドユーザー	• マネージド統合に接続するデバイスを自宅に管理します。
お客様	• マネージド統合と通信する特定のデバイスをセットアップして制御するための別のソリューションを構築します。 • 独自の顧客およびエンドユーザーにサービスを提供する。

マネージド統合の用語

マネージド統合には、独自のデバイス実装を管理するために理解しておくべき多くの概念と用語があります。以下のセクションでは、マネージド統合をよりよく理解するための主要な概念と用語について説明します。

一般的なマネージド統合の用語

マネージド型統合について理解すべき重要な概念は、 AWS IoT Core モノ`managedThing`との比較です。

- AWS IoT Core モノ: AWS IoT Core モノは、デジタル表現を提供する AWS IoT Core コンストラクトです。開発者は、ポリシー、データストレージ、ルール、アクション、MQTT トピック、およびデータストレージへのデバイス状態の配信を管理することが期待されます。 AWS IoT Core モノの詳細については、「 を使用したデバイスの管理 AWS IoT」を参照してください。
- マネージド統合 *managedThing*: では`managedThing`、デバイスの操作を簡素化するための抽象化を提供し、開発者がルール、アクション、MQTT トピック、ポリシーなどの項目を作成する必要はありません。

デバイスタイプ

マネージド統合は、さまざまなタイプのデバイスを管理します。これらのタイプのデバイスは、次の 3 つのカテゴリのいずれかに分類されます。

- 直接接続デバイス: このタイプのデバイスは、マネージド統合エンドポイントに直接接続します。通常、これらのデバイスは、直接接続用のマネージド統合デバイス SDK を含むデバイスメーカーによって構築および管理されます。
- ハブ接続デバイス: これらのデバイスは、デバイス検出、オンボーディング、制御機能を管理するマネージド統合 Hub SDK を実行するハブを介してマネージド統合に接続します。エンドユーザーは、ボタンの押下開始またはバーコードスキャンを使用して、これらのデバイスをオンボードできます。

  ハブ接続デバイスのオンボーディングでは、次の 2 つのワークフローがサポートされています。

  - エンドユーザーが開始したボタンを押してデバイス検出を開始する
  - デバイスの関連付けを実行するためのバーコードベースのスキャン
- Cloud-to-cloudデバイス: エンドユーザーがクラウドデバイスに初めて電源を入れるときは、デバイスの機能とメタデータを取得するために、マネージド統合のために、それぞれのサードパーティーのクラウドプロバイダーでプロビジョニングする必要があります。そのプロビジョニングワークフローを完了すると、マネージド統合はエンドユーザーに代わってクラウドデバイスおよびサードパーティーのクラウドプロバイダーと通信できます。

> **Note**
>
> ハブは、上記の特定のデバイスタイプではありません。その目的は、スマートホームデバイスのコントローラーとして機能し、マネージド統合とサードパーティーのクラウドプロバイ

ダー間の接続を容易にすることです。ロールは、上記のデバイスタイプとハブの両方として機能します。

Cloud-to-cloud の用語

マネージド統合と統合する物理デバイスは、サードパーティーのクラウドプロバイダーから発信される場合があります。これらのデバイスをマネージド統合にオンボードし、サードパーティーのクラウドプロバイダーと通信するために、以下の用語では、これらのワークフローをサポートする主要な概念の一部について説明します。

- Cloud-to-cloud (C2C) コネクタ: C2C コネクタは、マネージド統合とサードパーティーのクラウドプロバイダー間の接続を確立します。
- サードパーティーのクラウドプロバイダー: マネージド統合の外部で製造および管理されるデバイスの場合、サードパーティーのクラウドプロバイダーは、エンドユーザーとマネージド統合のためにこれらのデバイスの制御を可能にし、デバイスコマンドなどのさまざまなワークフローのためにサードパーティーのクラウドプロバイダーと通信します。

データモデル用語

マネージド統合では、2 つのデータモデルを使用してデータを整理し、デバイス間のend-to-end通信を行います。次の用語では、これら 2 つのデータモデルを理解するための主要な概念の一部について説明します。

- デバイス： 物理デバイス (ビデオドアベルなど) を表すエンティティ。複数のノードが連携して完全な機能セットを提供します。
- ノード： デバイスは複数のノードで構成されます (Matter Data Model の AWS実装から採用)。各ノードは他のノードとの通信を処理します。ノードは、通信を容易にするために一意にアドレス指定できます。
- エンドポイント： エンドポイントはスタンドアロン機能 (リンガー、モーション検出、ビデオドアベルの照明) をカプセル化します。
- 機能： エンドポイントで機能を使用できるようにするために必要なコンポーネントを表すエンティティ (ビデオドアベルのベル機能のボタンまたはライトとチャイム)。
- アクション： デバイスの機能とのやり取りを表すエンティティ (鐘を鳴らすか、ドアにいる人を表示する)。

- イベント： デバイスの機能からのイベントを表すエンティティ。デバイスはイベントを送信して、インシデント/アラーム、センサーからのアクティビティなどを報告することができます (ドアにノック/リングがあるなど) 。
- プロパティ： デバイス状態の特定の属性を表すエンティティ (ベルが鳴り、ポークライトがオン、カメラが録画中) 。
- データモデル： データレイヤーは、アプリケーションの機能をサポートするのに役立つデータと動詞要素に対応します。アプリケーションは、デバイスを操作するインテントがある場合、これらのデータ構造で動作します。詳細については、GitHub ウェブサイトの connectedhomeip を参照してください。
- スキーマ： スキーマは、JSON 形式のデータモデルを表します。

マネージド統合のセットアップ

以下のセクションでは、 のマネージド統合を使用するための初期設定について説明します AWS IoT Device Management。

トピック

- にサインアップする AWS アカウント
- 管理アクセスを持つユーザーを作成する

にサインアップする AWS アカウント

がない場合は AWS アカウント、次の手順を実行して作成します。

にサインアップするには AWS アカウント

1. https://portal.aws.amazon.com/billing/signup を開きます。
2. オンラインの手順に従います。

   サインアップ手順の一環として、通話呼び出しを受け取り、電話キーパッドで検証コードを入力するように求められます。

   にサインアップすると AWS アカウント、 AWS アカウントのルートユーザー が作成されます。ルートユーザーには、アカウントのすべての AWS のサービス とリソースへのアクセス権があります。セキュリティベストプラクティスとして、ユーザーに管理アクセス権を割り当て、ルートユーザーアクセスが必要なタスクの実行にはルートユーザーのみを使用するようにしてください。

AWS サインアッププロセスが完了すると、 から確認メールが送信されます。https://aws.amazon.com/ の [マイアカウント] をクリックして、いつでもアカウントの現在のアクティビティを表示し、アカウントを管理することができます。

管理アクセスを持つユーザーを作成する

にサインアップしたら AWS アカウント、日常的なタスクにルートユーザーを使用しないように AWS アカウントのルートユーザー、 のセキュリティを確保し AWS IAM Identity Center、 を有効にして管理ユーザーを作成します。

を保護する AWS アカウントのルートユーザー

1. ルートユーザーを選択し、 AWS アカウント E メールアドレスを入力して、アカウント所有者AWS Management Consoleとして にサインインします。次のページでパスワードを入力します。

   ルートユーザーを使用してサインインする方法については、AWS サインイン ユーザーガイドのルートユーザーとしてサインインするを参照してください。

2. ルートユーザーの多要素認証 (MFA) を有効にします。

   手順については、IAM ユーザーガイドの AWS アカウント 「ルートユーザー (コンソール) の仮想 MFA デバイスを有効にする」を参照してください。

管理アクセスを持つユーザーを作成する

1. IAM アイデンティティセンターを有効にします。

   手順については、「AWS IAM Identity Center ユーザーガイド」の「AWS IAM Identity Centerの有効化」を参照してください。

2. IAM アイデンティティセンターで、ユーザーに管理アクセスを付与します。

   を ID ソース IAM アイデンティティセンターディレクトリ として使用する方法のチュートリアルについては、AWS IAM Identity Center 「 ユーザーガイド」の「デフォルトを使用してユーザーアクセスを設定する IAM アイデンティティセンターディレクトリ」を参照してください。

管理アクセス権を持つユーザーとしてサインインする

- IAM アイデンティティセンターのユーザーとしてサインインするには、IAM アイデンティティセンターのユーザーの作成時に E メールアドレスに送信されたサインイン URL を使用します。

  IAM Identity Center ユーザーを使用してサインインする方法については、AWS サインイン 「 ユーザーガイド」の AWS 「 アクセスポータルにサインインする」を参照してください。

追加のユーザーにアクセス権を割り当てる

1. IAM アイデンティティセンターで、最小特権のアクセス許可を適用するというベストプラクティスに従ったアクセス許可セットを作成します。

手順については、「AWS IAM Identity Center ユーザーガイド」の「権限設定を作成する」を参照してください。

2. グループにユーザーを割り当て、そのグループにシングルサインオンアクセス権を割り当てます。

   手順については、「AWS IAM Identity Center ユーザーガイド」の「グループの結合」を参照してください。

の マネージド統合の開始方法 AWS IoT Device Management

以下のセクションでは、 マネージド統合の使用を開始するために必要なステップの概要を説明します。直接接続デバイス、ハブ接続デバイス、クラウド間デバイスの詳細については、「」を参照してください[デバイスタイプ](#)。 cloud-to-cloud

トピック

- 直接接続されたデバイスのオンボーディング
- ハブ接続デバイスのオンボーディング
- Cloud-to-cloudデバイスのオンボーディング

直接接続されたデバイスのオンボーディング

次の手順では、直接接続されたデバイスを マネージド統合にオンボーディングするワークフローの概要を説明します。

トピック

- 暗号化キーの設定 (オプション)
- カスタムエンドポイントを登録する (必須)
- デバイスプロビジョニング (必須)
- マネージド統合 エンドデバイス SDK (必須)
- Device Pre-Association with Credential Locker (オプション)
- デバイスの検出とオンボーディング (オプション)
- Device Command and Control
- API インデックス

暗号化キーの設定 (オプション)

セキュリティは、エンドユーザー、マネージド型統合、サードパーティークラウド間でルーティングされるデータにとって最も重要です。デバイスデータを保護するためにサポートされる方法の 1 つは、データをルーティングするための安全な暗号化キーを使用したend-to-endの暗号化です。

マネージド統合のお客様には、暗号化キーを使用するための次の 2 つのオプションがあります。

- デフォルトのマネージド統合マネージド暗号化キーを使用します。
- AWS KMS key 作成した を指定します。

`PutDefaultEncryptionConfiguration` API を呼び出すと、使用する暗号化キーオプションを更新するためのアクセス権が付与されます。デフォルトでは、マネージド統合はデフォルトのマネージド統合マネージド暗号化キーを使用します。`PutDefaultEncryptionConfiguration` API を使用して、暗号化キー設定をいつでも更新できます。

さらに、 `DescribeDefaultEncryptionConfiguration` API コマンドを呼び出すと、デフォルトまたは指定されたリージョンの AWS アカウントの暗号化設定に関する情報が返されます。

マネージド統合によるend-to-endの暗号化の詳細については、「」を参照してください[マネージド統合の保管時のデータ暗号化](#)。

AWS KMS サービスの詳細については、「」を参照してください。 AWS Key Management Service

このステップで使用される APIs:

- `PutDefaultEncryptionConfiguration`
- `DescribeDefaultEncryptionConfiguration`

カスタムエンドポイントを登録する (必須)

デバイスと マネージド統合間の双方向通信は、次の項目を容易にします。

- デバイスコマンドのプロンプトルーティング。
- 物理デバイスとマネージド統合マネージドモノのデジタル表現の状態が調整されます。
- デバイスデータの安全な送信。

マネージド統合に接続するには、デバイスにはトラフィックをルーティングするための専用エンドポイントが必要です。`RegisterCustomEndpoint` API を呼び出して、サーバーの信頼の管理方法を設定するだけでなく、このエンドポイントを作成します。カスタムエンドポイントは、 マネージド統合に接続するローカルハブまたは Wi-Fi デバイスのデバイス SDK に保存されます。

> **Important**
>
> RegisterCustomEndpoint の失敗エラーが発生した場合は、Service Quotas コンソールでクォータの引き上げを 0 から 1 にリクエストします。https://console.aws.amazon.com/servicequotas/

> **Note**
>
> クラウドに接続されたデバイスでは、このステップをスキップできます。

このステップで使用される APIs:

- `RegisterCustomEndpoint`

デバイスプロビジョニング (必須)

デバイスプロビジョニングは、将来の双方向通信のために、デバイスまたはデバイスのフリートとマネージド統合間のリンクを確立します。`CreateProvisioningProfile` API を呼び出してプロビジョニングテンプレートを作成し、証明書を要求します。

- プロビジョニングテンプレートは、プロビジョニングプロセス中にデバイスに適用される一連のリソースとポリシーを定義するドキュメントです。マネージド統合に初めて接続するときに、デバイスを登録して設定する方法を指定します。このプロセスにより、デバイスのセットアッププロセスが自動化され、各デバイスが安全かつ一貫して適切なアクセス許可、ポリシー、設定 AWS IoT でに統合されます。
- クレーム証明書は、フリートのプロビジョニング中に使用される一時的な証明書であり、エンドユーザーに配信される前に、製造中に一意のデバイス証明書がデバイスにプリインストールされていない場合にのみ使用されます。

次のリストは、デバイスのプロビジョニングワークフローと、それぞれの違いの概要を示しています。

- 単一デバイスのプロビジョニング
  - マネージド統合を使用して 1 つのデバイスをプロビジョニングする。
  - ワークフロー

    - `CreateManagedThing`: プロビジョニングテンプレートに基づいて、マネージド統合を使用して新しいマネージド型モノ (デバイス) を作成します。
  - エンドデバイスソフトウェア開発キット (SDK) の詳細については、「」を参照してください[End Device SDK とは](#)。
  - 単一デバイスのプロビジョニングの詳細については、「単一モノのプロビジョニング」を参照してください。
- クレームによるフリートのプロビジョニング
  - 承認されたユーザーによるプロビジョニング
    - エンドユーザーがマネージド統合にデバイスをプロビジョニングできるように、組織のデバイスプロビジョニングワークフロー (複数可) に固有の IAM ロールとポリシーを作成する必要があります。このワークフローの IAM ロールとポリシーの作成の詳細については、「デバイスをインストールするユーザーの IAM ポリシーとロールの作成」を参照してください。
    - ワークフロー
      - `CreateKeysAndCertificate`: デバイスの暫定クレーム証明書とキーを作成します。
      - `CreatePolicy`: デバイスのアクセス許可を定義するポリシーを作成します。
      - `AttachPolicy`: ポリシーを暫定クレーム証明書にアタッチします。
      - `CreateProvisioningTemplate`: デバイスのプロビジョニング方法を定義するプロビジョニングテンプレートを作成します。
      - `RegisterThing`: プロビジョニングテンプレートに基づいて IoT レジストリに新しいモノ (デバイス) を登録するデバイスプロビジョニングプロセスの一部。
      - さらに、プロビジョニングクレームを使用して初めて AWS IoT Core に接続すると、デバイスは MQTT または HTTPS プロトコルを使用して安全な通信を行います。このプロセス中、AWS IoT Core の内部メカニズムはクレームを検証し、プロビジョニングテンプレートを適用して、プロビジョニングプロセスを完了します。
  - クレーム証明書を使用したプロビジョニング
    - マネージド統合との最初の問い合わせのために、各デバイスクレーム証明書にアタッチされたクレーム証明書プロビジョニングポリシーを作成し、デバイス固有の証明書に置き換える必要があります。クレーム証明書によるプロビジョニングのワークフローを完了するには、ハードウェアシリアル番号を MQTT 予約済みトピックに送信する必要があります。
    - ワークフロー
      - `CreateKeysAndCertificate`: デバイスの暫定クレーム証明書とキーを作成します。
      - `CreatePolicy`: デバイスのアクセス許可を定義するポリシーを作成します。
      - `AttachPolicy`: ポリシーを暫定クレーム証明書にアタッチします。

- `CreateProvisioningTemplate`: デバイスのプロビジョニング方法を定義するプロビジョニングテンプレートを作成します。
- `RegisterThing`: プロビジョニングテンプレートに基づいて IoT レジストリに新しいモノ (デバイス) を登録するデバイスプロビジョニングプロセスの一部。
- さらに、プロビジョニングクレームを使用して初めて AWS IoT Core に接続すると、デバイスは MQTT または HTTPS プロトコルを使用して安全な通信を行います。このプロセス中、AWS IoT Core の内部メカニズムはクレームを検証し、プロビジョニングテンプレートを適用して、プロビジョニングプロセスを完了します。

- クレーム証明書によるプロビジョニングの詳細については、「クレームによるプロビジョニング」を参照してください。

プロビジョニングテンプレートの詳細については、「プロビジョニングテンプレート」を参照してください。

このステップで使用される APIs:

- `CreateManagedThing`
- `CreateProvisioningProfile`
- `RegisterCACertificate`
- `CreatePolicy`
- `CreateThing`
- `AttachPolicy`
- `AttachThingPrincipal`
- `CreateKeysAndCertificate`
- `CreateProvisioningTemplate`

マネージド統合 エンドデバイス SDK (必須)

初回製造時に、デバイスのファームウェアに End Device SDK を追加します。エンドユーザーのデバイスのプロビジョニングをサポートするために、マネージド統合用に、先ほど作成した暗号化キー、カスタムエンドポイントアドレス、セットアップ認証情報、クレーム証明書、プロビジョニングテンプレートを End Device SDK に追加します。

エンドデバイス SDK の詳細については、「」を参照してください。 End Device SDK とは

Device Pre-Association with Credential Locker (オプション)

フルフィルメントプロセス中、デバイスのバーコードがスキャンされ、デバイスの情報がマネージド統合にアップロードされます。これにより、 `CreateManagedThing` API が自動的に呼び出され、マネージド型統合に保存されている物理デバイスのデジタル表現である Managed Thing が作成されます。さらに、 `CreateManagedThing` API はデバイスのプロビジョニング中に`deviceID`使用するために を自動的に返します。

利用可能な場合は、所有者の情報を`CreateManagedThing`リクエストメッセージに含めることができます。この所有者情報を含めると、セットアップ認証情報と事前定義されたデバイス機能を取得して、 マネージド統合に保存`managedThing`されている に含めることができます。これにより、マネージド統合を使用してデバイスまたはデバイスのフリートをプロビジョニングする時間を短縮できます。

所有者の情報が利用できない場合、`CreateManagedThing`API コールの `owner`パラメータは空白のままになり、デバイスがオンになっているときにデバイスのオンボーディング中に更新されます。

このステップで使用される APIs:

- `CreateManagedThing`

デバイスの検出とオンボーディング (オプション)

エンドユーザーがデバイスをオンにするか、必要に応じてペアリングモードに設定すると、次の検出およびオンボーディングワークフローを使用できます。

簡易セットアップ (SS)

エンドユーザーは IoT デバイスの電源を入れ、マネージド統合アプリを使用して QR コードをスキャンします。アプリはマネージド統合クラウドにデバイスを登録し、IoT Hub に接続します。

ユーザーガイドによるセットアップ (UGS)

エンドユーザーはデバイスの電源を入れ、インタラクティブな手順に従ってマネージド統合にオンボードします。これには、IoT Hub のボタンを押す、マネージド統合アプリを使用する、ハブとデバイスの両方のボタンを押すことが含まれます。Simple Setup が失敗する場合は、この方法を使用します。

- スマートデバイス： Hub デバイスがローカルネットワーク認証情報と SSID を共有し、Wi-Fi デバイスをローカル Hub デバイスに関連付けるローカル Hub デバイスへの接続が自動的に開始されま

す。次に、スマートデバイスは、サーバー名表示 (SNI) 拡張機能を使用して、前に作成したカスタムエンドポイントへの接続を試みます。

- スマート機能のない Wi-Fi デバイス： Wi-Fi デバイスは `StartDeviceDiscovery` API を自動的に呼び出し、Wi-Fi デバイスを関連付けるローカル Hub デバイスに加えて、Wi-Fi デバイスとローカル Hub デバイス間のペアリングプロセスを開始します。次に、Wi-Fi デバイスは、サーバー名表示 (SNI) 拡張機能を使用して以前に作成したカスタムエンドポイントへの接続を試みます。
- モバイルアプリケーションが設定されていない Wi-Fi デバイス： ローカル Hub デバイスで、Wi-Fi などのすべての無線プロトコルの受信を開始できるようにします。Wi-Fi デバイスはローカル Hub デバイスに自動的に接続され、ローカル Hub デバイスは Wi-Fi デバイスを関連付けます。次に、Wi-Fi デバイスは、サーバー名表示 (SNI) 拡張機能を使用して以前に作成したカスタムエンドポイントへの接続を試みます。

このステップで使用される API:

- `StartDeviceDiscovery`

Device Command and Control

デバイスオンボーディングが完了したら、デバイスを管理するためのデバイスコマンドの送受信を開始できます。次のリストは、デバイスを管理するためのシナリオの一部を示しています。

- デバイスコマンドの送信： デバイスのライフサイクルを管理するためのコマンドをデバイスから送受信します。
  - 使用される APIsのサンプリング: `SendManagedThingCommand`。
- デバイスの状態の更新： 送信されたデバイス機能とデバイスコマンドに基づいて、デバイスの状態を更新します。
  - 使用される APIs のサンプリング: `GetManagedThingState`、`ListManagedThingState`、`UpdateManagedThing`、および `DeleteManagedThing`。
- デバイスイベントの受信： マネージド統合に送信されるサードパーティーのクラウドプロバイダーから C2C デバイスに関するイベントを受信します。
  - 使用される APIs のサンプリング: `SendDeviceEvent`、`CreateLogLevel`、`CreateNotificationConfiguration`。

このステップで使用される APIs:

- `SendManagedThingCommand`
- `GetManagedThingState`
- `ListManagedThingState`
- `UpdateManagedThing`
- `DeleteManagedThing`
- `SendDeviceEvent`
- `CreateLogLevel`
- `CreateNotificationConfiguration`

API インデックス

マネージド統合 APIs「 マネージド統合 API リファレンスガイド」を参照してください。

AWS IoT Core APIs の詳細については、 AWS IoT Core API リファレンスガイドを参照してください。

ハブ接続デバイスのオンボーディング

トピック

- モバイルアプリケーションの調整 (オプション)
- 暗号化キーの設定 (オプション)
- カスタムエンドポイントを登録する (必須)
- デバイスプロビジョニング (必須)
- マネージドインテグレーション Hub SDK (必須)
- 認証情報ロッカーとのデバイスの事前関連付け (オプション)
- デバイスの検出とオンボーディング (必須)
- Device Command and Control
- API インデックス

モバイルアプリケーションの調整 (オプション)

エンドユーザーにモバイルアプリケーションを提供すると、モバイルデバイスから直接デバイスを管理するための一貫したユーザーエクスペリエンスが容易になります。エンドユーザーは、モバイルア

プリケーションで直感的なユーザーインターフェイスを活用して、さまざまなマネージド統合 APIs を呼び出して、デバイスを制御、管理、運用できます。モバイルアプリケーションは、所有者 ID、サポートされているデバイスプロトコル、デバイス機能などのデバイスメタデータをルーティングすることで、デバイス検出を支援できます。

さらに、モバイルアプリケーションは、サードパーティーのクラウドデバイスのエンドユーザーのアカウントとデバイスデータを含むサードパーティーのクラウドとマネージド統合 AWS アカウントの をリンクするのに役立ちます。アカウントリンクにより、エンドユーザーのモバイルアプリケーション、 マネージド統合 AWS アカウント の 、サードパーティークラウド間でデバイスデータをシームレスにルーティングできます。

暗号化キーの設定 (オプション)

セキュリティは、エンドユーザー、マネージド型統合、サードパーティークラウド間でルーティングされるデータにとって最も重要です。デバイスデータを保護するためにサポートされる方法の 1 つは、データをルーティングするための安全な暗号化キーを使用したend-to-endの暗号化です。

マネージド統合のお客様には、暗号化キーを使用するための次の 2 つのオプションがあります。

- デフォルトのマネージド統合マネージド暗号化キーを使用します。
- AWS KMS key 作成した を指定します。

`PutDefaultEncryptionConfiguration` API を呼び出すと、使用する暗号化キーオプションを更新するためのアクセス権が付与されます。デフォルトでは、マネージド統合はデフォルトのマネージド統合マネージド暗号化キーを使用します。`PutDefaultEncryptionConfiguration` API を使用して、暗号化キー設定をいつでも更新できます。

さらに、 `DescribeDefaultEncryptionConfiguration` API コマンドを呼び出すと、デフォルトまたは指定されたリージョンの AWS アカウントの暗号化設定に関する情報が返されます。

マネージド統合によるend-to-endの暗号化の詳細については、「」を参照してください[マネージド統合の保管時のデータ暗号化](#)。

AWS KMS サービスの詳細については、「」を参照してください。 [AWS Key Management Service](#)

このステップで使用される APIs:

- `PutDefaultEncryptionConfiguration`
- `DescribeDefaultEncryptionConfiguration`

カスタムエンドポイントを登録する (必須)

デバイスと マネージド統合間の双方向通信により、デバイスコマンドの迅速なルーティング、物理デバイスと マネージド統合のモノのデジタル表現の状態の調整、デバイスデータの安全な送信が可能になります。マネージド統合に接続するには、デバイスにはトラフィックをルーティングするための専用エンドポイントが必要です。`RegisterCustomEndpoint` API を呼び出すと、サーバーの信頼の管理方法の設定に加えて、このエンドポイントが作成されます。カスタマーエンドポイントは、マネージド統合に接続するローカルハブまたは Wi-Fi デバイスのデバイス SDK に保存されます。

> **Note**
>
> クラウドに接続されたデバイスでは、このステップをスキップできます。

このステップで使用される APIs:

- `RegisterCustomEndpoint`

デバイスプロビジョニング (必須)

デバイスプロビジョニングは、将来の双方向通信のために、デバイスまたはデバイスのフリートとマネージド統合間のリンクを確立します。`CreateProvisioningProfile` API を呼び出してプロビジョニングテンプレートを作成し、証明書を要求します。プロビジョニングテンプレートは、プロビジョニングプロセス中にデバイスに適用される一連のリソースとポリシーを定義するドキュメントです。マネージド統合に初めて接続するときに、デバイスを登録して設定する方法を指定します。このプロセスにより、デバイスのセットアッププロセスが自動化され、各デバイスが安全かつ一貫して適切なアクセス許可、ポリシー、設定 AWS IoT で に統合されます。クレーム証明書は、フリートのプロビジョニング中に使用される一時的な証明書であり、エンドユーザーに配信される前に、一意のデバイス証明書が製造中にデバイスにプリインストールされていない場合にのみ使用されます。

次のリストは、デバイスのプロビジョニングワークフローと、それぞれの違いの概要を示しています。

- 単一デバイスのプロビジョニング
  - マネージド統合を使用して 1 つのデバイスをプロビジョニングする。
  - ワークフロー
    - `CreateManagedThing`: プロビジョニングテンプレートに基づいて、マネージド統合を使用して新しいマネージド型モノ (デバイス) を作成します。

- エンドデバイスソフトウェア開発キット (SDK) の詳細については、「」を参照してください[End Device SDK とは](#)。
- 単一デバイスのプロビジョニングの詳細については、「単一モノのプロビジョニング」を参照してください。

- クレームによるフリートのプロビジョニング
  - 承認されたユーザーによるプロビジョニング
    - エンドユーザーがマネージド統合にデバイスをプロビジョニングできるように、組織のデバイスプロビジョニングワークフロー (複数可) に固有の IAM ロールとポリシーを作成する必要があります。このワークフローの IAM ロールとポリシーの作成の詳細については、「デバイスをインストールするユーザーの IAM ポリシーとロールの作成」を参照してください。
    - ワークフロー
      - `CreateKeysAndCertificate`: デバイスの暫定クレーム証明書とキーを作成します。
      - `CreatePolicy`: デバイスのアクセス許可を定義するポリシーを作成します。
      - `AttachPolicy`: ポリシーを暫定クレーム証明書にアタッチします。
      - `CreateProvisioningTemplate`: デバイスのプロビジョニング方法を定義するプロビジョニングテンプレートを作成します。
      - `RegisterThing`: プロビジョニングテンプレートに基づいて IoT レジストリに新しいモノ (デバイス) を登録するデバイスプロビジョニングプロセスの一部。
      - さらに、プロビジョニングクレームを使用して初めて AWS IoT Core に接続すると、デバイスは MQTT または HTTPS プロトコルを使用して安全な通信を行います。このプロセス中、AWS IoT Core の内部メカニズムはクレームを検証し、プロビジョニングテンプレートを適用して、プロビジョニングプロセスを完了します。
    - 承認されたユーザーによるプロビジョニングの詳細については、「信頼できるユーザーによるプロビジョニング」を参照してください。
  - クレーム証明書を使用したプロビジョニング
    - マネージド統合との最初の問い合わせのために、各デバイスクレーム証明書にアタッチされたクレーム証明書のプロビジョニングポリシーを作成し、デバイス固有の証明書に置き換える必要があります。クレーム証明書によるプロビジョニングのワークフローを完了するには、ハードウェアシリアル番号を MQTT 予約済みトピックに送信する必要があります。
    - ワークフロー
      - `CreateKeysAndCertificate`: デバイスの暫定クレーム証明書とキーを作成します。
      - `CreatePolicy`: デバイスのアクセス許可を定義するポリシーを作成します。
      - `AttachPolicy`: ポリシーを暫定クレーム証明書にアタッチします。

- `CreateProvisioningTemplate`: デバイスのプロビジョニング方法を定義するプロビジョニングテンプレートを作成します。
- `RegisterThing`: プロビジョニングテンプレートに基づいて IoT レジストリに新しいモノ (デバイス) を登録するデバイスプロビジョニングプロセスの一部。
- さらに、プロビジョニングクレームを使用してデバイスが AWS IoT Core 初めて に接続すると、安全な通信のために MQTT または HTTPS プロトコルが使用されます。このプロセス中、 AWS IoT Coreの内部メカニズムはクレームを検証し、プロビジョニングテンプレートを適用して、プロビジョニングプロセスを完了します。

- クレーム証明書によるプロビジョニングの詳細については、「クレームによるプロビジョニング」を参照してください。

プロビジョニングテンプレートの詳細については、「プロビジョニングテンプレート」を参照してください。

このステップで使用される APIs:

- `CreateManagedThing`
- `CreateProvisioningProfile`
- `RegisterCACertificate`
- `CreatePolicy`
- `CreateThing`
- `AttachPolicy`
- `AttachThingPrincipal`
- `CreateKeysAndCertificate`
- `CreateProvisioningTemplate`

マネージドインテグレーション Hub SDK (必須)

最初の製造時に、デバイスのファームウェアにマネージド統合 Hub SDK を追加します。エンドユーザーのデバイスのプロビジョニングをサポートするために、先ほど作成した暗号化キー、カスタムエンドポイントアドレス、セットアップ認証情報、クレーム証明書を Hub SDK に追加します。

Hub SDK の詳細については、「」を参照してください。 Hub SDK アーキテクチャ

認証情報ロッカーとのデバイスの事前関連付け (オプション)

フルフィルメントプロセス中に、デバイスのバーコードをスキャンして、デバイスをエンドユーザーに事前に関連付けることができます。これにより、 `CreateManagedThing` API が自動的に呼び出され、マネージド型統合に保存されている物理デバイスのデジタル表現である Managed Thing が作成されます。さらに、 `CreateManagedThing` API はデバイスのプロビジョニング中に`deviceID`使用するために を自動的に返します。

利用可能な場合は、所有者の情報を`CreateManagedThing`リクエストメッセージに含めることができます。この所有者情報を含めると、セットアップ認証情報と事前定義されたデバイス機能を取得して、 マネージド統合に保存`managedThing`されている に含めることができます。これにより、マネージド統合を使用してデバイスまたはデバイスのフリートをプロビジョニングする時間を短縮できます。

所有者の情報が利用できない場合、`CreateManagedThing`API コールの `owner`パラメータは空白のままになり、デバイスがオンになっているときにデバイスのオンボーディング中に更新されます。

このステップで使用される APIs:

- `CreateManagedThing`

デバイスの検出とオンボーディング (必須)

エンドユーザーがデバイスをオンにするか、必要に応じてペアリングモードに設定すると、デバイスのタイプに応じて以下が発生します。

簡易セットアップ (SS)

エンドユーザーは IoT デバイスの電源を入れ、マネージド統合アプリを使用して QR コードをスキャンします。アプリは、マネージド統合クラウドにデバイスを登録し、IoT Hub に接続します。

ユーザーガイドによるセットアップ (UGS)

エンドユーザーはデバイスの電源を入れ、インタラクティブな手順に従ってマネージド統合にオンボードします。これには、IoT Hub のボタンを押す、マネージド統合アプリを使用する、ハブとデバイスの両方のボタンを押すことが含まれます。Simple Setup が失敗する場合は、この方法を使用します。

Device Command and Control

デバイスオンボーディングが完了したら、デバイスを管理するためのデバイスコマンドの送受信を開始できます。次のリストは、デバイスを管理するためのシナリオの一部を示しています。

- デバイスコマンドの送信： デバイスのライフサイクルを管理するためのコマンドをデバイスから送受信します。
  - 使用される APIsのサンプリング: `SendManagedThingCommand`。
- デバイスの状態の更新: 送信されたデバイスのライフサイクルとデバイスコマンドに基づいて、デバイスの状態を更新します。
  - 使用される APIs のサンプリング: `GetManagedThingState`、`ListManagedThingState`、`UpdateManagedThing`、および `DeleteManagedThing`。
- デバイスイベントの受信： マネージド統合に送信されるサードパーティーのクラウドプロバイダーから C2C デバイスに関するイベントを受信します。
  - 使用される APIs のサンプリング: `SendDeviceEvent`、`CreateLogLevel`、`CreateNotificationConfiguration`。

このステップで使用される APIs:

- `SendManagedThingCommand`
- `GetManagedThingState`
- `ListManagedThingState`
- `UpdateManagedThing`
- `DeleteManagedThing`
- `SendDeviceEvent`
- `CreateLogLevel`
- `CreateNotificationConfiguration`

API インデックス

マネージド統合 APIs「 マネージド統合 API リファレンスガイド」を参照してください。

AWS IoT Core APIs の詳細については、 AWS IoT Core API リファレンスガイドを参照してください。

Cloud-to-cloudデバイスのオンボーディング

次の手順では、サードパーティーのクラウドプロバイダーからマネージド統合にクラウドデバイスをオンボーディングするワークフローの概要を説明します。

> **Note**
>
> cloud-to-cloud デバイスの通知を設定するときに `RegisterCustomEndpoint` API を使用する必要はありません。この API は、直接接続されたデバイスまたはハブ接続されたデバイスをセットアップする場合にのみ必要です。

トピック

- モバイルアプリケーションの調整 (必須)
- 暗号化キーの設定 (オプション)
- アカウントのリンク (必須)
- デバイス検出 (必須)
- Device Command and Control
- API インデックス

モバイルアプリケーションの調整 (必須)

エンドユーザーにモバイルアプリケーションを提供すると、モバイルデバイスから直接デバイスを管理するための一貫したユーザーエクスペリエンスが容易になります。エンドユーザーは、モバイルアプリケーションで直感的なユーザーインターフェイスを活用して、さまざまなマネージド統合 APIs を呼び出して、デバイスを制御、管理、運用できます。モバイルアプリケーションは、所有者 ID、サポートされているデバイスプロトコル、デバイス機能などのデバイスメタデータをルーティングすることで、デバイス検出を支援できます。

さらに、モバイルアプリケーションは、マネージド統合 AWS アカウント の を、エンドユーザーのアカウントとサードパーティーのクラウドデバイスのデバイスデータを含むサードパーティーのクラウドとリンクするのに役立ちます。アカウントリンクにより、エンドユーザーのモバイルアプリケーション、 マネージド統合 AWS アカウント の 、サードパーティークラウド間でデバイスデータをシームレスにルーティングできます。

暗号化キーの設定 (オプション)

セキュリティは、エンドユーザー、マネージド型統合、サードパーティークラウド間でルーティングされるデータにとって最も重要です。デバイスデータを保護するためにサポートされる方法の 1 つは、データをルーティングするための安全な暗号化キーを使用したend-to-endの暗号化です。

マネージド統合のお客様には、暗号化キーを使用するための次の 2 つのオプションがあります。

- デフォルトのマネージド統合マネージド暗号化キーを使用します。
- AWS KMS key 作成した を指定します。

AWS KMS サービスの詳細については、「キー管理サービス (KMS)」を参照してください。

`PutDefaultEncryptionConfiguration` API を呼び出すと、使用する暗号化キーオプションを更新するためのアクセス権が付与されます。デフォルトでは、マネージド統合はデフォルトのマネージド統合マネージド暗号化キーを使用します。`PutDefaultEncryptionConfiguration` API を使用して、暗号化キー設定をいつでも更新できます。

さらに、 `DescribeDefaultEncryptionConfiguration` API コマンドを呼び出すと、デフォルトまたは指定されたリージョンの AWS アカウントの暗号化設定に関する情報が返されます。

このステップで使用される APIs:

- `PutDefaultEncryptionConfiguration`
- `DescribeDefaultEncryptionConfiguration`

アカウントのリンク (必須)

アカウントリンクは、エンドユーザーの認証情報を使用してクラウド環境をサードパーティープロバイダーのクラウドにリンクするプロセスです。このリンクは、クラウド環境とエンドユーザーのモバイルアプリケーション間でデバイスコマンドやその他のデバイス関連データをルーティングするために必要です。

アカウントリンクを開始するには、エンドユーザーはクラウド接続デバイスをサポートするモバイルアプリケーションで `StartAccountLinking` API コマンドを送信します。サードパーティーのクラウドはモバイルアプリケーションに URL を返し、エンドユーザーにサードパーティーのクラウドログイン認証情報を入力し、クラウド環境とエンドユーザーのモバイルアプリケーション間のアカウントリンクリクエストを承認するように求めます。

このステップで使用される APIs:

- `StartAccountLinking`

デバイス検出 (必須)

アカウントのリンクが完了すると、`StartDeviceDiscovery`API が自動的に呼び出されます。サードパーティークラウドは、エンドユーザーのサードパーティーアカウントに関連付けられたデバイスのリストを MQTT トピック に発行します`DevicesToApprove`。エンドユーザーは、マネージド統合へのデバイス登録のために、モバイルアプリケーションで選択したデバイスを承認します。次に、 `CreateManagedThing` API コマンドを使用して、登録されたデバイスごとにマネージド統合 Managed Thing が自動生成されます。マネージド型統合マネージド型モノは、マネージド型統合に保存されている物理デバイスのデジタル表現です。

このステップで使用される APIs:

- `StartDeviceDiscovery`
- `CreateManagedThing`

Device Command and Control

デバイスオンボーディングが完了したら、デバイスを管理するためのデバイスコマンドの送受信を開始できます。次のリストは、デバイスを管理するためのシナリオの一部を示しています。

- デバイスコマンドの送信: デバイスのライフサイクルを管理するためのコマンドをデバイスから送受信します。
  - 使用される APIsのサンプリング: `SendManagedThingCommand`。
- デバイスの状態の更新: デバイスのライフサイクルと送信されたデバイスコマンドに基づいて、デバイスの状態を更新します。
  - 使用される APIs のサンプリング: `GetManagedThingState`、`ListManagedThingState`、`UpdateManagedThing`、および `DeleteManagedThing`。
- デバイスイベントの受信: マネージド統合に送信されるサードパーティーのクラウドプロバイダーから C2C デバイスに関するイベントを受信します。
  - 使用される APIs のサンプリング: `SendDeviceEvent`、`CreateLogLevel`、`CreateNotificationConfiguration`。

このステップで使用される APIs:

- `SendManagedThingCommand`
- `GetManagedThingState`
- `ListManagedThingState`
- `UpdateManagedThing`
- `DeleteManagedThing`
- `SendDeviceEvent`
- `CreateLogLevel`
- `CreateNotificationConfiguration`

API インデックス

マネージド統合 APIs「 マネージド統合 API リファレンスガイド」を参照してください。

AWS IoT Core APIs の詳細については、 AWS IoT Core API リファレンスガイドを参照してください。

デバイスプロビジョニング

マネージド型統合にデバイスをプロビジョニングすることは、サードパーティー製デバイスを使用する場合に、物理デバイス、ローカルハブ、マネージド型統合、サードパーティークラウドプロバイダー間の双方向でほぼリアルタイムの通信を容易にするための、最初のオンボーディングプロセスにおける重要なステップです。

デバイスプロビジョニングとは

デバイスプロビジョニングは、シームレスなデバイスオンボーディングプロセスを容易にし、デバイスのライフサイクル全体を監督し、マネージド統合の他の側面からアクセスできるデバイス情報用の一元化されたリポジトリを確立します。マネージド統合は、さまざまなデバイスタイプを管理するための統合インターフェイスを提供し、ハブデバイスを介して間接的にリンクされたデバイスソフトウェア開発キット (SDK) またはcommercial-off-the-shelf (COTS) デバイスを介して直接接続されたファーストパーティのカスタマーデバイスに対応します。

マネージド統合の各デバイスには、デバイスタイプに関係なく、 と呼ばれるグローバルに一意の識別子がありますdeviceId。この識別子は、デバイスのライフサイクル全体のデバイスのオンボーディングと管理に使用されます。マネージド統合によって完全に管理され、すべてのマネージド統合にわたってその特定のデバイスに固有です AWS リージョン。デバイスが最初にマネージド統合に追加されると、この識別子が作成され、マネージド統合managedThingの にアタッチされます。managedThing は、物理デバイスのすべてのデバイスメタデータをミラーリングするための、マネージド統合内の物理デバイスのデジタル表現です。サードパーティーデバイスの場合、物理デバイスを表すマネージド統合に保存deviceIdされている に加えて、サードパーティークラウドに固有の独自の一意の識別子を持つ場合があります。

マネージド統合を使用してデバイスをプロビジョニングするには、次のオンボーディングフローが用意されています。

- 簡易セットアップ (SS): エンドユーザーは IoT デバイスに電源を入れ、デバイスの製造元アプリケーションを使用して QR コードをスキャンします。その後、デバイスはマネージド統合クラウドに登録され、IoT ハブに接続します。
- ユーザーガイド設定 (UGS): エンドユーザーはデバイスの電源を入れ、インタラクティブな手順に従ってマネージド統合にオンボードします。これには、IoT ハブのボタンを押す、デバイスメーカーアプリを使用する、ハブとデバイスの両方のボタンを押すことが含まれます。このメソッドは、簡易セットアップが失敗した場合に使用できます。

> **Note**
>
> マネージド統合のデバイスプロビジョニングワークフローは、デバイスのオンボーディング要件に依存しません。マネージド統合は、デバイスタイプやデバイスプロトコルに関係なく、デバイスをオンボーディングおよび管理するための効率的なユーザーインターフェイスを提供します。

デバイスとデバイスプロファイルのライフサイクル

デバイスとデバイスプロファイルのライフサイクルを管理することで、デバイスのフリートが安全で効率的に実行されるようにします。

トピック

- デバイス
- デバイスプロファイル

デバイス

最初のオンボーディング手順では、 と呼ばれる物理デバイスのデジタル表現managedThingが作成されます。は、すべてのリージョンにわたるマネージド統合のデバイスを識別するためのグローバル一意識別子managedThingを提供します。デバイスは、 マネージド統合またはサードパーティーデバイスのサードパーティークラウドとのリアルタイム通信のプロビジョニング中にローカルハブとペアになります。デバイスは、 managedThingなどの のパブリック APIs の ownerパラメータによって識別される所有者にも関連付けられますGetManagedThing。デバイスは、デバイスのタイプに基づいて、対応するデバイスプロファイルにリンクされます。

> **Note**
>
> 物理デバイスが異なる顧客で複数回プロビジョニングされている場合、複数のレコードを持つことがあります。

デバイスライフサイクルは、 CreateManagedThing API を使用したマネージド統合managedThingでの の作成から始まり、お客様が DeleteManagedThing API managedThing を使用して を削除したときに終了します。デバイスのライフサイクルは、次のパブリック APIs。

- CreateManagedThing
- ListManagedThings
- GetManagedThing
- UpdateManagedThing
- DeleteManagedThing

デバイスプロファイル

デバイスプロファイルは、電球やドアベルなどの特定のタイプのデバイスを表します。これは製造元に関連付けられており、デバイスの機能が含まれています。デバイスプロファイルは、 マネージド統合によるデバイス接続セットアップリクエストに必要な認証マテリアルを保存します。使用される認証マテリアルは、デバイスのバーコードです。

デバイスの製造プロセス中に、製造元はデバイスプロファイルをマネージド統合に登録できます。これにより、製造元はオンボーディングとプロビジョニングのワークフロー中にマネージド統合からデバイスに必要なマテリアルを取得できます。デバイスプロファイルからのメタデータは、物理デバイスに保存されるか、デバイスのラベル付けに出力されます。デバイスプロファイルのライフサイクルは、製造元がマネージド統合でデバイスプロファイルを削除したときに終了します。

データモデル

データモデルは、システム内でのデータの整理方法の組織階層を表します。さらに、デバイス実装全体でend-to-endの通信をサポートしています。マネージド統合には、2 つのデータモデルが使用されます。マネージド統合データモデルと Matter データモデルの の AWS実装。どちらも類似点を共有しますが、以下のトピックで説明されている微妙な違いも共有します。

サードパーティーデバイスの場合、エンドユーザー、マネージド統合、サードパーティーのクラウドプロバイダー間の通信には両方のデータモデルが使用されます。デバイスコマンドやデバイスイベントなどのメッセージを 2 つのデータモデルから変換するには、Cloud-to-Cloud Connector 機能を活用します。

トピック

- マネージド統合データモデル
- AWS Matter データモデルの実装

マネージド統合データモデル

マネージド統合データモデルは、エンドユーザーとマネージド統合間のすべての通信を管理します。

デバイス階層

`endpoint` および `capability` データ要素は、マネージド統合データモデル内のデバイスを記述するために使用されます。

`endpoint`

は、機能によって提供される論理インターフェイスまたはサービスendpointを表します。

```
{
    "endpointId": { "type":"string" },
    "name": { "$ref": "aws.name" },       // Optional human readable name
    "capabilities": Capability[]
}
```

`Capability`

はデバイスの機能capabilityを表します。

```
{
    "$id": "string",                // Schema identifier (e.g. namespace or
 resourceType)
    "name": "string",               // Human readable name
    "version": "string",            // e.g. 1.0
    "properties": Property[],
    "actions": Action[],
    "events": Event[]
}
```

capability データ要素には、その項目を構成する 、property、 actionの 3 つの項目がありますevent。デバイスを操作およびモニタリングするために使用できます。

- プロパティ: 調光可能な光の現在の明るさレベルの属性など、デバイスが保持する状態。
  -
    ```
    {
        "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.ssssssZ
        "mutable": boolean,
        "retrievable": boolean,
        "reportable": boolean

    }
    ```
- アクション: ドアロックでドアをロックするなど、実行できるタスク。アクションはレスポンスと結果を生成する場合があります。
  -
    ```
    {
        "name": { "$ref": "aws.name" }, //required
        "parameters": Map<String name, JSONNode value>,
        "responseCode": HTTPResponseCode,
        "errors": {
            "code": "string",
            "message": "string"
        }
    }
    ```
- イベント: 基本的には、過去の状態遷移の記録です。は現在の状態propertyを表しますが、イベントは過去のジャーナルであり、単調に増加するカウンター、タイムスタンプ、優先度が含まれ

ます。これにより、状態遷移と、 では容易に達成できないデータモデリングをキャプチャできますproperty。

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

AWS Matter データモデルの実装

AWS Matter Data Model の 実装は、マネージド統合とサードパーティーのクラウドプロバイダー間のすべての通信を管理します。

詳細については、「 Matter Data Model: Developer Resources」を参照してください。

デバイス階層

デバイスの説明に使用されるデータ要素には、endpoint、、および の 2 つがありますcluster。

endpoint

は、機能によって提供される論理インターフェイスまたはサービスendpointを表します。

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

cluster

はデバイスの機能clusterを表します。

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

cluster データ要素には、その項目を構成する 、attribute、 commandの 3 つの項目がありますevent。デバイスを操作およびモニタリングするために使用できます。

- 属性: 調光可能な光の現在の明るさレベルの属性など、デバイスが保持する状態。
  - ```
    {
        "id" (hexadecimalString): (JsonNode) value
    }
    ```
- コマンド: ドアロックでドアをロックするなど、実行できるタスク。コマンドはレスポンスと結果を生成する場合があります。
  - ```
    "id": {
        "fieldId": "fieldValue",
        ...
        "responseCode": HTTPResponseCode,
        "errors": {
            "code": "string",
            "message": "string"
        }
    }
    ```
- イベント: 基本的には、過去の状態遷移の記録です。は現在の状態attributesを表しますが、イベントは過去のジャーナルであり、単調に増加するカウンター、タイムスタンプ、優先度が含まれます。これにより、状態遷移と、 では容易に達成できないデータモデリングをキャプチャできますattributes。
  - ```
    "id": {
        "fieldId": "fieldValue",
        ...
    }
    ```
    ```

IoT デバイスコマンドとイベントを管理する

デバイスコマンドを使用すると、物理デバイスをリモートで管理して、重要なセキュリティ、ソフトウェア、ハードウェアの更新の実行に加えて、デバイスを完全に制御できます。多数のデバイスでは、デバイスがコマンドを実行するタイミングを知ることで、デバイスの実装全体を監視できます。デバイスコマンドまたは自動更新により、デバイス状態の変更がトリガーされ、新しいデバイスイベントが作成されます。このデバイスイベントは、エンドユーザーを更新するためにカスタマー管理の送信先に自動送信される通知をトリガーします。

トピック

- Device コマンド
- デバイスイベント

Device コマンド

コマンドリクエストは、お客様がデバイスに送信するコマンドです。コマンドリクエストには、電球をオンにするなどのアクションを指定するペイロードが含まれています。デバイスコマンドを送信するには、マネージド統合によって SendManagedThingCommand API がエンドユーザーに代わって呼び出され、コマンドリクエストがデバイスに送信されます。

SendManagedThingCommand API オペレーションの詳細については、SendManagedThingCommand」を参照してください。

UpdateState のアクション

ライトがオンになる時間などのデバイスの状態を更新するには、 SendManagedThingCommand API を呼び出すときに UpdateStateアクションを使用します。で更新するデータモデルプロパティと新しい値を指定しますparameters。次の例は、電球OnTimeの を に更新する SendManagedThingCommand API リクエストを示しています5。

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

```
          "version": "1",
          "actions": [
            {
              "name": "UpdateState",
              "parameters": {
                "OnTime": 5
              }
            }
          ]
        }
      ]
    }
  ]
}
```

ReadState のアクション

すべてのデータモデルプロパティの現在の値を含むデバイスの最新の状態を取得するには、SendManagedThingCommand API を呼び出すときに ReadStateアクションを使用します。ではpropertiesToRead、次のオプションを使用できます。

- 特定のデータモデルプロパティを指定して、照明がオンかオフかOnOffを判断するなど、 で最新の値を取得します。
- ワイルドカード演算子 (*) を使用して、機能のすべてのデバイス状態プロパティを読み取ります。

以下の例は、 ReadStateアクションを使用した SendManagedThingCommand API リクエストの両方のシナリオを示しています。

```
{
  "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": [ "*" ]
              }
            }
          ]
        }
      ]
    }
  ]}
```

デバイスイベント

デバイスイベントには、デバイスの現在の状態が含まれます。これは、デバイスが状態を変更したか、状態が変更されていない場合でもその状態を報告していることを意味します。これには、データモデルで定義されたプロパティレポートとイベントが含まれます。イベントには、マシンサイクルが完了したか、サーモスタットがエンドユーザーによって設定された目標温度に達した可能性があります。

デバイスの状態とは

デバイスの状態は、リソースのコレクションによって記述されます。リソースとは、スキーマによって記述される一連の機能を指します。各リソース状態の変更には、関連するバージョン番号があります。デバイスに関連付けられたリソースは、機能がデバイスに追加または削除されると、時間の経過とともに変化する可能性があります。

デバイスイベント通知

エンドユーザーは、特定のデバイスイベントの更新用に作成する特定のカスタマー管理の送信先にサブスクライブできます。カスタマー管理の送信先を作成するには、 `CreateDestination` API を呼び出します。デバイスイベントがデバイスによって管理統合に報告されると、カスタマー管理の送信先が存在する場合は通知されます。

マネージド統合通知を設定する

マネージド統合通知は、顧客へのすべての通知を管理し、デバイスに関する更新とインサイトを配信するためのリアルタイムの通信を促進します。デバイスイベント、デバイスライフサイクル、デバイス状態を顧客に通知するかどうかにかかわらず、マネージド統合通知はカスタマーエクスペリエンス全体を向上させる上で重要な役割を果たします。実用的な情報を提供することで、お客様は情報に基づいた意思決定を行い、リソース使用率を最適化できます。

トピック

- マネージド統合通知の設定

マネージド統合通知の設定

マネージド統合通知を設定するには、次の 4 つのステップを実行します。

Amazon Kinesis データストリームを作成する

Kinesis データストリームを作成するには、「Kinesis データストリームの作成と管理」で説明されているステップに従います。

現在、Amazon Kinesis データストリームのみが、マネージド統合通知のカスタマーマネージド送信先のオプションとしてサポートされています。

Amazon Kinesis ストリームアクセスロールを作成する

作成した Kinesis ストリームへのアクセス許可を持つ AWS Identity and Access Management アクセスロールを作成する

詳細については、「 AWS Identity and Access Managementユーザーガイド」の「IAM ロールの作成」を参照してください。

`CreateDestination` API を呼び出す

Amazon Kinesis データストリームとストリームアクセスロールを作成したら、`CreateDestination` API を呼び出して、マネージド統合通知がルーティングされるカスタマーマネージドの送信先を作成します。`deliveryDestinationArn` パラメータには、新しい Amazon Kinesis データストリーム`arn`の を使用します。

```
{
    "DeliveryDestinationArn": "Your Kinesis arn"
```

```
    "DeliveryDestinationType": "KINESIS"
    "Name": "DestinationName"
    "ClientToken": "Random string"
    "RoleArn": "Your Role arn"
}
```

CreateNotificationConfiguration API を呼び出す

最後に、Amazon Kinesis データストリームで表される カスタマー管理の宛先に通知をルーティングすることで、選択したイベントタイプを通知する通知設定を作成します。CreateNotificationConfiguration API を呼び出して通知設定を作成します。destinationName パラメータで、CreateDestinationAPI を使用してカスタマー管理の送信先を作成したときに最初に作成したのと同じ送信先名を使用します。

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

マネージド統合通知でモニタリングできるイベントタイプを以下に示します。

- コネクタの関連付けステータスを示します。
- DEVICE_COMMAND
  - SendManagedThing API コマンドのステータス。この有効な値は成功または失敗します。

```
{
  "version":"0",
  "messageId":"6a7e8feb-b491-4cf7-a9f1-bf3703467718",
  "messageType":"DEVICE_EVENT",
  "source":"aws.iotmanagedintegrations",
  "customerAccountId":"123456789012",
  "timestamp":"2017-12-22T18:43:48Z",
  "region":"ca-central-1",
  "resources":[
    "arn:aws:iotmanagedintegrations:ca-
central-1:123456789012:managedThing/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
  - Web Real-Time Communication (WebRTC) からのコマンドリクエスト。

    WebRTC 標準では、2 つのピア間の通信が可能です。これらのピアは、リアルタイムの動画、オーディオ、および任意のデータを送信できます。マネージド統合は WebRTC をサポートし、顧客のモバイルアプリケーションとエンドユーザーのデバイス間のこれらのタイプのストリーミングを可能にします。WebRTC 標準の詳細については、「」を参照してくださいhttps://webrtc.org/。

```
{
  "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:managedThing/6a7e8feb-b491-4cf7-a9f1-bf3703467718"
  ],
  "payload":{
    "endpoints":[{
      "endpointId":"1",
      "capabilities":[{
        "id":"aws.DoorLock",
        "name":"Door Lock",
        "version":"1.0"
      }]
    }]
  }
}
```

- DEVICE_EVENT
  - デバイスイベントの発生に関する通知。

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

- DEVICE_LIFE_CYCLE
  - デバイスライフサイクルのステータス。

```
{
  "version": "1.0.0",
  "messageId": "8d1e311a473f44f89d821531a0907b05",
  "messageType": "DEVICE_LIFE_CYCLE",
  "source": "aws.iotmanagedintegrations",
  "customerAccountId": "123456789012",
  "timestamp": "2024-11-14T19:55:57.568284645Z",
  "region": "us-west-2",
  "resources": [
    "arn:aws:iotmanagedintegrations:us-west-2:123456789012:managed-thing/
d5c280b423a042f3933eed09cf408657"
    ],
```

```
  "payload": {
    "deviceDetails": {
      "id": "d5c280b423a042f3933eed09cf408657",
      "arn": "arn:aws:iotmanagedintegrations:us-west-2:123456789012:managed-thing/
d5c280b423a042f3933eed09cf408657",
      "createdAt": "2024-11-14T19:55:57.515841147Z",
      "updatedAt": "2024-11-14T19:55:57.515841559Z"
    },
    "status": "UNCLAIMED"
  }
}
```

- DEVICE_OTA
  - デバイス OTA 通知。
- DEVICE_STATE
  - デバイスの状態が更新されたときの通知。

```
{
  "messageType": "DEVICE_STATE",
  "source": "aws.iotmanagedintegrations",
  "customerAccountId": "123456789012",
  "timestamp": "1731623291671",
  "resources": [
    "arn:aws:iotmanagedintegrations:us-west-2:123456789012: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"
            }
          }
        ]}
```

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

マネージド統合 Hub SDK

この章では、 マネージド統合 Hub SDK を使用した IoT ハブデバイスのオンボーディングと制御について説明します。マネージド統合は現在パブリックプレビュー中です。マネージド統合 Hub SDK の最新バージョンをダウンロードするには、マネージド統合コンソールからお問い合わせください。マネージド統合 エンドデバイス SDK の詳細については、「」を参照してくださいマネージド統合 エンドデバイス SDK。

Hub SDK アーキテクチャ

デバイスオンボーディングの概要

マネージド統合の使用を開始する前に、Hub SDK コンポーネントがデバイスのオンボーディングをどのようにサポートしているかを確認してください。このセクションでは、コアプロビジョナーとプロトコル固有のプラグインが連携してデバイスの認証、通信、セットアップを処理する方法など、デバイスのオンボーディングに必要な重要なアーキテクチャコンポーネントについて説明します。

デバイスオンボーディング用の Hub SDK コンポーネント

SDK コンポーネント

- コアプロビジョナー
- プロトコル固有のプロビジョナープラグイン
- プロトコル固有のミドルウェア

コアプロビジョナー

コアプロビジョナーは、IoT ハブデプロイでデバイスのオンボーディングを調整する中央コンポーネントです。マネージド統合とプロトコル固有のプロビジョナープラグイン間のすべての通信を調整し、安全で信頼性の高いデバイスのオンボーディングを実現します。デバイスをオンボードすると、コアプロビジョナーは認証フローを処理し、MQTT メッセージングを管理し、以下の機能を通じてデバイスリクエストを処理します。

MQTT 接続

クラウドトピックの発行とサブスクライブのために MQTT ブローカーとの接続を作成します。

メッセージキューとハンドラー

受信するデバイスの追加および削除リクエストを順番に処理します。

プロトコルプラグインインターフェイス

認証モードと無線結合モードを管理することで、デバイスオンボーディング用のプロトコル固有のプロビジョナープラグインと連携します。

CDMB へのプロセス間通信 (IPC) クライアント

プロトコル固有の CDMB プラグインからデバイス機能レポートを受信し、マネージド統合に転送します。

プロトコル固有のプロビジョナープラグイン

プロトコル固有のプロビジョナープラグインは、さまざまな通信プロトコルのデバイスのオンボーディングを管理するライブラリです。各プラグインは、コマンドをコアプロビジョナーから IoT デバイスのプロトコル固有のアクションに変換します。これらのプラグインは以下を実行します。

- プロトコル固有のミドルウェアの初期化
- コアプロビジョナーリクエストに基づく無線結合モード設定
- ミドルウェア API コールによるデバイスの削除

プロトコル固有のミドルウェア

プロトコル固有のミドルウェアは、デバイスプロトコルとマネージド統合間の変換レイヤーとして機能します。このコンポーネントは、プロビジョナープラグインからコマンドを受信してプロトコルスタックに送信し、デバイスからレスポンスを収集してシステム経由でルーティングしながら、双方向の通信を処理します。

デバイスオンボーディングフロー

Hub SDK を使用してデバイスをオンボードするときに発生するオペレーションのシーケンスを確認します。このセクションでは、オンボーディングプロセス中にコンポーネントがどのように相互作用するかを示し、サポートされているオンボーディング方法の概要を説明します。

オンボーディングフロー

- 簡易セットアップ (SS)

- ゼロタッチセットアップ (ZTS)
- ユーザーガイドによるセットアップ (UGS)

簡易セットアップ (SS)

エンドユーザーは IoT デバイスの電源を入れ、デバイスの製造元アプリケーションを使用して QR コードをスキャンします。その後、デバイスはマネージド統合クラウドに登録され、IoT ハブに接続します。

ゼロタッチセットアップ (ZTS)

ゼロタッチセットアップ (ZTS) は、サプライチェーンの上流にあるデバイスを事前に関連付けることで、デバイスのオンボーディングを合理化します。例えば、エンドユーザーがデバイスの QR コードをスキャンする代わりに、このステップは以前に完了し、デバイスをカスタマーアカウントに事前リンクします。たとえば、このステップはフルフィルメントセンターで完了できます。

エンドユーザーがデバイスを受信して電源を入れると、マネージド統合クラウドに自動的に登録され、追加のセットアップアクションを必要とせずに IoT ハブに接続します。

ユーザーガイドによるセットアップ (UGS)

エンドユーザーはデバイスの電源を入れ、インタラクティブな手順に従ってマネージド統合にオンボードします。これには、IoT ハブのボタンを押す、デバイス製造元アプリを使用する、ハブとデバイスの両方のボタンを押すことが含まれます。Simple Setup が失敗した場合、このメソッドを使用できます。

デバイスコントロールの概要

マネージド統合は、デバイス登録、コマンド実行、制御を処理します。ベンダーやプロトコルに依存しないデバイス管理を使用して、デバイス固有のプロトコルを知らなくてもエンドユーザーエクスペリエンスを構築できます。

デバイスコントロールを使用すると、電球の明るさやドアの位置などのデバイスの状態を表示および変更できます。この機能は状態変更のイベントを出力し、分析、ルール、モニタリングに使用できます。

主な特徴

デバイスの状態を変更または読み取る

デバイスタイプに基づいてデバイス属性を表示および変更します。以下にアクセスできます。

- デバイスの状態： 現在のデバイス属性値
- 接続状態： デバイス到達可能性ステータス
- ヘルスステータス： バッテリーレベルや信号強度 (RSSI) などのシステム値

状態変更通知

電球の明るさの調整やドアロックのステータスの変化など、デバイス属性や接続状態が変化したときにイベントを受信します。

オフラインモード

デバイスは、インターネット接続がなくても、同じ IoT ハブ上の他のデバイスと通信します。デバイスの状態は、接続が再開されるとクラウドと同期します。

状態同期

複数のソース、デバイス製造元アプリ、手動デバイス調整からの状態の変化を追跡します。

マネージド統合を通じてデバイスを制御するために必要な Hub SDK コンポーネントとプロセスを確認します。このトピックでは、エッジエージェント、共通データモデルブリッジ (CDMB)、プロトコル固有のプラグインが連携してデバイスコマンドを処理し、デバイスの状態を管理し、さまざまなプロトコル間でレスポンスを処理する方法について説明します。

デバイス制御用の Hub SDK コンポーネント

Hub SDK アーキテクチャでは、以下のコンポーネントを使用して、IoT 実装でデバイス制御コマンドを処理およびルーティングします。各コンポーネントは、クラウドコマンドをデバイスアクションに変換し、デバイスの状態を管理し、レスポンスを処理する特定の役割を果たします。以下のセクションでは、これらのコンポーネントがデプロイでどのように連携するかについて詳しく説明します。

Hub SDK は、以下のコンポーネントで構成され、IoT ハブでのデバイスのオンボーディングと制御を容易にします。

プライマリコンポーネント：

Edge エージェント

IoT ハブとマネージド統合間のゲートウェイとして機能します。

共通データモデルブリッジ (CDMB)

AWS データモデルと Z-Wave や Zigbee などのローカルプロトコルデータモデルを変換します。これには、コア CDMB プラグインとプロトコル固有の CDMB プラグインが含まれています。

プロビジョナー

デバイスの検出とオンボーディングを処理します。これには、プロトコル固有のオンボーディングタスク用のコアプロビジョナーとプロトコル固有のプロビジョナープラグインが含まれています。

セカンダリコンポーネント

ハブオンボーディング

安全なクラウド通信のために、クライアント証明書とキーを使用してハブをプロビジョニングします。

MQTT プロキシ

マネージド統合クラウドへの MQTT 接続を提供します。

ロガー

ローカルまたはマネージド統合クラウドにログを書き込みます。

デバイスコントロールフロー

次の図は、エンドユーザーが Zigbee スマートプラグをオンにする方法を説明することでend-to-end のデバイス制御フローを示しています。

マネージド統合へのハブのオンボーディング

必要なディレクトリ構造、証明書、デバイス設定ファイルを設定して、マネージド統合と通信するようにハブデバイスをセットアップします。このセクションでは、ハブオンボーディングサブシステムコンポーネントがどのように連携するか、証明書と設定ファイルを保存する場所、デバイス設定ファ

イルを作成および変更する方法、ハブプロビジョニングプロセスを完了する手順について説明します。

Hub オンボーディングサブシステム

ハブオンボーディングサブシステムは、これらのコアコンポーネントを使用してデバイスのプロビジョニングと設定を管理します。

Hub オンボーディングコンポーネント

ハブの状態、プロビジョニングアプローチ、および認証マテリアルを調整することで、ハブのオンボーディングプロセスを管理します。

デバイス設定ファイル

以下を含む重要なハブ設定データをデバイスに保存します。

- デバイスプロビジョニング状態 (プロビジョニング済みまたは非プロビジョニング)
- 証明書とキーの場所
- 認証情報 MQTT プロキシなどのその他の SDK プロセスは、このファイルを参照してハブの状態と接続設定を決定します。

証明書ハンドラーインターフェイス

デバイス証明書とキーを読み書きするためのユーティリティインターフェイスを提供します。このインターフェイスを実装して、以下を操作できます。

- ファイルシステムストレージ
- ハードウェアセキュリティモジュール (HSM)
- 信頼できるプラットフォームモジュール (TPM)
- カスタムセキュアストレージソリューション

MQTT プロキシコンポーネント

以下を使用して、device-to-cloud通信を管理します。

- プロビジョニングされたクライアント証明書とキー
- 設定ファイルからのデバイス状態情報
- マネージド統合への MQTT 接続

次の図は、ハブオンボーディングサブシステムのアーキテクチャとそのコンポーネントを示しています。を使用していない場合は AWS IoT Greengrass、図のそのコンポーネントを無視できます。

Hub オンボーディングの設定

フリートプロビジョニングのオンボーディングプロセスを開始する前に、ハブデバイスごとにこれらのセットアップステップを完了します。このセクションでは、マネージド型モノの作成、ディレクトリ構造の設定、必要な証明書の設定方法について説明します。

セットアップステップ

- ステップ 1: カスタムエンドポイントを登録する
- ステップ 2: プロビジョニングプロファイルを作成する
- ステップ 3: マネージド型モノを作成する (フリートプロビジョニング)
- ステップ 4: ディレクトリ構造を作成する
- ステップ 5: ハブデバイスに認証マテリアルを追加する
- ステップ 6: デバイス設定ファイルを作成する
- ステップ 7: 設定ファイルをハブにコピーする

ステップ 1: カスタムエンドポイントを登録する

マネージド統合とデータを交換するためにデバイスが使用する専用の通信エンドポイントを作成します。このエンドポイントは、device-to-cloudメッセージングの安全な接続ポイントを確立します。

エンドポイントを登録するには

- RegisterCustomEndpoint API を使用して、device-to-managed統合通信用のエンドポイントを作成します。

  RegisterCustomEndpoint リクエストの例

```
curl 'https://api.iotmanagedintegrations.AWS-REGION.api.aws/custom-endpoint' \
  -H 'Content-Encoding: amz-1.0' \
  -H 'Content-Type: application/json; charset=UTF-8' \
  -H 'X-Amz-Target: iotmanagedintegrations.RegisterCustomEndpoint' \
  -H 'X-Amz-Security-Token: $AWS_SESSION_TOKEN' \
  --user "$AWS_ACCESS_KEY_ID:$AWS_SECRET_ACCESS_KEY" \
  --aws-sigv4 "aws:amz:AWS-REGION:iotmanagedintegrations" \
  -X POST --data '{}'
```

レスポンス:

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

> **Note**
> エンドポイントアドレスを保存します。今後のデバイス通信に必要になります。

エンドポイント情報を返すには、 `GetCustomEndpoint` API を使用します。

詳細については、「 マネージド統合 API リファレンス」のRegisterCustomEndpoint API」とGetCustomEndpoint API」を参照してください。

ステップ 2: プロビジョニングプロファイルを作成する

プロビジョニングプロファイルには、デバイスがマネージド統合に接続するために必要なセキュリティ認証情報と設定が含まれています。

フリートプロビジョニングプロファイルを作成するには

- CreateProvisioningProfile API を呼び出して、以下を生成します。
  - デバイス接続設定を定義するプロビジョニングテンプレート
  - デバイス認証のクレーム証明書とプライベートキー

> **Important**
> クレーム証明書、プライベートキー、テンプレート ID を安全に保存します。これらの認証情報は、デバイスをマネージド統合にオンボードするために必要です。これらの認証情報を紛失した場合は、新しいプロビジョニングプロファイルを作成する必要があります。

`CreateProvisioningProfile` リクエストの例

```
curl https://api.iotmanagedintegrations.AWS-REGION.api.aws/provisioning-profiles' \
  -H 'Content-Encoding: amz-1.0' \
  -H 'Content-Type: application/json; charset=UTF-8' \
  -H 'X-Amz-Target: iotmanagedintegrations.CreateProvisioningProfile' \
  -H "X-Amz-Security-Token: $AWS_SESSION_TOKEN" \
  --user "$AWS_ACCESS_KEY_ID:$AWS_SECRET_ACCESS_KEY" \
  --aws-sigv4 "aws:amz:AWS-REGION:iotmanagedintegrations" \
  -X POST --data '{ "ProvisioningType": "FLEET_PROVISIONING", "Name": "PROFILE-NAME" }'
```

レスポンス:

```
{
"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"
}
```

ステップ 3: マネージド型モノを作成する (フリートプロビジョニング)

`CreateManagedThing` API を使用して、ハブデバイスのマネージド型モノを作成します。各ハブには、一意の認証マテリアルを持つ独自のマネージド型モノが必要です。詳細については、「 マネージド統合 API リファレンス」のCreateManagedThing API」を参照してください。

マネージド型モノを作成するときは、次のパラメータを指定します。

- `Role`: この値を に設定しますCONTROLLER。
- `AuthenticationMaterial`: 次のフィールドを含めます。
  - SN: このデバイスの一意のシリアル番号
  - UPC: このデバイスのユニバーサル製品コード
- `Owner`: このマネージドモノの所有者識別子。

> **Important**
> 各デバイスには、認証マテリアルに一意のシリアル番号 (SN) が必要です。

CreateManagedThing リクエストの例：

```
{
 "Role": "CONTROLLER",
 "Owner": "ThingOwner1",
 "AuthenticationMaterialType": "WIFI_SETUP_QR_BAR_CODE",
 "AuthenticationMaterial": "SN:123456789524;UPC:829576019524"
}
```

詳細については、「 マネージド統合 API リファレンス」の「CreateManagedThing」を参照してください。

(オプション) マネージドモノを取得する

マネージドモノProvisioningStatusの は、続行するUNCLAIMED前に である必要があります。GetManagedThing API を使用して、マネージド型モノが存在し、プロビジョニングの準備が整っていることを確認します。詳細については、「 マネージド統合 API リファレンス」のGetManagedThing」を参照してください。

ステップ 4: ディレクトリ構造を作成する

設定ファイルと証明書のディレクトリを作成します。デフォルトでは、ハブオンボーディングプロセスは を使用します/data/aws/iotmi/config/iotmi_config.json。

設定ファイルで証明書とプライベートキーのカスタムパスを指定できます。このガイドでは、デフォルトのパス を使用します/data/aws/iotmi/certs。

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

/data/
    aws/
        iotmi/
            config/
            certs/
```

ステップ 5: ハブデバイスに認証マテリアルを追加する

証明書とキーをハブデバイスにコピーし、デバイス固有の設定ファイルを作成します。これらのファイルは、プロビジョニングプロセス中にハブとマネージド統合間の安全な通信を確立します。

クレーム証明書とキーをコピーするには

- `CreateProvisioningProfile` API レスポンスからハブデバイスにこれらの認証ファイルをコピーします。
  - `claim_cert.pem`: クレーム証明書 (すべてのデバイス共通)
  - `claim_pk.key`: クレーム証明書のプライベートキー

  両方のファイルを `/data/aws/iotmi/certs` ディレクトリに配置します。

  > **Note**
  > セキュアストレージを使用する場合は、これらの認証情報をファイルシステムではなく、安全なストレージの場所に保存します。詳細については、「セキュアストレージ用のカスタム証明書ハンドラーを作成する」を参照してください。

ステップ 6: デバイス設定ファイルを作成する

一意のデバイス識別子、証明書の場所、プロビジョニング設定を含む設定ファイルを作成します。SDK はハブのオンボーディング中にこのファイルを使用して、デバイスを認証し、プロビジョニングステータスを管理し、接続設定を保存します。

> **Note**
> 各ハブデバイスには、一意のデバイス固有の値を持つ独自の設定ファイルが必要です。

次の手順を使用して、設定ファイルを作成または変更し、ハブにコピーします。

- 設定ファイル (フリートプロビジョニング) を作成または変更します。

  デバイス設定ファイルで以下の必須フィールドを設定します。

- 証明書パス
  1. `iot_claim_cert_path`: クレーム証明書の場所 (`claim_cert.pem`)
  2. `iot_claim_pk_path`: プライベートキーの場所 (`claim_pk.key`)
  3. Secure Storage Cert Handler を実装するときに両方のフィールドに `SECURE_STORAGE`を使用する
- 接続の設定
  1. `fp_template_name`: 以前の`ProvisioningProfile`名前。
  2. `endpoint_url`: `RegisterCustomEndpoint` API レスポンスからのマネージド統合エンドポイント URL (リージョン内のすべてのデバイスで同じ) 。
- デバイス識別子
  1. SN: `CreateManagedThing` API コールに一致するデバイスのシリアル番号 (デバイスごとに一意)
  2. UPC`CreateManagedThing` API コールのユニバーサル製品コード (この製品のすべてのデバイスと同じ)

```
{
    "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"
    }
}
```

設定ファイルの内容

`iotmi_config.json` ファイルの内容を確認します。

内容

キー	値	顧客によって追加されましたか？	コメント
iot_provisioning_method	FLEET_PROVISIONING	はい	使用するプロビジョニング方法を指定します。
iot_claim_cert_path	または を指定するファイルパスSECURE_STORAGE 。例: /data/aws/iotmi/certs/claim_cert.pem	はい	または を使用するファイルパスを指定しますSECURE_STORAGE 。
iot_claim_pk_path	または を指定するファイルパスSECURE_STORAGE 。例: /data/aws/iotmi/certs/claim_pk.pem	はい	または を使用するファイルパスを指定しますSECURE_STORAGE 。
fp_template_name	フリートプロビジョニングテンプレート名は、以前にProvisioningProfile 使用された の名前と等しくなければなりません。	はい	以前にProvisioningProfile 使用された の名前と同じ
endpoint_url	マネージド統合のエンドポイント URL。	はい	デバイスは、この URL を使用してマネージド統合クラウドに接続します。この情報を取得するには、RegisterCustomEndpoint API を使用します。
SN	デバイスのシリアル番号。例えば、AIDACKCEVSQ6C2EXAMPLE 。	はい	この一意の情報をデバイスごとに指定する必要があります。

キー	値	顧客によって追加されましたか？	コメント
UPC	デバイスユニバーサル製品コード。例えば、841667145075 。	はい	デバイスにこの情報を指定する必要があります。
managed_thing_id	マネージドモノの ID。	いいえ	この情報は、ハブプロビジョニング後のオンボーディングプロセスによって後で追加されます。
iot_provisioning_state	プロビジョニング状態。	はい	プロビジョニング状態は に設定する必要がありますNOT_PROVISIONED 。
iot_permanent_cert_path	IoT 証明書パス。例えば、/data/aws/iotmi/iot_cert.pem 。	いいえ	この情報は、ハブプロビジョニング後のオンボーディングプロセスによって後で追加されます。
iot_permanent_pk_path	IoT プライベートキーファイルパス。例えば、/data/aws/iotmi/iot_pk.pem 。	いいえ	この情報は、ハブプロビジョニング後のオンボーディングプロセスによって後で追加されます。
client_id	MQTT 接続に使用されるクライアント ID。	いいえ	この情報は、ハブのプロビジョニング後のオンボーディングプロセスによって後で追加され、他のコンポーネントが消費します。
event_manager_upper_bound	デフォルト値は 500 です	いいえ	この情報は、ハブのプロビジョニング後のオンボーディングプロセスによって後で追加され、他のコンポーネントが消費します。

ステップ 7: 設定ファイルをハブにコピーする

設定ファイルを `/data/aws/iotmi/config`またはカスタムディレクトリパスにコピーします。このHubOnboardingバイナリへのパスは、オンボーディングプロセス中に指定します。

フリートプロビジョニングの場合

```
/data/
    aws/
        iotmi/
            config/
                iotmi_config.json
            certs/
                claim_cert.pem
                claim_pk.key
```

マネージド統合 Hub SDK をインストールして検証する

次のデプロイ方法を選択して、マネージド統合 Hub SDK をデバイスにインストールします。これは、自動デプロイまたは手動スクリプトインストールAWS IoT Greengrass 用です。このセクションでは、両方のアプローチのセットアップと検証の手順について説明します。

デプロイ方法

- で Hub SDK をインストールする AWS IoT Greengrass
- スクリプトを使用して Hub SDK をデプロイする
- systemd を使用して Hub SDK をデプロイする

で Hub SDK をインストールする AWS IoT Greengrass

AWS IoT Greengrass (Java バージョン) を使用して、デバイスのマネージド統合 Hub SDK コンポーネントをデプロイします。

> **Note**
>
> をセットアップし、理解しておく必要があります AWS IoT Greengrass。詳細については、デAWS IoT Greengrass ベロッパーガイドドキュメントの「とは AWS IoT Greengrass」を参照してください。

AWS IoT Greengrass ユーザーには、次のディレクトリを変更するアクセス許可が必要です。

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

トピック

- コンポーネントをローカルにデプロイする
- クラウドデプロイ
- ハブのプロビジョニングを検証する
- CDMB オペレーションの確認
- LPW-Provisioner オペレーションを検証する

コンポーネントをローカルにデプロイする

デバイスで CreateDeployment AWS IoT Greengrass API を使用して、Hub SDK コンポーネントをデプロイします。バージョン番号は静的ではなく、その時点で使用しているバージョンによって異なる場合があります。には、com.amazon.IoTManagedIntegrationsDevice.AceCommon=`0.2.0` **`version`**の形式を使用します。

```
/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"
```

クラウドデプロイ

デAWS IoT Greengrass ベロッパーガイドの指示に従って、次の手順を実行します。

1. Amazon S3 にアーティファクトをアップロードします。

2. レシピを更新して、Amazon S3 アーティファクトの場所を含めます。
3. 新しいコンポーネントのデバイスにクラウドデプロイを作成します。

ハブのプロビジョニングを検証する

設定ファイルを確認して、プロビジョニングが成功したことを確認します。`/data/aws/iotmi/config/iotmi_config.json` ファイルを開き、状態が に設定されていることを確認しますPROVISIONED。

CDMB オペレーションの確認

CDMB 起動メッセージと正常な初期化については、ログファイルを確認してください。`######`の場所は、 AWS IoT Greengrass がインストールされている場所によって異なります。

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

例

```
[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
```

LPW-Provisioner オペレーションを検証する

ログファイルで LPW-Provisioner 起動メッセージと正常な初期化を確認します。`######`の場所は、AWS IoT Greengrass がインストールされている場所によって異なります。

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

例

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

スクリプトを使用して Hub SDK をデプロイする

インストールスクリプトを使用してマネージド統合 Hub SDK コンポーネントを手動でデプロイし、デプロイを検証します。このセクションでは、スクリプトの実行ステップと検証プロセスについて説明します。

トピック

- 環境を準備する
- Hub SDK スクリプトを実行する
- ハブのプロビジョニングを検証する
- エージェントオペレーションの確認
- LPW-Provisioner オペレーションを検証する

環境を準備する

SDK インストールスクリプトを実行する前に、以下の手順を実行します。

1. フォルダmiddleware内に という名前のartifactsフォルダを作成します。
2. ハブミドルウェアファイルを middlewareフォルダにコピーします。
3. SDK を開始する前に、初期化コマンドを実行します。

> **Important**
> ハブを再起動するたびに、初期化コマンドを繰り返します。

```
#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
```

Hub SDK スクリプトを実行する

アーティファクトディレクトリに移動し、`start_iotmi_sdk.sh`スクリプトを実行します。このスクリプトは、ハブ SDK コンポーネントを正しい順序で起動します。次のログ例を確認して、正常に起動されたことを確認します。

> **Note**
>
> 実行中のすべてのコンポーネントのログは、 `artifacts/logs`フォルダ内にあります。

```
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 ./DeviceCdmb
Process 'DeviceCdmb' 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----
```

ハブのプロビジョニングを検証する

の `iot_provisioning_state`フィールド`/data/aws/iotmi/config/iotmi_config.json`がに設定されていることを確認しますPROVISIONED。

エージェントオペレーションの確認

ログファイルで、エージェントの起動メッセージと正常な初期化を確認します。

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

例

```
[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**
>
> デバイスステートマネージャーデータベース
> `prov.db` データベースが `artifacts` ディレクトリに存在することを確認します。

LPW-Provisioner オペレーションを検証する

ログファイルでLPW-Provisioner起動メッセージと正常な初期化を確認します。

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

コードの例を以下に示します。

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

systemd を使用して Hub SDK をデプロイする

> **Important**
>
> 最新の更新についてはreadme.md、 release.tgz ファイルの hubSystemdSetup ディレクトリの に従います。

このセクションでは、Linux ベースのハブデバイスにサービスをデプロイおよび設定するためのスクリプトとプロセスについて説明します。

概要

デプロイプロセスは、次の 2 つの主要なスクリプトで構成されます。

- copy_to_hub.sh: ホストマシンで を実行して、必要なファイルをハブにコピーします
- setup_hub.sh: ハブで を実行して環境を設定し、サービスをデプロイします

さらに、 はプロセスのブートストラップシーケンスとプロセスのアクセス許可管理systemd/deploy_iotshd_services_on_hub.shを処理し、 によって自動的にトリガーされますsetup_hub.sh。

前提条件

デプロイを成功させるには、リストされている前提条件が必要です。

- systemd サービスがハブで利用可能に

- ハブデバイスへの SSH アクセス
- ハブデバイスの Sudo 権限
- scp ホストマシンにインストールされた ユーティリティ
- sed ホストマシンにインストールされた ユーティリティ
- ホストマシンにインストールされている unzip ユーティリティ

ファイル構造

ファイル構造は、さまざまなコンポーネントの整理と管理を容易にし、コンテンツの効率的なアクセスとナビゲーションを可能にするように設計されています。

```
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
```

SDK リリース tgz ファイルでは、全体的なファイル構造は次のとおりです。

```
IoTSmartHomeDevice-SDK-v0.4.0-linux-aarch64-20240106.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
```

の初期セットアップ

SDK パッケージを抽出する

```
tar -xzf IoTSmartHomeDevice-SDK-vVersion-linux-aarch64-timestamp.tgz
```

抽出されたディレクトリに移動し、パッケージを準備します。

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

デバイス設定ファイルを追加する

リストされている 2 つのステップに従ってデバイス設定ファイルを作成し、ハブにコピーします。

1. デバイス設定ファイルを追加して、必要なデバイス設定ファイルを作成します。SDK は、このファイルをその関数に使用します。
2. 設定ファイルをコピーして、作成した設定ファイルをハブにコピーします。

ハブにファイルをコピーする

ホストマシンからデプロイスクリプトを実行します。

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

Example 例

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

これにより、以下がコピーされます。

- パッケージファイル (ハブの package.zip に名前変更)
- 設定ファイル
- 証明書
- Systemd サービスファイル

ハブのセットアップ

ファイルがコピーされたら、ハブに SSH 接続し、セットアップスクリプトを実行します。

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

ユーザーとグループの設定

デフォルトでは、SDK コンポーネントにユーザーハブとグループハブを使用します。設定には複数の方法があります。

- カスタムユーザー/グループを使用する：

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

- セットアップスクリプトを実行する前に、手動で作成します。

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

- でコマンドを追加しますsetup_hub.sh。

のサービスを管理する

すべてのサービスを再起動するには、ハブから次のスクリプトを実行します。

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

セットアップスクリプトは、必要なディレクトリを作成し、適切なアクセス許可を設定し、サービスを自動的にデプロイします。SSH/SCP を使用していない場合は、特定のデプロイ方法copy_to_hub.shに合わせて を変更する必要があります。デプロイする前に、すべての証明書ファイルと設定が正しく設定されていることを確認します。

セキュアストレージ用のカスタム証明書ハンドラーを作成する

マネージド統合ハブをオンボーディングするときは、デバイス証明書の管理が不可欠です。証明書はデフォルトでファイルシステムに保存されていますが、セキュリティを強化し、柔軟な認証情報管理を行うためのカスタム証明書ハンドラーを作成できます。

マネージド統合 エンドデバイス SDK は、共有オブジェクト (.so) ライブラリとして実装できるストレージインターフェイスを保護するための証明書ハンドラーを提供します。安全なストレージ実装を構築して証明書を読み書きし、実行時にライブラリファイルを HubOnboarding プロセスにリンクします。

API 定義とコンポーネント

次の`secure_storage_cert_handler_interface.hpp`ファイルを確認して、実装の API コンポーネントと要件を理解します。

トピック

- API 定義
- 主要コンポーネント

API 定義

`secure_storage_cert_hander_interface.hpp` の内容

```
/*
    * 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
```

主要コンポーネント

- CERT_TYPE_T - ハブ上のさまざまなタイプの証明書。
  - CLAIM - クレーム証明書は、元々ハブにあり、永続的な証明書と交換されます。
  - DHA - 現時点では未使用です。
  - PERMANENT - マネージド統合エンドポイントに接続するための永続的な証明書。
- read_cert_and_private_key - (FUNCTION TO BE IMPLEMENTED) の証明書とキーの値をリファレンス入力に読み取ります。この関数は、CLAIM 証明書と PERMANENT 証明書の両方を読み取ることができ、上記の証明書タイプによって区別される必要があります。
- write_permanent_cert_and_private_key - (FUNCTION TO BE IMPLEMENTED) は、永続的な証明書とキーの値を目的の場所に書き込みます。

ビルド例

内部実装ヘッダーをパブリックインターフェイス (secure_storage_cert_handler_interface.hpp) から分離して、クリーンなプロジェクト構造を維持します。この分離により、証明書ハンドラーを構築しながら、パブリックコンポーネントとプライベートコンポーネントを管理できます。

> **Note**
>
> パブリックsecure_storage_cert_handler_interface.hppとして宣言します。

トピック

- プロジェクト構造
- インターフェイスを継承する
- 実装
- CMakeList.txt

プロジェクト構造

インターフェイスを継承する

インターフェイスを継承する具体的なクラスを作成します。このヘッダーファイルやその他のファイルを別のディレクトリに非表示にして、プライベートヘッダーとパブリックヘッダーをビルド時に簡単に区別できるようにします。

```
#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
```

実装

上記で定義したストレージクラス を実装します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;
 }
```

インターフェイス で定義されたファクトリ関数を実装します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_hander_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
     )
```

使用方法

コンパイル後、libSecureStorageCertHandler.so共有オブジェクトライブラリファイルとそれに関連するシンボリックリンクが作成されます。ライブラリファイルとシンボリックリンクの両方を、HubOnboarding バイナリで想定されるライブラリの場所にコピーします。

トピック

- 主な考慮事項
- 安全なストレージを使用する

主な考慮事項

- ユーザーアカウントに HubOnboarding バイナリとlibSecureStorageCertHandler.soライブラリの両方に対する読み取りおよび書き込みアクセス許可があることを確認します。
- を唯一のパブリックヘッダーファイルsecure_storage_cert_handler_interface.hppとして保持します。他のすべてのヘッダーファイルは、プライベート実装に残ります。
- 共有オブジェクトライブラリ名を確認します。の構築中にlibSecureStorageCertHandler.so、HubOnboarding で などのファイル名に特定のバージョンが必要になる場合がありますlibSecureStorageCertHandler.so.1.0。ldd コマンドを使用してライブラリの依存関係を確認し、必要に応じてシンボリックリンクを作成します。
- 共有ライブラリの実装に外部依存関係がある場合は、HubOnboarding がアクセスできる ディレクトリに/usr/lib or the iotmi_common保存します。

安全なストレージを使用する

iot_claim_cert_path と の両方iot_claim_pk_pathを に設定して、 iotmi_config.jsonファイルを更新します**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"
  }
}
```

プロセス間通信 (IPC) APIs

マネージド統合ハブの外部コンポーネントは、エージェントコンポーネントとプロセス間通信 (IPC) を使用して、マネージド統合エンドデバイス SDK と通信できます。ハブの外部コンポーネントの例は、ローカルルーチンを管理するデーモン (バックグラウンドプロセスを継続的に実行) です。通信中、IPC クライアントはコマンドやその他のリクエストを発行し、イベントをサブスクライブする外部コンポーネントです。IPC サーバーは、 マネージド統合エンドデバイス SDK の エージェントコンポーネントです。詳細については、「IPC クライアントのセットアップ」を参照してください。

IPC クライアントを構築するために、IPC クライアントライブラリ`IotmiLocalControllerClient`が用意されています。このライブラリは、コマンドリクエストの送信、デバイス状態のクエリ、イベント (デバイス状態イベントなど) のサブスクライブ、イベントベースのインタラクションの処理など、 エージェントで IPC サーバーと通信するためのクライアント側の APIs を提供します。

トピック

- IPC クライアントのセットアップ
- IPC インターフェイスの定義とペイロード

IPC クライアントのセットアップ

`IotmiLocalControllerClient` ライブラリは、基本的な IPC APIs のラッパーであり、アプリケーションに IPC を実装するプロセスを簡素化および合理化します。以下のセクションでは、 が提供する APIsについて説明します。

> **Note**
>
> このトピックは、IPC サーバーの実装ではなく、IPC クライアントとしての外部コンポーネント専用です。

1. IPC クライアントを作成する

   リクエストの処理に使用する前に、まず IPC クライアントを初期化する必要があります。`IotmiLocalControllerClient` ライブラリでコンストラクタを使用できます。これにより、サブスクライバーコンテキストをパラメータ `char *subscriberCtx`として受け取り、それに基づいて IPC クライアントマネージャーが作成されます。IPC クライアントを作成する例を次に示します。

```
// Context for subscriber
char subscriberCtx[] = "example_ctx";

// Instantiate the client
IotmiLocalControllerClient lcc(subscriberCtx);
```

2. イベントをサブスクライブする

   IPC クライアントをターゲット IPC サーバーのイベントにサブスクライブできます。IPC サーバーがクライアントがサブスクライブしているイベントを発行すると、クライアントはそのイベントを受け取ります。サブスクライブするには、 `registerSubscriber`関数を使用して、サブスクライブするイベント IDs とカスタマイズされたコールバックを指定します。

   以下は、 `registerSubscriber`関数の定義とその使用例です。

```
iotmi_statusCode_t registerSubscriber(
        std::vector<iotmiIpc_eventId_t> eventIds,
        SubscribeCallbackFunction cb);
```

```
// A basic example of customized subscribe callback, which prints the event ID,
 data, and length received
void customizedSubscribeCallback(iotmiIpc_eventId_t event_id, uint32_t length,
 const uint8_t *data, void *ctx) {
    IOTMI_IPC_LOGI("Received subscribed event id: %d\n"
                   "length: %d\n"
                   "data: %s\n",
                   event_id, length, data);
}

iotmi_statusCode_t status;
status = lcc.registerSubscriber({IOTMI_IPC_EVENT_DEVICE_UPDATE_TO_RE},
 customerProvidedSubscribeCallback);
```

status は、オペレーション (サブスクライブやリクエストの送信など) が成功したかどうかを確認するように定義されています。オペレーションが成功した場合、返されるステータスは ですIOTMI_STATUS_OK (= 0)。

> **Note**
>
> IPC ライブラリには、サブスクリプション内のサブスクライバーとイベントの最大数に対して次のサービスクォータがあります。
>
> - プロセスあたりのサブスクライバーの最大数: 5
>
> IPC ライブラリIOTMI_IPC_MAX_SUBSCRIBERで として定義されます。
> - 定義されたイベントの最大数: 32
>
> IPC ライブラリIOTMI_IPC_EVENT_PUBLIC_ENDで として定義されます。
> - 各サブスクライバーには 32 ビットのイベントフィールドがあり、各ビットは定義されたイベントに対応します。

3. IPC クライアントをサーバーに接続する

   IotmiLocalControllerClient ライブラリの接続関数は、IPC クライアントの初期化、サブスクライバーの登録、registerSubscriber関数で提供されたイベントのサブスクライブなどのジョブを実行します。IPC クライアントで接続関数を呼び出すことができます。

```
status = lcc.connect();
```

   リクエストを送信したり、他のオペレーションを行ったりIOTMI_STATUS_OKする前に、返されたステータスが であることを確認します。

4. コマンドリクエストとデバイス状態のクエリを送信する

   エージェント内の IPC サーバーは、コマンドリクエストとデバイス状態リクエストを処理できます。

   - コマンドリクエスト

     コマンドリクエストペイロード文字列を形成し、 sendCommandRequest関数を呼び出して送信します。例 :

```
status = lcc.sendCommandRequest(payloadData, iotmiIpcMgr_commandRequestCb,
 nullptr);
```

```
/**
* @brief method to send local control command
* @param payloadString A pre-defined data format for local command request.
* @param callback a callback function with typedef as PublishCallbackFunction
* @param client_ctx client provided context
* @return
*/
iotmi_statusCode_t sendCommandRequest(std::string payloadString,
 PublishCallbackFunction callback, void *client_ctx);
```

コマンドリクエスト形式の詳細については、「コマンドリクエスト」を参照してください。

Example コールバック関数

IPC サーバーはまず、IPC クライアントCommand received, will send command response backにメッセージ確認を送信します。この確認を受け取ると、IPC クライアントはコマンドレスポンスイベントを期待できます。

```
void iotmiIpcMgr_commandRequestCb(iotmi_statusCode_t ret_status,
                                  void *ret_data, void *ret_client_ctx) {
    char* data = NULL;
    char *ctx = NULL;

    if (ret_status != IOTMI_STATUS_OK)
        return;

    if (ret_data == NULL) {
        IOTMI_IPC_LOGE("error, event data NULL");
        return;
    }

    if (ret_client_ctx == NULL) {
        IOTMI_IPC_LOGE("error, event client ctx NULL");
        return;
    }
```

```
    data = (char *)ret_data;
    ctx = (char *)ret_client_ctx;
    IOTMI_IPC_LOGI("response received: %s \n", data);
}
```

- デバイス状態リクエスト

  sendCommandRequest 関数と同様に、このsendDeviceStateQuery関数はペイロード文字列、対応するコールバック、およびクライアントコンテキストも受け取ります。

```
status = lcc.sendDeviceStateQuery(payloadData, iotmiIpcMgr_deviceStateQueryCb,
 nullptr);
```

IPC インターフェイスの定義とペイロード

このセクションでは、エージェントと外部コンポーネント間の通信専用の IPC インターフェイスに焦点を当て、これら 2 つのコンポーネント間の IPC APIs の実装例を示します。次の例では、外部コンポーネントがローカルルーチンを管理します。

IoTManagedIntegrationsDevice-IPC ライブラリでは、エージェントと外部コンポーネント間の通信用に次のコマンドとイベントが定義されています。

```
typedef enum {
    // The async cmd used to send commands from the external component to Agent
    IOTMI_IPC_SVC_SEND_REQ_FROM_RE = 32,
    // The async cmd used to send device state query from the external component to
 Agent
    IOTMI_IPC_SVC_SEND_QUERY_FROM_RE = 33,
    // ...
} iotmiIpcSvc_cmd_t;
```

```
typedef enum {
    // Event about device state update from Agent to the component
    IOTMI_IPC_EVENT_DEVICE_UPDATE_TO_RE = 3,
    // ...
} iotmiIpc_eventId_t;
```

コマンドリクエスト

コマンドリクエスト形式

- Example コマンドリクエスト

```
{
   "payload": {
        "traceId": "LIGHT_DIMMING_UPDATE",
        "nodeId": "1",
        "managedThingId": <ManagedThingId of the device>,
        "endpoints": [{
            "id": "1",
            "capabilities": [
                {
                    "id": "matter.LevelControl@1.4",
                    "name": "Level Control",
                    "version": "1.0",
                    "actions":[
                        {
                            "name": "UpdateState",
                            "parameters": {
                                "OnLevel": 5,
                                "DefaultMoveRate": 30
                            }
                        }
                    ]
                }
            ]
        }]
    }
}
```

コマンドレスポンス形式

- 外部コンポーネントからのコマンドリクエストが有効な場合、エージェントは CDMB (共通データモデルブリッジ) に送信します。コマンドの実行時間やその他の情報を含む実際のコマンドレスポンスは、コマンドの処理に時間がかかるため、外部コンポーネントにすぐには送信されません。このコマンドレスポンスは、 エージェントからの即時レスポンスです (確認応答など) 。レスポンスは、マネージド統合が コマンドを受信した外部コンポーネントに指示し、有効なローカルトークンがない場合は処理するか破棄します。コマンドレスポンスは文字列形式で送信されます。

```
std::string errorResponse = "No valid token for local command, cannot process.";
    *ret_buf_len = static_cast<uint32_t>(errorResponse.size());
    *ret_buf = new uint8_t[*ret_buf_len];
    std::memcpy(*ret_buf, errorResponse.data(), *ret_buf_len);
```

デバイス状態リクエスト

外部コンポーネントは、デバイス状態リクエストを エージェントに送信します。リクエストはmanagedThingIdデバイスの を提供し、エージェントはそのデバイスの状態を返信します。

デバイス状態のリクエスト形式

- デバイス状態リクエストには、クエリされたデバイスの managedThingId が必要です。

```
{
    "payload": {
        "managedThingId": "testManagedThingId"
    }
}
```

デバイス状態のレスポンス形式

- デバイス状態リクエストが有効な場合、エージェントは状態を文字列形式で返します。

  Example 有効なリクエストのデバイス状態レスポンス

  ```
  {
      "payload": {
          "currentState": "exampleState"
      }
  }
  ```

  デバイス状態リクエストが有効でない場合 (有効なトークンがない場合、ペイロードを処理できない場合、または別のエラーケースがある場合など) 、エージェントはレスポンスを返します。レスポンスには、エラーコードとエラーメッセージが含まれます。

Example 無効なリクエストに対するデバイス状態レスポンス

```
{
    "payload": {
        "response":{
            "code": 111,
            "message": "errorMessage"
        }
    }
}
```

コマンドレスポンス

Example コマンドレスポンス形式

```
{
   "payload": {
        "traceId": "LIGHT_DIMMING_UPDATE",
        "commandReceivedAt": "1684911358.533",
        "commandExecutedAt": "1684911360.123",
        "managedThingId": <ManagedThingId of the device>,
        "nodeId": "1",
        "endpoints": [{
            "id": "1",
            "capabilities": [
                {
                    "id": "matter.OnOff@1.4",
                    "name": "On/Off",
                    "version": "1.0",
                    "actions":[
                        {}
                    ]
                }
            ]
        }]
    }
}
```

通知イベント

Example 通知イベント形式

```
{
   "payload": {
        "hasState": "true"
        "nodeId": "1",
        "managedThingId": <ManagedThingId of the device>,
        "endpoints": [{
            "id": "1",
            "capabilities": [
                {
                    "id": "matter.OnOff@1.4",
                    "name": "On/Off",
                    "version": "1.0",
                    "properties":[
                        {
                            "name": "OnOff",
                            "value": true
                        }
                    ]
                }
            ]
        }]
    }
}
```

ハブコントロールの設定

ハブコントロールは、 マネージド統合の拡張です。エンドデバイス SDK は、Hub SDK の `MQTTProxy`コンポーネントとインターフェイスできます。ハブ制御を使用すると、エンドデバイス SDK を使用してコードを実装し、マネージド統合クラウドを介してハブを別のデバイスとして制御できます。ハブコントロール SDK は、 というラベルが付いた Hub SDK 内の別のパッケージとして提供されます`hub-control-x.x.x`。

トピック

- 前提条件
- エンドデバイス SDK コンポーネント
- エンドデバイス SDK との統合

- 例: ハブコントロールを構築する
- サポートされている例
- サポートされているプラットフォーム

前提条件

ハブコントロールを設定するには、まず以下が必要です。

- End device SDK、バージョン 0.4.0 以降にオンボードされたハブ。
- ハブ、バージョン 0.5.0 以降で実行されている MQTT プロキシコンポーネント。

エンドデバイス SDK コンポーネント

エンドデバイス SDK から次のコンポーネントを使用します。

- データモデルのコードジェネレーター
- データモデルハンドラー

Hub SDK にはすでにオンボーディングプロセスとクラウドへの接続があるため、以下のコンポーネントは必要ありません。

- プロビジョンド
- PKCS インターフェイス
- ジョブハンドラー
- MQTT エージェント

エンドデバイス SDK との統合

1. データモデルのコードジェネレーターの指示に従って、低レベル C コードを生成します。
2. 「エンドデバイス SDK の統合」の手順に従って、次の操作を行います。

   a. ビルド環境をセットアップします。

      開発ホストとして Amazon Linux 2023/x86_64 でコードを構築します。必要なビルドの依存関係をインストールします。

```
dnf install make gcc gcc-c++ cmake
```

b. ハードウェアコールバック関数の開発

ハードウェアコールバック関数を実装する前に、API の仕組みを理解してください。この例では、On/Off クラスターと OnOff 属性を使用してデバイス関数を制御します。API の詳細については、「」を参照してください低レベル C 関数の API オペレーション。

```
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. エンドポイントを設定し、ハードウェアコールバック関数をフックする

関数を実装したら、エンドポイントを作成し、コールバックを登録します。以下のタスクを完了します。

i. デバイスエージェントを作成する

ii. サポートするクラスター構造ごとにコールバック関数ポイントを埋める

iii. エンドポイントを設定し、サポートされているクラスターを登録する

```
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);
}
```

例: ハブコントロールを構築する

Hub コントロールは Hub SDK パッケージの一部として提供されます。ハブコントロールサブパッケージには というラベルが付けられ`hub-control-x.x.x`、変更されていないデバイス SDK とは異なるライブラリが含まれています。

1. コード生成ファイルを `example`フォルダに移動します。

```
cp codegen/out/* example/dm
```

2. ハブコントロールを構築するには、次のコマンドを実行します。

```
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. ハブの MQTTProxyコンポーネントで例を実行し、 HubOnboardingおよび MQTTProxyコンポーネントを実行します。

```
./examples/iotmi_device_sample_camera/iotmi_device_sample_camera
```

サポートされている例

以下の例が構築され、テストされています。

- iotmi_device_dm_air_purifier_demo
- iotmi_device_basic_diagnostics
- iotmi_device_dm_camera_demo

サポートされているプラットフォーム

次の表に、ハブ制御でサポートされているプラットフォームを示します。

アーキテクチャ	オペレーティングシステム	GCC バージョン	Binutils バージョン
X86_64	リナックス	10.5.0	2.37
AARCH64	リナックス	10.5.0	2.37

サポートされている Zigbee および Z-Wave デバイスタイプ

このページには、マネージド統合でテストされ、サポートされているハブ接続デバイスタイプが一覧表示されます。マネージド統合は、これらのデバイスユーザーガイドによるセットアップ (UGS)に対して 簡易セットアップ (SS)と の両方をサポートします。

この表は、サポートされている Zigbee デバイスの一覧です。

Zigbee デバイスタイプ	サポートされる機能
スマート電球/調光可能ライト/RGB ライト	OnOff、LevelControl、ColorControl
スマートプラグ	OnOff
スマートスイッチ	OnOff
LED ストリップ	OnOff、LevelControl、ColorControl
水バルブ	OnOff
ドアロック	DoorLock、アラーム、PowerSourceConfiguration、タイマー
ラジエーターバルブ	サーモスタット、OnOff、タイマー
サーモスタット	サーモスタット、FanControl、OnOff、タイマー
ガレージドアオープナー	WindowCovering、OnOff、LevelControl
煙アラーム	BooleanState、OnOff、TemperatureMeasurement、タイマー、SmokeCOAlarm
モーションセンサー	BooleanState
占有率/ヒューマンプレゼンスセンサー	BooleanState、OccupancySensing
ドアおよびウィンドウセンサー	BooleanState
水漏れセンサー	BooleanState
振動センサー	BooleanState
温度および湿度センサー	TemperatureMeasurement、RelativeHumidityMeasurement

この表は、サポートされている Z-Wave デバイスの一覧です。

Z-Wave デバイスタイプ	サポートされる機能
スマート電球/調光可能ライト	OnOff、LevelControl
スマートプラグ	OnOff
ドアロック	DoorLock
ガレージドアコントローラー	OnOff、LevelControl
エネルギーメーター	ElectricalEnergyMeasurementElectricalPowerMeasurement
バッテリー	LevelControl
サイレン	LevelControl
モーションセンサー	BooleanState
ドアおよびウィンドウセンサー	BooleanState
水漏れセンサー	BooleanState
温度センサー	TemperatureMeasurement
CO センサー	SmokeCOAlarm
煙センサー	SmokeCOAlarm

CloudWatch Logs を有効にする

Hub SDK は包括的なログ記録機能を提供します。デフォルトでは、Hub SDK はローカルファイルシステムにログを書き込みます。ただし、クラウド API を使用して CloudWatch Logs へのログストリーミングを設定できます。CloudWatch Logs は以下を提供します。

- デバイスのパフォーマンスのモニタリング: プロアクティブなデバイス管理のために詳細なランタイムログをキャプチャします。デバイスフリート全体で高度なログ分析とモニタリングを有効にする

- 問題のトラブルシューティング: 迅速な診断分析のための詳細なログエントリを生成します。システムとアプリケーションレベルのイベントを記録して、詳細な調査を行います。
- 柔軟で一元化されたログ記録: デバイスに直接アクセスしないリモートログ管理。複数のデバイスからのログを単一の検索可能なリポジトリに集約します。

前提条件

- マネージドデバイスをクラウドにオンボードします。詳細については、「Hub オンボーディングの設定」を参照してください。
- Hub エージェントの起動と正常な初期化を検証します。詳細については、「マネージド統合 Hub SDK をインストールして検証する」を参照してください。

> **Note**
> ログ記録設定を作成するには、PutRuntimeLogConfiguration API を参照してください。

> **Warning**
> ログを有効にすると、階層化されたクォータの計測にカウントされます。ログレベルを増やすと、メッセージ量が増え、追加コストが発生します。

Setup Hub SDK ログ設定

API を呼び出してランタイムログ設定を設定することで、ハブ SDK ログ設定を構成します。

Example サンプル API リクエスト

```
curl \
    --request PUT \
    --location "api.iotmanagedintegrations.$AWS_REGION.api.aws/runtime-log-
configurations/$ManagedThingId" \
    --aws-sigv4 "aws:amz:$AWS_REGION:iotmanagedintegrations" \
    --user "$AWS_ACCESS_KEY_ID:$AWS_SECRET_ACCESS_KEY" \
    --header "x-amz-security-token: $AWS_SESSION_TOKEN" \
    --header "Content-Type: application/json" \
    --data '{ "RuntimeLogConfigurations": { "UploadLog": true, "LogLevel": "DEBUG" } }'
```

RuntimeLogConfigurations 属性

次の属性はオプションであり、 `RuntimeLogConfigurations` API で設定できます。

LogLevel

ランタイムトレースの最小重要度レベルを設定します。値: `DEBUG`, `ERROR`, `INFO`, `WARN`

デフォルト: `WARN` (リリースされたビルド)

LogFlushLevel

ローカルストレージへの即時データフラッシュの重要度レベルを決定します。値: `DEBUG`, `ERROR`, `INFO`, `WARN`

デフォルト: `DISABLED`

LocalStoreLocation

ランタイムトレースのストレージの場所を指定します。デフォルト: `/var/log/awsiotmi`

- アクティブなログ: `/var/log/awsiotmi/ManagedIntegrationsDeviceSdkHub.log`
- ローテーションされたログ: `/var/log/awsiotmi/ManagedIntegrationsDeviceSdkHub.N.log` (N はローテーションの順序を示します)

LocalStoreFileRotationMaxBytes

現在のファイルが指定されたサイズを超えると、ファイルのローテーションをトリガーします。

> **Important**
> 最適な効率を得るには、ファイルサイズを 125 KB 未満に保ちます。125 KB を超える値は自動的に制限されます。

LocalStoreFileRotationMaxFiles、

ログデーモンで許可されるローテーションファイルの最大数を設定します。

UploadLog

クラウドへのランタイムトレース転送を制御します。ログは `/aws/iotmanagedintegration` CloudWatch Logs グループに保存されます。

デフォルト: `false`。

UploadPeriodMinutes

ランタイムトレースのアップロード頻度を定義します。デフォルト: 5

DeleteLocalStoreAfterUpload

アップロード後のファイルの削除を制御します。デフォルト: `true`

> **Note**
>
> false に設定すると、アップロードされたファイルの名前は次のように変更されます。 `/var/log/awsiotmi/ManagedIntegrationsDeviceSdkHub.uploaded.{uploaded_timestamp}`

ログファイルの例

以下の CloudWatch Logs ファイルの例を参照してください。

マネージド統合 エンドデバイス SDK

スマートデバイスをマネージド統合に接続し、統合制御インターフェイスを介してコマンドを処理する IoT プラットフォームを構築します。End device SDK はデバイスファームウェアと統合され、SDK エッジコンポーネントによるシンプルなセットアップと、 AWS IoT Core および AWS IoT Device Management への安全な接続を提供します。

このガイドでは、ファームウェアに End Device SDK を実装する方法について説明します。アーキテクチャ、コンポーネント、統合のステップを確認して、実装の構築を開始します。

トピック

- End Device SDK とは
- エンドデバイス SDK アーキテクチャとコンポーネント
- プロビジョニング先
- ジョブハンドラー
- データモデルコードジェネレーター
- 低レベル C 関数の API オペレーション
- マネージド統合での機能とデバイスのインタラクション
- エンドデバイス SDK の統合
- End Device SDK をデバイスに移植する
- 付録

End Device SDK とは

End Device SDK とは

End device SDK は、 が提供するソースコード、ライブラリ、ツールのコレクションです AWS IoT。リソースに制約のある環境向けに構築された SDK は、組み込み Linux およびリアルタイムオペレーティングシステム (RTOS) で実行されているカメラやエアピュリファイアなど、わずか 512 KB の RAM と 4 MB のフラッシュメモリを持つデバイスをサポートします。 AWS IoT Device Management のマネージド統合はパブリックプレビューです。AWS IoT マネジメントコンソールから End device SDK の最新バージョンをダウンロードします。

コアコンポーネント

SDK は、クラウド通信用の MQTT エージェント、タスク管理用のジョブハンドラー、マネージド統合の Data Model Handler を組み合わせます。これらのコンポーネントは連携して、デバイスとマネージド統合間の安全な接続と自動データ変換を提供します。

技術的な要件の詳細については、「」を参照してください付録。

エンドデバイス SDK アーキテクチャとコンポーネント

このセクションでは、エンドデバイス SDK アーキテクチャと、そのコンポーネントが低レベルの C 関数とどのように相互作用するかについて説明します。次の図は、SDK フレームワークのコアコンポーネントとその関係を示しています。

エンドデバイス SDK コンポーネント

End Device SDK アーキテクチャには、マネージド統合機能統合用の以下のコンポーネントが含まれています。

プロビジョニング先

安全な MQTT 通信のためのデバイス証明書やプライベートキーなど、マネージド統合クラウドにデバイスリソースを作成します。これらの認証情報は、デバイスと マネージド統合間の信頼された接続を確立します。

MQTT エージェント

スレッドセーフな C クライアントライブラリを介して MQTT 接続を管理します。このバックグラウンドプロセスは、マルチスレッド環境でコマンドキューを処理し、メモリに制約のあるデバイス用にキューサイズを設定できます。メッセージは、処理のためにマネージド統合を介してルーティングされます。

ジョブハンドラー

デバイスファームウェア、セキュリティパッチ、およびファイル配信のover-the-air (OTA) 更新を処理します。この組み込みサービスは、すべての登録済みデバイスのソフトウェア更新を管理します。

データモデルハンドラー

Matter データモデルの AWS実装を使用して、マネージド統合と低レベル C 機能間のオペレーションを変換します。詳細については、GitHub の Matter ドキュメントを参照してください。

キーと証明書

PKCS #11 API を使用して暗号化オペレーションを管理し、ハードウェアセキュリティモジュールと corePKCS11 などのソフトウェア実装の両方をサポートします。この API は、TLS 接続中にプロビジョニング先や MQTT エージェントなどのコンポーネントの証明書オペレーションを処理します。

プロビジョニング先

プロビジョニング先は、クレームによるフリートプロビジョニングを可能にするマネージド統合のコンポーネントです。プロビジョニング先を使用すると、デバイスを安全にプロビジョニングできます。SDK は、デバイスプロビジョニングに必要なリソースを作成します。これには、マネージド統合クラウドから取得したデバイス証明書とプライベートキーが含まれます。デバイスをプロビジョニングする場合、またはデバイスを再プロビジョニングする必要がある変更がある場合は、プロビジョニング先を使用できます。

トピック

- プロビジョンドワークフロー
- 環境変数を設定する
- カスタムエンドポイントを作成する
- プロビジョニングプロファイルを作成する
- マネージド型モノを作成する
- SDK ユーザーの Wi-Fi プロビジョニング
- クレームによるフリートのプロビジョニング
- モノのマネージド機能

プロビジョンドワークフロー

このプロセスでは、クラウド側とデバイス側の両方でセットアップする必要があります。お客様は、カスタムエンドポイント、プロビジョニングプロファイル、マネージド型モノなどのクラウド要件を設定します。最初のデバイスの電源をオンにすると、プロビジョニング者は次の操作を行います。

1. クレーム証明書を使用してマネージド統合エンドポイントに接続します。
2. フリートプロビジョニングフックを使用してデバイスパラメータを検証する
3. 永続的な証明書とプライベートキーを取得してデバイスに保存します。

4. デバイスは永続的な証明書を使用して再接続します
5. デバイス機能を検出してマネージド統合にアップロードします

プロビジョニングが成功すると、デバイスはマネージド統合と直接通信します。プロビジョニング先は、再プロビジョニングタスクに対してのみアクティブ化します。

環境変数を設定する

クラウド環境で次の AWS 認証情報を設定します。

```
$ 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
```

カスタムエンドポイントを作成する

クラウド環境で GetCustomEndpoint API コマンドを使用して、device-to-cloud通信用のカスタムエンドポイントを作成します。

```
aws iotmanagedintegrations get-custom-endpoint
```

レスポンスの例

```
{ "EndpointAddress":"ACCOUNT-SPECIFIC-ENDPOINT.mqtt-api.ACCOUNT-ID.YOUR-AWS-
REGION.iotmanagedintegrations.iot.aws.dev"}
```

プロビジョニングプロファイルを作成する

フリートプロビジョニング方法を定義するプロビジョニングプロファイルを作成します。クラウド環境で CreateProvisioningProfile API を実行して、デバイス認証用のクレーム証明書とプライベートキーを返します。

```
aws iotmanagedintegrations create-provisioning-profile \
--provisioning-type "FLEET_PROVISIONING" \
--name "PROVISIONING-PROFILE-NAME"
```

レスポンスの例

```
{ "Arn":"arn:aws:iotmanagedintegrations:AWS-REGION:YOUR-ACCOUNT-ID:provisioning-
profile/PROFILE_NAME",
    "ClaimCertificate":"string",
    "ClaimCertificatePrivateKey":"string",
    "Name":"ProfileName",
    "ProvisioningType":"FLEET_PROVISIONING"}
```

corePKCS11 プラットフォーム抽象化ライブラリ (PAL) を実装して、corePKCS11 ライブラリをデバイスと連携させることができます。corePKCS11 PAL ポートは、クレーム証明書とプライベートキーを保存する場所を提供する必要があります。この機能を使用すると、デバイスのプライベートキーと証明書を安全に保存できます。プライベートキーと証明書は、ハードウェアセキュリティモジュール (HSM) または信頼されたプラットフォームモジュール (TPM) に保存できます。

マネージド型モノを作成する

CreateManagedThing API を使用して、マネージド統合クラウドにデバイスを登録します。デバイスのシリアル番号 (SN) とユニバーサル製品コード (UPC) を含めます。

```
aws iotmanagedintegrations create-managed-thing –role DEVICE \
  --authentication-material-type WIFI_SETUP_QR_BAR_CODE \
  --authentication-material "SN:DEVICE-SN;UPC:DEVICE-UPC;"
```

以下に、API レスポンスの例を示します。

```
{
   "Arn":"arn:aws:iotmanagedintegrations:AWS-REGION:ACCOUNT-ID:managed-
thing/59d3c90c55c4491192d841879192d33f",
   "CreatedAt":1.730960226491E9,
   "Id":"59d3c90c55c4491192d841879192d33f"
}
```

API は、プロビジョニングの検証に使用できる Managed Thing ID を返します。プロビジョニングトランザクション中に承認されたマネージドモノと一致するデバイスシリアル番号 (SN) とユニバーサル製品コード (UPC) を指定する必要があります。トランザクションは次のような結果を返します。

```
/**
 * @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;
```

SDK ユーザーの Wi-Fi プロビジョニング

デバイスメーカーとサービスプロバイダーには、Wi-Fi 認証情報を受信および設定するための独自の Wi-Fi プロビジョニングサービスがあります。Wi-Fi プロビジョニングサービスでは、専用モバイルアプリ、Bluetooth Low Energy (BLE) 接続、およびその他の独自のプロトコルを使用して、初期セットアッププロセスのために Wi-Fi 認証情報を安全に転送します。

エンドデバイス SDK のコンシューマーは Wi-Fi プロビジョニングサービスを実装する必要があり、デバイスは Wi-Fi ネットワークに接続できます。

クレームによるフリートのプロビジョニング

プロビジョニング先を使用すると、エンドユーザーはクレームによるプロビジョニングを使用して一意の証明書をプロビジョニングし、マネージド統合に登録できます。

クライアント ID は、プロビジョニングテンプレートレスポンスまたはデバイス証明書から取得できます。 `<common name>“_”<serial number>`

モノのマネージド機能

プロビジョニング先はマネージドモノの機能を検出し、その機能をマネージド統合にアップロードします。これにより、アプリケーションやその他のサービスがアクセスできるようになります。デバイス、他のウェブクライアント、およびサービスは、MQTT と予約された MQTT トピック、または REST API を使用した HTTP を使用して機能を更新できます。

ジョブハンドラー

マネージド統合ジョブハンドラーは、フィールドデバイスのover-the-air更新を受信するためのコンポーネントです。ファームウェア更新用のカスタムジョブドキュメントをダウンロードしたり、リモートオペレーションを実行したりできます。OTA 更新は、デバイスのファームウェアの更新、影響を受けるデバイスのセキュリティ問題の修正、またはマネージド統合に登録されたデバイスへのファイルの送信にも使用できます。

ジョブハンドラーの仕組み

ジョブハンドラーを使用する前に、クラウドとデバイス側で以下のセットアップ手順が必要です。

- デバイス側では、デバイス製造元はover-the-air (OTA) 更新用のファームウェア更新メソッドを準備します。
- クラウド側では、お客様はリモートオペレーションとジョブの作成を説明するカスタム定義のジョブドキュメントを準備します。

このプロセスでは、クラウド側とデバイス側の両方でセットアップする必要があります。デバイスメーカーは、顧客がジョブドキュメントを準備し、更新タスクを作成する間に、ファームウェア更新メソッドを実装します。デバイスが接続する場合：

1. デバイスが保留中のジョブリストを取得する
2. ジョブハンドラーは、リストに 1 つ以上のジョブ実行があるかどうかを確認し、1 つを選択します。
3. ジョブハンドラーは、ジョブドキュメントで指定されたアクションを実行します。
4. ジョブハンドラーはジョブの実行をモニタリングし、ジョブのステータスを `SUCCESS`または で更新します。 `FAILED`

ジョブハンドラーの実装

デバイスでover-the-air (OTA) 更新を処理するためのキーオペレーションを実装します。ジョブドキュメントの Amazon S3 アクセスをセットアップし、API を使用して OTA タスクを作成し、デバイスでジョブドキュメントを処理し、OTA エージェントを統合します。次の図のステップは、エンドデバイス SDK と 機能間のやり取りによってover-the-air (OTA) 更新リクエストがどのように処理されるかを示しています。

ジョブハンドラーは、次のキーオペレーションを通じて OTA 更新を処理します。

オペレーション

- 更新をアップロードして開始する
- Amazon S3 アクセスを設定する
- ジョブドキュメントを処理する
- OTA エージェントを実装する

更新をアップロードして開始する

カスタムジョブドキュメント (JSON 形式) を Amazon S3 バケットにアップロードし、CreateOtaTask API を使用して OTA タスクを作成します。以下のパラメータを含めます。

- `S3Url`: ジョブドキュメントの URL の場所
- `Target`: マネージド型のモノの ARN 配列 (最大 100 デバイス)

リクエストの例

```
aws iotmanagedintegrations create-ota-task
          --description JOB-DESCRIPTION \
--s3-url "s3:/amzn-s3-demo-bucket/your-file.txt \
--protocol HTTP \
--target  "arn:aws:iotmanagedintegrations:AWS-REGION:ACCOUNT-ID:managed-thing/
${MANAGED-THING-ID}"  \
--ota-mechanism PUSH --ota-type ONE_TIME \
--client-token foo
```

Amazon S3 アクセスを設定する

マネージド統合にジョブドキュメントへのアクセスを許可する Amazon S3 バケットポリシーを追加します。

```
{
    "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"
            ]
        }
    ]
```

```
}
```

ジョブドキュメントを処理する

OTA タスクを作成すると、ジョブハンドラーはデバイスで次の手順を実行します。更新が利用可能になると、MQTT 経由でジョブドキュメントをリクエストします。

1. MQTT 通知トピックをサブスクライブします
2. 保留中のジョブの `StartNextPendingJobExecution` API を呼び出します
3. 利用可能なジョブドキュメントを受け取ります。
4. 指定したタイムアウトに基づいて更新を処理します

ジョブハンドラーを使用すると、アプリケーションはすぐにアクションを実行するか、指定されたタイムアウト期間まで待機するかを判断できます。

OTA エージェントを実装する

マネージド統合からジョブドキュメントを受け取るときは、ジョブドキュメントを処理し、更新をダウンロードし、インストールオペレーションを実行する独自の OTA エージェントを実装している必要があります。OTA エージェントは次のステップを実行する必要があります。

1. ファームウェア Amazon S3 URL URLs
2. HTTP を使用してファームウェアの更新をダウンロードする
3. デジタル署名を検証する
4. 検証済み更新をインストールする
5. `SUCCESS` または `FAILED`ステータス`iotmi_JobsHandler_updateJobStatus`で を呼び出す

デバイスが OTA オペレーションを正常に完了したら、 ステータスの `iotmi_JobsHandler_updateJobStatus` API を呼び出し`JobSucceeded`て、成功したジョブをレポートする必要があります。

```
/**
 * @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 );
```

データモデルコードジェネレーター

データモデルにコードジェネレーターを使用する方法について説明します。生成されたコードを使用して、クラウドとデバイスの間で交換されるデータモデルをシリアル化および逆シリアル化できます。

プロジェクトリポジトリには、C コードデータモデルハンドラーを作成するためのコード生成ツールが含まれています。以下のトピックでは、コードジェネレーターとワークフローについて説明します。

トピック

- コード生成プロセス

- 環境のセットアップ
- デバイスのコードを生成する

コード生成プロセス

コードジェネレーターは、3 つの主要な入力から C ソースファイルを作成します。 AWS Zigbee クラスターライブラリ (ZCL) アドバンストプラットフォームからの Matter Data Model (.matter ファイル) の実装、前処理を処理する Python プラグイン、およびコード構造を定義する Jinja2 テンプレートです。生成中、Python プラグインはグローバル型定義を追加し、依存関係に基づいてデータ型を整理し、テンプレートレンダリングの情報をフォーマットすることで、.matter ファイルを処理します。

次の図は、コードジェネレーターが C ソースファイルを作成する方法を示しています。

End device SDK には、connectedhomeipプロジェクトcodegen.pyで と連携する Python プラグインと Jinja2 テンプレートが含まれています。この組み合わせにより、.matter ファイルの入力に基づいて、クラスターごとに複数の C ファイルが生成されます。

次のサブトピックでは、これらのファイルについて説明します。

- Python プラグイン
- Jinja2 テンプレート

Python プラグイン

コードジェネレーター はcodegen.py、.matter ファイルを解析し、その情報を Python オブジェクトとしてプラグインに送信します。プラグインファイルは、このデータをiotmi_data_model.py前処理し、提供されたテンプレートを使用してソースをレンダリングします。前処理には以下が含まれます。

1. グローバルタイプなどcodegen.py、 から利用できない情報の追加
2. データ型に対してトポロジーソートを実行して正しい定義の順序を確立する

> **Note**
> トポロジソートにより、元の順序に関係なく、依存関係の後に依存関係のタイプが定義されます。

Jinja2 テンプレート

エンドデバイス SDK は、データモデルハンドラーと低レベルの C 機能用にカスタマイズされた Jinja2 テンプレートを提供します。

Jinja2 テンプレート

テンプレート	生成されたソース	解説
cluster.h.jinja	iotmi_device_<cluster>.h	低レベルの C 関数ヘッダーファイルを作成します。
cluster.c.jinja	iotmi_device_<cluster>.c	データモデルハンドラーにコールバック関数ポインタを実装して登録します。
cluster_type_helpers.h.jinja	iotmi_device_type_helpers_<cluster>.h	データ型の関数プロトタイプを定義します。
cluster_type_helpers.c.jinja	iotmi_device_type_helpers_<cluster>.c	クラスター固有の列挙、ビットマップ、リスト、構造のデータ型関数プロトタイプを生成します。
iot_device_dm_types.h.jinja	iotmi_device_dm_types.h	グローバルデータ型の C データ型を定義します。
iot_device_type_helpers_global.h.jinja	iotmi_device_type_helpers_global.h	グローバルオペレーションの C データ型を定義します。
iot_device_type_helpers_global.c.jinja	iotmi_device_type_helpers_global.c	ブール値、整数、浮動小数点、文字列、ビットマップ、リスト、構造などの標準データ型を宣言します。

環境のセットアップ

codegen.py コードジェネレーターを使用するように環境を設定する方法について説明します。

トピック

- 前提条件
- 環境の設定

前提条件

環境を設定する前に、次の項目をインストールします。

- Git
- Python 3.10 以降
- Poetry 1.2.0 以降

環境の設定

codegen.py コードジェネレーターを使用するように環境を設定するには、次の手順に従います。

1. Python 環境をセットアップします。Codegen プロジェクトは Python ベースで、依存関係管理に Poetry を使用します。

   - `codegen` ディレクトリに poetry を使用してプロジェクトの依存関係をインストールします。

     ```
     poetry run poetry install --no-root
     ```

2. リポジトリをセットアップします。

   a. connectedhomeip リポジトリのクローンを作成します。コード生成には、`connectedhomeip/scripts/`フォルダにある `codegen.py` スクリプトを使用します。詳細については、GitHub の「connectedhomeip」を参照してください。 GitHub

     ```
     git clone https://github.com/project-chip/connectedhomeip.git
     ```

   b. `IoT-managed-integrations-End-Device-SDK` ルートフォルダと同じレベルでクローンを作成します。フォルダ構造は以下と一致する必要があります。

     ```
     |-connectedhomeip
     |-IoT-managed-integrations-End-Device-SDK
     ```

> **Note**
> サブモジュールを再帰的にクローンする必要はありません。

デバイスのコードを生成する

マネージド統合コード生成ツールを使用して、デバイス用にカスタマイズされた C コードを作成します。このセクションでは、 SDK に含まれているサンプルファイルまたは独自の仕様からコードを生成する方法について説明します。生成スクリプトを使用し、ワークフロープロセスを理解し、デバイス要件に合ったコードを作成する方法について説明します。

トピック

- 前提条件
- カスタム .matter ファイルのコードを生成する
- コード生成ワークフロー

前提条件

1. Python 3.10 以降。

2. コード生成用の .matter ファイルから始めます。End device SDK は、 に 2 つのサンプルファイルを提供します`codgen/matter_files folder`。

- `custom-air-purifier.matter`
- `aws_camera.matter`

> **Note**
> これらのサンプルファイルは、デモアプリケーションクラスターのコードを生成します。

コードの生成

次のコマンドを実行して、アウトフォルダにコードを生成します。

```
bash ./gen-data-model-api.sh
```

カスタム .matter ファイルのコードを生成する

特定の`.matter`ファイルのコードを生成したり、独自の`.matter`ファイルを提供するには、次のタスクを実行します。

カスタム .matter ファイルのコードを生成するには

1. .matter ファイルを準備する
2. generation コマンドを実行します。

```
./codegen.sh [--format] configs/dm_basic.json path-to-matter-file output-directory
```

このコマンドは、いくつかのコンポーネントを使用して .matter ファイルを C コードに変換します。

- `codegen.py` ConnectedHomeIP プロジェクトから
- にある Python プラグイン `codegen/py_scripts/iotmi_data_model.py`
- `codegen/py_scripts/templates` フォルダの Jinja2 テンプレート

プラグインは Jinja2 テンプレートに渡す変数を定義し、最終的な C コード出力を生成するために使用されます。`--format` フラグを追加すると、生成されたコードに Clang 形式が適用されます。

コード生成ワークフロー

コード生成プロセスでは、ユーティリティ関数と によるトポロジーソートを使用して、.matter ファイルデータ構造を整理します`topsort.py`。これにより、データ型とその依存関係の適切な順序付けが保証されます。

次に、スクリプトは .matter ファイル仕様と Python プラグイン処理を組み合わせて、必要な情報を抽出してフォーマットします。最後に、Jinja2 テンプレートフォーマットを適用して、最終的な C コード出力を作成します。

このワークフローにより、.matter ファイルからのデバイス固有の要件が、マネージド統合システムと統合される機能 C コードに正確に変換されます。

低レベル C 関数の API オペレーション

提供されている低レベルの C-Function APIs。このセクションでは、デバイスとクラウド間の効率的なやり取りのために、 AWS データモデル内の各クラスターで使用できる API オペレーションについ

て説明します。コールバック関数の実装、イベントの出力、属性の変更の通知、デバイスエンドポイントのクラスターの登録を行う方法について説明します。

主な API コンポーネントは次のとおりです。

1. 属性とコマンドのコールバック関数ポインタ構造
2. イベント放出関数
3. 属性変更通知関数
4. クラスター登録関数

これらの APIs を実装することで、デバイスの物理オペレーションとマネージド統合のクラウド機能の間にブリッジを作成し、シームレスな通信と制御を確保できます。

次のセクションでは、OnOffクラスター API について説明します。

OnOff クラスター API

OnOff.xml クラスターは、次の属性とコマンドをサポートしています: 。

- 属性:
  - `OnOff (boolean)`
  - `GlobalSceneControl (boolean)`
  - `OnTime (int16u)`
  - `OffWaitTime (int16u)`
  - `StartUpOnOff (StartUpOnOffEnum)`
- コマンド:
  - `Off : () -> Status`
  - `On : () -> Status`
  - `Toggle : () -> Status`
  - `OffWithEffect : (EffectIdentifier: EffectIdentifierEnum, EffectVariant: enum8) -> Status`
  - `OnWithRecallGlobalScene : () -> Status`
  - `OnWithTimedOff : (OnOffControl: OnOffControlBitmap, OnTime: int16u, OffWaitTime: int16u) -> Status`

コマンドごとに、実装をフックするために使用できる 1:1 マッピングされた関数ポインタを提供します。

属性とコマンドのすべてのコールバックは、クラスターにちなんだ という名前の C 構造体内で定義されます。

C 構造体の例

```
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);
```

```
};
```

C 構造体に加えて、属性変更レポート関数はすべての属性に定義されます。

```
/* 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);
```

イベントレポート関数は、すべてのクラスター固有のイベントに対して定義されます。OnOff クラスターはイベントを定義しないため、CameraAvStreamManagementクラスターの例を以下に示します。

```
/* 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);
```

各クラスターには登録関数もあります。

```
iotmiDev_DMStatus iotmiDev_OnOffRegisterCluster(struct iotmiDev_Endpoint *endpoint,
 const struct iotmiDev_clusterOnOff *cluster, void *user);
```

登録関数に渡されるユーザーポインタは、コールバック関数に渡されます。

マネージド統合での機能とデバイスのインタラクション

このセクションでは、C-Function 実装の役割と、デバイスとマネージド統合デバイス機能間のやり取りについて説明します。

トピック

- リモートコマンドの処理
- 未承諾イベントの処理

リモートコマンドの処理

リモートコマンドは、エンドデバイス SDK と 機能間の相互作用によって処理されます。次のアクションでは、この操作を使用して電球を有効にする方法の例を示します。

MQTT クライアントがペイロードを受け取り、データモデルハンドラーに渡す

リモートコマンドを送信すると、MQTT クライアントは JSON 形式のマネージド統合からメッセージを受信します。次に、ペイロードをデータモデルハンドラーに渡します。たとえば、 マネージド統合を使用して電球をオンにするとします。電球には、OnOffクラスターをサポートするエンドポイント #1 があります。この場合、電球をオンにするコマンドを送信すると、マネージド統合は MQTT 経由でデバイスにリクエストを送信します。これは、エンドポイント #1 で On コマンドを呼び出す必要があることを示しています。

データモデルハンドラーはコールバック関数をチェックして呼び出します

データモデルハンドラーは JSON リクエストを解析します。リクエストにプロパティまたはアクションが含まれている場合、データモデルハンドラーはエンドポイントを検索し、対応するコールバック関数を順番に呼び出します。例えば、電球の場合、データモデルハンドラーが MQTT メッセージを受信すると、OnOffクラスターで定義された On コマンドに対応するコールバック関数がエンドポイント #1 に登録されているかどうかを確認します。

ハンドラーと C-Function の実装で コマンドを実行する

データモデルハンドラーは、見つけた適切なコールバック関数を呼び出し、呼び出します。次に、C-Function 実装は対応するハードウェア関数を呼び出して物理ハードウェアを制御し、実行結果を返します。例えば、電球の場合、データモデルハンドラーはコールバック関数を呼び出し、実行結果を保存します。その結果、コールバック関数は電球をオンにします。

データモデルハンドラーが実行結果を返す

すべてのコールバック関数が呼び出されると、データモデルハンドラーはすべての結果を組み合わせます。次に、レスポンスを JSON 形式でパックし、MQTT クライアントを使用して結果をマネージド統合クラウドに発行します。電球の場合、レスポンスの MQTT メッセージには、コールバック関数によって電球がオンになった結果が含まれます。

未承諾イベントの処理

未承諾イベントは、エンドデバイス SDK と 機能間の相互作用によっても処理されます。次のアクションでは、その方法について説明します。

デバイスがデータモデルハンドラーに通知を送信する

デバイスで物理的なボタンがプッシュされたときなど、プロパティの変更またはイベントが発生すると、C-Function 実装は未承諾のイベント通知を生成し、対応する通知関数を呼び出してデータモデルハンドラーに通知を送信します。

データモデルハンドラーは通知を翻訳します

データモデルハンドラーは受信した通知を処理し、それを AWS データモデルに変換します。

データモデルハンドラーが クラウドに通知を発行する

次に、データモデルハンドラーは、MQTT クライアントを使用して、未承諾イベントをマネージド統合クラウドに発行します。

エンドデバイス SDK の統合

Linux デバイスで End Device SDK を実行するには、次の手順に従います。このセクションでは、環境設定、ネットワーク設定、ハードウェア機能の実装、エンドポイント設定について説明します。

> **Important**
>
> `examples` ディレクトリのデモンストレーションアプリケーションと のプラットフォーム抽象化レイヤー (PAL) 実装`platform/posix`は、参照のみを目的としています。実稼働環境では使用しないでください。

以下の手順の各ステップを注意深く確認し、マネージド統合とデバイスが適切に統合されていることを確認します。

エンドデバイス SDK の統合

1. ビルド環境をセットアップします。

   開発ホストとして Amazon Linux 2023/x86_64 でコードを構築します。必要なビルドの依存関係をインストールします。

   ```
   dnf install make gcc gcc-c++ cmake
   ```

2. ネットワークをセットアップする

   サンプルアプリケーションを使用する前に、ネットワークを初期化し、デバイスを使用可能な Wi-Fi ネットワークに接続します。デバイスのプロビジョニング前にネットワーク設定を完了します。

```
/* Provisioning the device PKCS11 with claim credential. */
status = deviceCredentialProvisioning();
```

3. プロビジョニングパラメータを設定する

   次のプロビジョニングパラメータ`example/project_name/device_config.sh`を使用して設定ファイルを変更します。

> **Note**
>
> アプリケーションを構築する前に、これらのパラメータを正しく設定してください。

パラメータのプロビジョニング

マクロパラメータ	説明	この情報を取得する方法
`IOTMI_ROOT_CA_PATH`	ルート CA 証明書ファイル。	このファイルは、 デAWS IoT Core ベロッパーガイドの「Amazon ルート CA 証明書のダウンロード」セクションからダウンロードできます。
`IOTMI_CLAIM_CERTIFICATE_PATH`	クレーム証明書ファイルへのパス。	クレーム証明書とプライベートキーを取得するには、CreateProvisioningProfile API を使用してプロビジョニングプロファイルを作成します。手順については、「プロビジョニングプロファイルを作成する」を参照してください。
`IOTMI_CLAIM_PRIVATE_KEY_PATH`	クレームプライベートキーファイルへのパス。	
`IOTMI_MANAGEDINTEGRATIONS_ENDPOINT`	マネージド統合のエンドポイント URL。	マネージド統合エンドポイントを取得するには、RegisterCustomEndpoint API を使用します。手順について

マクロパラメータ	説明	この情報を取得する方法
		は、「カスタムエンドポイントを作成する」を参照してください。
IOTMI_MANAGEDINTEGRATIONS_ENDPOINT_PORT	マネージド統合エンドポイントのポート番号	デフォルトでは、ポート 8883 は MQTT 発行およびサブスクライブオペレーションに使用されます。ポート 443 は、デバイスが使用する Application Layer Protocol Negotiation (ALPN) TLS 拡張機能用に設定されています。

4. ハードウェアコールバック関数の開発

   ハードウェアコールバック関数を実装する前に、API の仕組みを理解してください。この例では、オン/オフクラスターと OnOff 属性を使用してデバイス関数を制御します。API の詳細については、「」を参照してください低レベル C 関数の API オペレーション。

```
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;
}
```

5. エンドポイントをセットアップし、ハードウェアコールバック関数をフックする

   関数を実装したら、エンドポイントを作成し、コールバックを登録します。以下のタスクを実行します。

a. デバイスエージェントを作成します。デバイスエージェントは、すべてのエンドポイントとそのクラスターを管理する中央コンポーネントです。

b. サポートするクラスター構造ごとにコールバック関数ポイントを埋める

c. エンドポイントをセットアップし、サポートされているクラスターを登録する

```
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);
}
```

6. ジョブハンドラーを使用してジョブドキュメントを取得する

   a. OTA アプリケーションへの呼び出しを開始します。

```
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;
}
```

b. `iotmi_JobsHandler_start` を呼び出してジョブハンドラーを初期化します。

c. を呼び出し`iotmi_JobsHandler_getJobDocument`て、マネージド統合からジョブドキュメントを取得します。

d. ジョブドキュメントが正常に取得されたら、 `processOTA`関数にカスタム OTA オペレーションを書き込み、`JobSucceeded`ステータスを返します。

```
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." ) );

}
```

7. デモアプリケーションを構築して実行する

   このセクションでは、CMake をビルドシステムとして使用する、シンプルなセキュリティカメラとエアピューリファイアの 2 つの Linux デモアプリケーションを示します。

   a. シンプルなセキュリティカメラアプリケーション

      アプリケーションを構築して実行するには、次のコマンドを実行します。

```
>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
```

      このデモでは、RTC Session Controller および Recording クラスターを備えたシミュレートされたカメラの低レベルの C 関数を実装します。実行プロビジョンドワークフローする前に、「」で説明されているフローを完了してください。

      デモアプリケーションの出力例：

```
[2406832727][MAIN][INFO] ======= Device initialization and WIFI provisioning
 =======
[2406832728][MAIN][INFO] fleetProvisioningTemplateName: XXXXXXXXXXX
[2406832728][MAIN][INFO] managedintegrationsEndpoint: XXXXXXXXX.account-prefix-
ats.iot.region.amazonaws.com
[2406832728][MAIN][INFO] pDeviceSerialNumber: XXXXXXXXXXXX
[2406832728][MAIN][INFO] universalProductCode: XXXXXXXXXXXX
[2406832728][MAIN][INFO] rootCertificatePath: XXXXXXXXX
[2406832728][MAIN][INFO] pClaimCertificatePath: XXXXXXXX
```

```
[2406832728][MAIN][INFO] pClaimKeyPath: XXXXXXXXXXXXXXXXX
[2406832728][MAIN][INFO] deviceInfo.serialNumber XXXXXXXXXXXX
[2406832728][MAIN][INFO] deviceInfo.universalProductCode XXXXXXXXXXXXXXX
[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 XXXXXXXXXXXXXXX.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: XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
[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 :
 XXXXXXXXXXXXXXXXXXXX_XXXXXXXXXXXXXXXXXXXXXXXX
[2406840540][MAIN][INFO] Managed thing ID : XXXXXXXXXXXXXXXXXXXXXXX
[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 XXXXXXXXX.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. シンプルな空気フィルターアプリケーション

アプリケーションを構築して実行するには、次のコマンドを実行します。

```
>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
```

このデモでは、2 つのエンドポイントと以下のサポートされているクラスターを備えたシミュレートされたエアピューリファイアの低レベル C 関数を実装します。

Air Purifier エンドポイントでサポートされているクラスター

Endpoint	クラスター
エンドポイント #1: Air Purifier	OnOff
	ファンコントロール

Endpoint	クラスター
	HEPA フィルターのモニタリング
	アクティブ化されたカーボンフィルターのモニタリング
エンドポイント #2: Air Quality Sensor	空気品質
	二酸化炭素濃度測定
	フォームの「濃度測定」
	Pm25 濃度測定
	Pm1 濃度測定
	揮発性の有害性化合物の総濃度測定

出力はカメラデモアプリケーションに似ており、サポートされているクラスターは異なります。

End Device SDK をデバイスに移植する

End device SDK をデバイスプラットフォームに移植します。デバイスを AWS IoT Device Management に接続するには、次の手順に従います。

End Device SDK をダウンロードして検証する

1. のマネージド統合 AWS IoT Device Management はパブリックプレビューです。マネージド統合コンソールから End device SDK の最新バージョンをダウンロードします。
2. プラットフォームが でサポートされているプラットフォームのリストに含まれていることを確認します付録 A: サポートされているプラットフォーム。

> **Note**
>
> End device SDK は、指定されたプラットフォームでテストされています。他のプラットフォームは機能する可能性がありますが、まだテストされていません。

3. SDK ファイルをワークスペースに抽出 (解凍) します。
4. 以下の設定でビルド環境を設定します。
   - ソースファイルパス
   - ヘッダーファイルディレクトリ
   - 必要なライブラリ
   - コンパイラとリンカーのフラグ
5. プラットフォーム抽象化レイヤー (PAL) を移植する前に、プラットフォームの基本機能が初期化されていることを確認してください。機能には以下が含まれます。
   - オペレーティングシステムタスク
   - 周辺機器
   - ネットワークインターフェイス
   - プラットフォーム固有の要件

PAL をデバイスに移植する

1. 既存のプラットフォームディレクトリに、プラットフォーム固有の実装用の新しいディレクトリを作成します。たとえば、FreeRTOS を使用する場合は、 にディレクトリを作成しますplatform/freertos。

   Example SDK ディレクトリ構造

```
### <SDK_ROOT_FOLDER>
#   ### CMakeLists.txt
#   ### LICENSE.txt
#   ### cmake
#   ### commonDependencies
#   ### components
#   ### docs
```

```
#   ### examples
#   ### include
#   ### lib
#   ### platform
#   ### test
#   ### tools
```

2. POSIX リファレンス実装ファイル (.c および .h) を posix フォルダから新しいプラットフォームディレクトリにコピーします。これらのファイルは、実装する必要がある関数のテンプレートを提供します。
   - 認証情報ストレージのフラッシュメモリ管理
   - PKCS#11 の実装
   - ネットワークトランスポートインターフェイス
   - 時刻同期
   - システムの再起動とリセット関数
   - ログ記録メカニズム
   - デバイス固有の設定
3. MBedTLS を使用して Transport Layer Security (TLS) 認証を設定します。
   - プラットフォームの SDK バージョンと一致する MBedTLS バージョンが既にある場合は、提供されている POSIX 実装を使用します。
   - 別の TLS バージョンでは、TCP/IP スタックを使用して TLS スタックのトランスポートフックを実装します。
4. プラットフォームの MbedTLS 設定を の SDK `platform/posix/mbedtls/mbedtls_config.h` 要件と比較します。必要なオプションがすべて有効になっていることを確認します。
5. SDK は coreMQTT を使用してクラウドとやり取りします。したがって、次の構造を使用するネットワークトランスポートレイヤーを実装する必要があります。

```
typedef struct TransportInterface
{
    TransportRecv_t recv;
    TransportSend_t send;
    NetworkContext_t * pNetworkContext;
} TransportInterface_t;
```

詳細については、FreeRTOS ウェブサイトの「Transport interface documentation」を参照してください。

6. （オプション) SDK は、PCKS#11 API を使用して証明書オペレーションを処理します。corePKCS は、プロトタイピング用のハードウェア固有ではない PKCS#11 実装です。本番環境では、トラステッドプラットフォームモジュール (TPM)、ハードウェアセキュリティモジュール (HSM)、セキュアエレメントなどの安全な暗号化プロセッサを使用することをお勧めします。

   - での認証情報管理に Linux ファイルシステムを使用するサンプル PKCS#11 実装を確認しますplatform/posix/corePKCS11-mbedtls。
   - で PKCS#11 PAL レイヤーを実装しますcommonDependencies/core_pkcs11/corePKCS11/source/include/core_pkcs11.h。
   - で Linux ファイルシステムを実装しますplatform/posix/corePKCS11-mbedtls/source/iotmi_pal_Pkcs11Operations.c。
   - ストレージタイプのストアおよびロード関数を に実装しますplatform/include/iotmi_pal_Nvm.h。
   - で標準ファイルアクセスを実装しますplatform/posix/source/iotmi_pal_Nvm.c。

   移植手順の詳細については、FreeRTOS ユーザーガイドの corePKCS11 ライブラリの移植を参照してください。

7. SDK 静的ライブラリをビルド環境に追加します。

   - リンカーの問題や記号の競合を解決するためのライブラリパスの設定
   - すべての依存関係が適切にリンクされていることを確認します。

ポートをテストする

既存のサンプルアプリケーションを使用してポートをテストできます。コンパイルは、エラーや警告なしで完了する必要があります。

> **Note**
>
> 可能な限りシンプルなマルチタスクアプリケーションから始めることをお勧めします。サンプルアプリケーションは、マルチタスクに相当するものを提供します。

1. でサンプルアプリケーションを見つけますexamples/[device_type_sample]。
2. main.c ファイルをプロジェクトに変換し、既存の main() 関数を呼び出すエントリを追加します。
3. デモアプリケーションを正常にコンパイルできることを確認します。

付録

トピック

- 付録 A: サポートされているプラットフォーム
- 付録 B: 技術要件
- 付録 C: 共通 API

付録 A: サポートされているプラットフォーム

次の表に、 SDK でサポートされているプラットフォームを示します。

サポートされているプラットフォーム

プラットフォーム	アーキテクチャ	オペレーティングシステム
Linux x86_64	x86_64	リナックス
アンバレラ	Armv8 (AArch64)	リナックス
AmebaD	Armv8-M 32 ビット	FreeRTOS
ESP32S3	Xtensa LX7 32 ビット	FreeRTOS

付録 B: 技術要件

次の表は、RAM スペースを含む SDK の技術要件を示しています。同じ設定を使用する場合、エンドデバイス SDK 自体には約 5 ~ 10 MB の ROM スペースが必要です。

RAM スペース

SDK とコンポーネント	スペース要件 (バイト使用)
エンドデバイス SDK 自体	180 KB
デフォルトの MQTT エージェントコマンドキュー	480 バイト (設定可能)
デフォルトの MQTT エージェント受信キュー	320 バイト (設定可能)

付録 C: 共通 API

このセクションでは、クラスターに固有ではない API オペレーションのリストを示します。

```
/* 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);
```

プロトコル固有のミドルウェアとは

> **Important**
>
> ここで提供されるドキュメントとコードでは、ミドルウェアのリファレンス実装について説明します。SDK の一部として提供されるものではありません。

プロトコル固有のミドルウェアには、基盤となるプロトコルスタックを操作する上で重要な役割があります。 AWS IoT Smart Home Hub SDK のデバイスオンボーディングコンポーネントとデバイスコントロールコンポーネントはどちらも、これを使用してエンドデバイスとやり取りします。

ミドルウェアは次の関数を実行します。

- 一般的な APIs セットを提供することで、さまざまなベンダーのデバイスプロトコルスタックから APIs。
- スレッドスケジューラ、イベントキュー管理、データキャッシュなどのソフトウェア実行管理を提供します。
- Zigbee クラスターライブラリ (ZCL) や BLE メッシュなどのプロトコル固有のアプリケーションスタック。

ミドルウェアアーキテクチャ

以下のブロック図は、Zigbee ミドルウェアのアーキテクチャを示しています。Z-Wave などの他のプロトコルのミドルウェアのアーキテクチャも似ています。

プロトコル固有のミドルウェアには 3 つの主要コンポーネントがあります。

- ACS Zigbee DPK: Zigbee Device Porting Kit (DPK) は、基盤となるハードウェアとオペレーティングシステムから抽象化するために使用され、移植性を可能にします。基本的に、これはハードウェア抽象化レイヤー (HAL) と見なすことができます。HAL は、さまざまなベンダーの Zigbee 無線を制御して通信するための共通のセット APIs を提供します。Zigbee ミドルウェアには、Silicon Labs Zigbee アプリケーションフレームワークの DPK API 実装が含まれています。
- ACS Zigbee サービス: Zigbee サービスは専用のデーモンとして実行されます。これには、IPC チャネルを介してクライアントアプリケーションからの API コールを処理する API ハンドラーが

含まれています。AIPC は、Zigbee アダプターと Zigbee サービス間の IPC チャネルとして使用されます。非同期/同期コマンドの処理、HAL からのイベントの処理、イベント登録/発行に ACS Event Manager を使用するなどの他の機能を提供します。

- ACS Zigbee アダプター: Zigbee アダプターは、アプリケーションプロセス内で実行されるライブラリです (この場合、アプリケーションは CDMB プラグインです) 。Zigbee アダプターは、エンドデバイスを制御して通信するために CDMB/Provisioner プロトコルプラグインなどのクライアントアプリケーションで消費される一連の APIs を提供します。

End-to-endミドルウェアコマンドフローの例

Zigbee ミドルウェアを介したコマンドフローの例を次に示します。

Z-Wave ミドルウェアを介したコマンドフローの例を次に示します。

プロトコル固有のミドルウェアコードの整理

このセクションでは、`IotManagedIntegrationsDeviceSDK-Middleware`リポジトリ内の各コンポーネントのコードの場所について説明します。このリポジトリのフォルダ構造の例を次に示します。

```
./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
```

トピック

- Zigbee ミドルウェアコードの整理
- Z-Wave ミドルウェアコードの整理

Zigbee ミドルウェアコードの整理

以下は、Zigbee リファレンスミドルウェアコードの組織を示しています。

トピック

- ACS Zigbee DPK
- Silicon Labs Zigbee SDK
- ACS Zigbee サービス
- ACS Zigbee アダプター

ACS Zigbee DPK

Zigbee DPK のコードは `IotManagedIntegrationsDeviceSDK-Middleware/telus-iot-ace-dpk/telus/dpk/ace_hal/zigbee`フォルダ内にあります。

```
./IotManagedIntegrationsDeviceSDK-Middleware/example-iot-ace-dpk/example/dpk/ace_hal/
|- common
|-   |- fxnDbusClient
|-   |- include
|- kvs
|- log
|- wifi
|-   |- include
|-   |- src
|-   |- wifid
|-       |- fxnWifiClient
|-       |- include
|- zibgee
|-   |- include
|-   |- src
|-   |- zigbeed
|-       |- ember
|-       |- include
|- zwave
|-   |- include
|-   |- src
|-   |- zwaved
|-       |- fxnZwaveClient
|-       |- include
|-       |- zware
```

Silicon Labs Zigbee SDK

Silicon Labs SDK は `IotManagedIntegrationsDeviceSDK-Middleware/example-iot-ace-z3-gateway`フォルダ内に表示されます。この ACS Zigbee DPK レイヤーは、この Silicon Labs SDK に実装されています。

```
./IotManagedIntegrationsDeviceSDK-Middleware/example-iot-ace-zz3-gateway/
|— autogen
|— config
|— gecko_sdk_4.3.2
|—   |— platform
|—   |— protocol
|—   |— util
```

ACS Zigbee サービス

Zigbee サービスのコードは `IotManagedIntegrationsDeviceSDK-Middleware/example-iot-ace-general/middleware/zigbee/`フォルダ内にあります。この場所の `src`および `include`サブフォルダには、ACS Zigbee サービスに関連するすべてのファイルが含まれています。

```
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_uml.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
```

ACS Zigbee アダプター

ACS Zigbee アダプターのコードは `IotManagedIntegrationsDeviceSDK-Middleware/`*`example`*`-iot-ace-general/middleware/zigbee/api`フォルダ内にあります。この場所の `src`および `include`サブフォルダには、ACS Zigbee アダプターライブラリに関連するすべてのファイルが含まれています。

```
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
```

Z-Wave ミドルウェアコードの整理

以下は、Z ウェーブリファレンスミドルウェアコードの組織を示しています。

トピック

- ACS Z-Wave DPK
- Silicon Labs の ZWare と Zip Gateway
- ACS Z-Wave サービス
- ACS Z-Wave アダプター

ACS Z-Wave DPK

Z-Wave DPK のコードは `IotManagedIntegrationsDeviceSDK-Middleware/example-iot-ace-dpk/example/dpk/ace_hal/zwave`フォルダ内にあります。

```
./IotManagedIntegrationsDeviceSDK-Middleware/example-iot-ace-dpk/example/dpk/ace_hal/
|- common
|-    |- fxnDbusClient
|-    |- include
|- kvs
|- log
|- wifi
|-    |- include
|-    |- src
|-    |- wifid
|-        |- fxnWifiClient
|-        |- include
|- zibgee
|-    |- include
|-    |- src
|-    |- zigbeed
|-        |- ember
|-        |- include
|- zwave
|-    |- include
|-    |- src
|-    |- zwaved
```

```
|-       |- fxnZwaveClient
|-       |- include
|-       |- zware
```

Silicon Labs の ZWare と Zip Gateway

Silicon Labs ZWare および Zip Gateway のコードは、 `IotManagedIntegrationsDeviceSDK-Middleware/example-iot-ace-z3-gateway`フォルダ内にあります。この ACS Z-Wave DPK レイヤーは、Z-Wave C-APIs。

```
./IotManagedIntegrationsDeviceSDK-Middleware/example-iot-ace-z3-gateway/
|- autogen
|- config
|- gecko_sdk_4.3.2
|-   |- platform
|-   |- protocol
|-   |- util
```

ACS Z-Wave サービス

Z-Wave サービスのコードは `IoTmanagedintegrationsMiddlewares/telus-iot-ace-zwave-mw/`フォルダ内にあります。この場所にある フォルダ`src`と `include`フォルダには、ACS Z-Wave サービスに関連するすべてのファイルが含まれています。

```
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
```

ACS Z-Wave アダプター

ACS Zigbee アダプターのコードは `IotManagedIntegrationsDeviceSDK-Middleware/example-iot-ace-zwave-mw/cli/`フォルダ内にあります。この場所の `src`および `include`フォルダには、ACS Z-Wave アダプターライブラリに関連するすべてのファイルが含まれています。

```
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
```

デバイス SDK のミドルウェア部分をハブに統合する

新しいハブでのミドルウェア統合については、以下のセクションで説明します。

トピック

- デバイス移植キット (DPK) API 統合
- リファレンス実装とコード編成

デバイス移植キット (DPK) API 統合

チップセットベンダー SDK をミドルウェアと統合するには、中央の DPK (デバイス移植キット) レイヤーによって標準の API インターフェイスが提供されます。マネージド統合サービスプロバイダーまたは ODMs は、IoT ハブで使用される Zigbee/Z-wave/Wi-Fi チップセットでサポートされているベンダー SDK に基づいて、これらの APIs を実装する必要があります。

リファレンス実装とコード編成

ミドルウェアを除き、マネージド統合 Device Agent や Common Data Model Bridge (CDMB) などの他のすべての Device SDK コンポーネントは、変更なしで使用でき、クロスコンパイルのみが必要です。

ミドルウェアの実装は、Silicon Labs SDK for Zigbee と Z-Wave に基づいています。新しいハブで使用される Z-Wave および Zigbee チップセットがミドルウェアに存在する Silicon Labs SDK でサポートされている場合、リファレンスミドルウェアは変更なしで使用できます。ミドルウェアをクロスコンパイルするだけで、新しいハブで実行できます。

Zigbee 用の DPK (デバイス移植キット) APIs は にあり`acehal_zigbee.c`、DPK APIs のリファレンス実装は `zigbee`フォルダ内にあります。

```
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
```

Z-Wave の DPK APIs は にあり`acehal_zwave.c`、DPK APIs のリファレンス実装は `zwaved`フォルダ内にあります。

```
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
```

別のベンダー SDK の DPK レイヤーを実装するための出発点として、リファレンス実装を使用および変更できます。別のベンダー SDK をサポートするには、次の 2 つの変更が必要です。

1. 現在のベンダー SDK をリポジトリの新しいベンダー SDK に置き換えます。
2. 新しいベンダー SDK に従ってミドルウェア DPK (デバイス移植キット) APIs を実装します。

のマネージド統合のセキュリティ AWS IoT Device Management

のクラウドセキュリティが最優先事項 AWS です。お客様は AWS 、最もセキュリティの影響を受けやすい組織の要件を満たすように構築されたデータセンターとネットワークアーキテクチャからメリットを得られます。

セキュリティは、 AWS とお客様の間で共有される責任です。責任共有モデルでは、これをクラウドのセキュリティおよびクラウド内のセキュリティと説明しています。

- クラウドのセキュリティ – AWS は、 で AWS サービスを実行するインフラストラクチャを保護する責任を担います AWS クラウド。 AWS また、 は、お客様が安全に使用できるサービスも提供します。サードパーティーの監査者は、AWS コンプライアンスプログラムコンプライアンスプログラムの一環として、当社のセキュリティの有効性を定期的にテストおよび検証。マネージド統合に適用されるコンプライアンスプログラムの詳細については、「コンプライアンスプログラムAWS による対象範囲内のサービスコンプライアンスプログラム」を参照してください。
- クラウドのセキュリティ – お客様の責任は、使用する AWS サービスによって決まります。また、ユーザーは、データの機密性、会社の要件、適用される法律や規制など、その他の要因についても責任を負います。

このドキュメントは、 マネージド統合を使用する際の責任共有モデルの適用方法を理解するのに役立ちます。以下のトピックでは、セキュリティおよびコンプライアンスの目的を達成するためにマネージド統合を設定する方法を示します。また、マネージド統合リソースのモニタリングと保護に役立つ他の AWS サービスの使用方法についても説明します。

トピック

- マネージド型統合でのデータ保護
- マネージド統合の ID とアクセスの管理
- マネージド統合のコンプライアンス検証
- マネージド統合の耐障害性

マネージド型統合でのデータ保護

責任 AWS 共有モデル、 マネージド統合のデータ保護に適用されます AWS IoT Device Management。このモデルで説明されているように、 AWS はすべての を実行するグローバルインフ

ラストラクチャを保護する責任があります AWS クラウド。ユーザーは、このインフラストラクチャでホストされるコンテンツに対する管理を維持する責任があります。また、使用する「 AWS のサービス 」のセキュリティ設定と管理タスクもユーザーの責任となります。データプライバシーの詳細については、データプライバシーに関するよくある質問を参照してください。欧州でのデータ保護の詳細については、AWS セキュリティブログに投稿された AWS 責任共有モデルおよび GDPR のブログ記事を参照してください。

データ保護の目的で、認証情報を保護し AWS アカウント 、 AWS IAM Identity Center または AWS Identity and Access Management (IAM) を使用して個々のユーザーを設定することをお勧めします。この方法により、それぞれのジョブを遂行するために必要な権限のみが各ユーザーに付与されます。また、次の方法でデータを保護することもお勧めします:

- 各アカウントで多要素認証 (MFA) を使用します。
- SSL/TLS を使用して AWS リソースと通信します。TLS 1.2 が必須で、TLS 1.3 をお勧めします。
- で API とユーザーアクティビティのログ記録を設定します AWS CloudTrail。CloudTrail 証跡を使用して AWS アクティビティをキャプチャする方法については、「 AWS CloudTrail ユーザーガイド」のCloudTrail 証跡の使用」を参照してください。
- AWS 暗号化ソリューションと、その中のすべてのデフォルトのセキュリティコントロールを使用します AWS のサービス。
- Amazon Macie などの高度な管理されたセキュリティサービスを使用します。これらは、Amazon S3 に保存されている機密データの検出と保護を支援します。
- コマンドラインインターフェイスまたは API AWS を介して にアクセスするときに FIPS 140-3 検証済み暗号化モジュールが必要な場合は、FIPS エンドポイントを使用します。利用可能な FIPS エンドポイントの詳細については、「連邦情報処理規格 (FIPS) 140-3」を参照してください。

お客様の E メールアドレスなどの極秘または機密情報を、タグ、または [名前] フィールドなどの自由形式のテキストフィールドに含めないことを強くお勧めします。これは、コンソール、API、AWS IoT Device Management または AWS CLI SDK を使用して または他の AWS のサービス のマネージド統合を使用する場合も同様です。 AWS SDKs タグ、または名前に使用される自由記述のテキストフィールドに入力したデータは、請求または診断ログに使用される場合があります。外部サーバーに URL を提供する場合、そのサーバーへのリクエストを検証できるように、認証情報を URL に含めないことを強くお勧めします。

マネージド統合の保管時のデータ暗号化

のマネージド統合 AWS IoT Device Management は、デフォルトでデータ暗号化を提供し、暗号化キーを使用して保管中の機密データを保護します。

マネージド統合のお客様の機密データを保護するために使用する暗号化キーには、次の 2 種類があります。

カスタマーマネージドキー (CMK)

マネージド統合は、作成、所有、管理できる対称カスタマーマネージドキーの使用をサポートします。これらの KMS キーは、キーポリシー、IAM ポリシー、グラントの確立と管理、有効化と無効化、暗号化マテリアルのローテーション、タグの追加、KMS キーを参照するエイリアスの作成 、KMS キー削除のスケジューリングなどを完全に制御できます。

AWS 所有キー

マネージド統合では、デフォルトでこれらのキーを使用して、機密データを自動的に暗号化します。それらの使用を表示、管理、監査することはできません。データを暗号化するキーを保護するために、アクションを実行したり、プログラムを変更したりする必要はありません。デフォルトでは、保管中のデータを暗号化することで、機密データの保護に伴う運用のオーバーヘッドと複雑な作業を軽減できます。同時に、セキュリティを重視したアプリケーションを構築することで、暗号化のコンプライアンスと規制の厳格な要件を満たすことができます。

使用されるデフォルトの暗号化キーは、 AWS 所有キーです。または、暗号化キーを更新するオプションの API は ですPutDefaultEncryptionConfiguration。

AWS KMS 暗号化キーのタイプの詳細については、AWS KMS 「 キー」を参照してください。

AWS KMS マネージド統合の の使用

マネージド統合は、エンベロープ暗号化を使用してすべての顧客データを暗号化および復号します。このタイプの暗号化では、プレーンテキストデータを取得し、データキーで暗号化します。次に、ラッピングキーと呼ばれる暗号化キーは、プレーンテキストデータの暗号化に使用される元のデータキーを暗号化します。エンベロープ暗号化では、追加のラッピングキーを使用して、元のデータキーからの分離度が近い既存のラッピングキーを暗号化できます。元のデータキーは個別に保存されているラッピングキーによって暗号化されるため、元のデータキーと暗号化されたプレーンテキストデータを同じ場所に保存できます。キーリングは、データキーの暗号化と復号に使用されるラッピングキーに加えて、データキーの生成、暗号化、復号に使用されます。

> **Note**
>
> AWS Database Encryption SDK は、クライアント側の暗号化実装にエンベロープ暗号化を提供します。 AWS Database Encryption SDK の詳細については、AWS 「Database Encryption SDK とは」を参照してください。

エンベロープ暗号化、データキー、ラッピングキー、およびキーリングの詳細については、「エンベロープ暗号化、データキー、ラッピングキー、およびキーリング」を参照してください。

マネージド統合では、以下の内部オペレーションにカスタマーマネージドキーを使用する必要があります。

- にDescribeKeyリクエストを送信 AWS KMS して、データキーのローテーションの実行時に提供された対称カスタマーマネージドキー ID を確認します。
- カスタマーマネージドキーによって暗号化されたデータキーを生成するGenerateDataKeyWithoutPlaintextリクエスト AWS KMS を に送信します。
- カスタマーマネージドキーによってデータキーを再暗号化 AWS KMS するReEncrypt*リクエストを に送信します。
- カスタマーマネージドキーによってデータを復号 AWS KMS 化するためのDecryptリクエストをに送信します。

暗号化キーを使用して暗号化されたデータのタイプ

マネージド統合では、暗号化キーを使用して、保管時に保存される複数のタイプのデータを暗号化します。次のリストは、暗号化キーを使用して保管時に暗号化されたデータのタイプの概要を示しています。

- デバイス検出やデバイスステータスの更新などのクラウド間 (C2C) コネクタイベント。
- 物理デバイスmanagedThingを表す と、特定のデバイスタイプの機能を含むデバイスプロファイルの作成。デバイスとデバイスプロファイルの詳細については、デバイス「」および「」を参照してくださいデバイス。
- デバイス実装のさまざまな側面に関するマネージド統合通知。マネージド統合通知の詳細については、「」を参照してくださいマネージド統合通知の設定。
- デバイス認証マテリアル、デバイスのシリアル番号、エンドユーザーの名前、デバイス識別子、デバイスの Amazon リソースネーム (arn) など、エンドユーザーの個人を特定できる情報 (PII)。

でマネージド統合がキーポリシーを使用する方法 AWS KMS

ブランチキーのローテーションと非同期呼び出しの場合、マネージド統合には暗号化キーを使用するためのキーポリシーが必要です。キーポリシーは、次の理由で使用されます。

- 暗号化キーの使用を他の AWS プリンシパルにプログラムで認可します。

マネージド統合で暗号化キーへのアクセスを管理するために使用されるキーポリシーの例については、「」を参照してください。 暗号化キーを作成する

> **Note**
>
> AWS 所有キーの場合、キーポリシーは必要ありません。 AWS 所有キーは によって所有 AWS されており、表示、管理、使用することはできません。マネージド統合では、デフォルトで AWS 所有キーを使用して、機密データを自動的に暗号化します。

マネージド統合では、キーポリシーを使用してキーで AWS KMS 暗号化設定を管理するだけでなく、IAM ポリシーも使用します。IAM ポリシーの詳細については、「 のポリシーとアクセス許可」を参照してください AWS Identity and Access Management。

暗号化キーを作成する

AWS Management Console または AWS KMS APIs を使用して暗号化キーを作成できます。

暗号化キーを作成するには

「 AWS Key Management Service デベロッパーガイド」の「KMS キーの作成」の手順に従います。

キーポリシー

キーポリシーステートメントは、 AWS KMS キーへのアクセスを制御します。各 AWS KMS キーには 1 つのキーポリシーのみが含まれます。このキーポリシーは、キーを使用できる AWS プリンシパルとその使用方法を決定します。キーポリシーステートメントを使用した AWS KMS キーのアクセスと使用の管理の詳細については、「ポリシーを使用したアクセスの管理」を参照してください。

以下は、 マネージド統合 AWS アカウント の に保存されているキーへのアクセスと使用状況を管理するために使用できる AWS KMS キーポリシーステートメントの例です。

```
{
   "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": "*"
    }
  ]
 }
```

キーストアの詳細については、「キーストア」を参照してください。

暗号化設定の更新

マネージド統合のデータ暗号化実装を管理するには、暗号化設定をシームレスに更新する機能が不可欠です。マネージド統合で最初にオンボードすると、暗号化設定を選択するように求められます。オプションは、デフォルトの AWS 所有キーであるか、独自の AWS KMS キーを作成します。

AWS Management Console

で暗号化設定を更新するには AWS Management Console、 AWS IoT サービスのホームページを開き、Managed Integration for Unified Control > Settings > Encryption に移動します。暗号化設定ウィンドウで、暗号化設定を更新するには、新しい AWS KMS キーを選択して暗号化保護を強化します。暗号化設定をカスタマイズ (アドバンスド) を選択して既存の AWS KMS キーを選択するか、AWS KMS キーの作成を選択して独自のカスタマーマネージドキーを作成します。

API コマンド

マネージド統合で AWS KMS キーの暗号化設定を管理するために使用される APIs は `PutDefaultEncryptionConfiuration`と の 2 つですGetDefaultEncryptionConfiguration。

デフォルトの暗号化設定を更新するには、 を呼び出しますPutDefaultEncryptionConfiuration。の詳細についてはPutDefaultEncryptionConfiuration、「PutDefaultEncryptionConfiuration」を参照してください。

デフォルトの暗号化設定を表示するには、 を呼び出しますGetDefaultEncryptionConfiguration。の詳細についてはGetDefaultEncryptionConfiguration、「GetDefaultEncryptionConfiguration」を参照してください。

マネージド統合の ID とアクセスの管理

AWS Identity and Access Management (IAM) は、管理者が AWS リソースへのアクセスを安全に制御 AWS のサービス するのに役立つ です。IAM 管理者は、誰を認証 (サインイン) し、誰にマネージド統合リソースの使用を許可する (アクセス許可を付与する) かを制御します。IAM は、追加料金なしで使用できる AWS のサービス です。

トピック

- 対象者
- アイデンティティを使用した認証
- ポリシーを使用したアクセスの管理
- AWS マネージド統合の マネージドポリシー
- マネージド統合と IAM の連携方法
- マネージド統合のアイデンティティベースのポリシーの例
- マネージド統合のアイデンティティとアクセスのトラブルシューティング
- マネージド統合でサービスにリンクされたロールを使用する

対象者

AWS Identity and Access Management (IAM) の使用方法は、マネージド統合で行う作業によって異なります。

サービスユーザー – マネージド統合サービスを使用してジョブを実行する場合、管理者から必要な認証情報とアクセス許可が提供されます。より多くのマネージド統合機能を使用して作業を行うには、追加のアクセス許可が必要になる場合があります。アクセスの管理方法を理解すると、管理者に適切なアクセス許可をリクエストするのに役に立ちます。マネージド統合の機能にアクセスできない場合は、「」を参照してくださいマネージド統合のアイデンティティとアクセスのトラブルシューティング。

サービス管理者 – 社内のマネージド統合リソースを担当している場合は、通常、マネージド統合へのフルアクセスがあります。サービスユーザーがどのマネージド統合機能とリソースにアクセスする必要があるかを決めるのは管理者の仕事です。その後、IAM 管理者にリクエストを送信して、サービスユーザーの権限を変更する必要があります。このページの情報を点検して、IAM の基本概念を理解してください。マネージド統合で IAM を使用する方法の詳細については、「」を参照してくださいマネージド統合と IAM の連携方法。

IAM 管理者 - IAM 管理者は、マネージド統合へのアクセスを管理するポリシーの作成方法の詳細について確認する場合があります。IAM で使用できるマネージド統合のアイデンティティベースのポリシーの例を表示するには、「」を参照してくださいマネージド統合のアイデンティティベースのポリシーの例。

アイデンティティを使用した認証

認証は、ID 認証情報 AWS を使用して にサインインする方法です。として、IAM ユーザーとして AWS アカウントのルートユーザー、または IAM ロールを引き受けることによって、認証 (にサインイン AWS) される必要があります。

ID ソースを介して提供された認証情報を使用して、フェデレーティッド ID AWS として にサインインできます。 AWS IAM Identity Center （ IAM Identity Center) ユーザー、会社のシングルサインオン認証、Google または Facebook 認証情報は、フェデレーティッド ID の例です。フェデレーティッド ID としてサインインする場合、IAM ロールを使用して、前もって管理者により ID フェデレーションが設定されています。フェデレーションを使用して にアクセスすると、間接的 AWS にロールを引き受けます。

使用するユーザーのタイプに応じて、 AWS Management Console または AWS アクセスポータルにサインインできます。へのサインインの詳細については AWS、 AWS サインイン ユーザーガイドの「 へのサインイン方法 AWS アカウント」を参照してください。

AWS プログラムで にアクセスする場合、 は Software Development Kit (SDK) とコマンドラインインターフェイス (CLI) AWS を提供し、認証情報を使用してリクエストを暗号化して署名します。AWS ツールを使用しない場合は、自分でリクエストに署名する必要があります。リクエストに自分で署名する推奨方法の使用については、「IAM ユーザーガイド」の「API リクエストに対するAWS Signature Version 4」を参照してください。

使用する認証方法を問わず、追加セキュリティ情報の提供をリクエストされる場合もあります。たとえば、 では、多要素認証 (MFA) を使用してアカウントのセキュリティを向上させる AWS ことをお勧めします。詳細については、「AWS IAM Identity Center ユーザーガイド」の「多要素認証」および「IAM ユーザーガイド」の「IAM のAWS 多要素認証」を参照してください。

AWS アカウント ルートユーザー

を作成するときは AWS アカウント、アカウント内のすべての およびリソースへの AWS のサービス完全なアクセス権を持つ 1 つのサインインアイデンティティから始めます。この ID は AWS アカウント ルートユーザーと呼ばれ、アカウントの作成に使用した E メールアドレスとパスワードでサインインすることでアクセスできます。日常的なタスクには、ルートユーザーを使用しないことを強くお勧めします。ルートユーザーの認証情報は保護し、ルートユーザーでしか実行できないタスクを実行するときに使用します。ルートユーザーとしてサインインする必要があるタスクの完全なリストについては、「IAM ユーザーガイド」の「ルートユーザー認証情報が必要なタスク」を参照してください。

フェデレーティッドアイデンティティ

ベストプラクティスとして、管理者アクセスを必要とするユーザーを含む人間のユーザーに、一時的な認証情報を使用して にアクセスするために ID プロバイダーとのフェデレーション AWS のサービス の使用を要求します。

フェデレーティッド ID は、エンタープライズユーザーディレクトリ、ウェブ ID プロバイダー、、AWS Directory Serviceアイデンティティセンターディレクトリ、または ID ソースを通じて提供された認証情報 AWS のサービス を使用して にアクセスするすべてのユーザーです。フェデレーティッド ID がアクセスすると AWS アカウント、ロールを引き受け、ロールは一時的な認証情報を提供します。

アクセスを一元管理する場合は、 AWS IAM Identity Centerを使用することをお勧めします。IAM Identity Center でユーザーとグループを作成するか、独自の ID ソース内のユーザーとグループのセットに接続して同期し、すべての AWS アカウント とアプリケーションで使用できます。IAM Identity Center の詳細については、「AWS IAM Identity Center ユーザーガイド」の「What is IAM Identity Center?」(IAM Identity Center とは) を参照してください。

IAM ユーザーとグループ

IAM ユーザーは、単一のユーザーまたはアプリケーションに対して特定のアクセス許可 AWS アカウント を持つ 内のアイデンティティです。可能であれば、パスワードやアクセスキーなどの長期的な認証情報を保有する IAM ユーザーを作成する代わりに、一時的な認証情報を使用することをお勧めします。ただし、IAM ユーザーでの長期的な認証情報が必要な特定のユースケースがある場合は、アクセスキーをローテーションすることをお勧めします。詳細については、「IAM ユーザーガイド」の「長期的な認証情報を必要とするユースケースのためにアクセスキーを定期的にローテーションする」を参照してください。

IAM グループは、IAM ユーザーの集団を指定するアイデンティティです。グループとしてサインインすることはできません。グループを使用して、複数のユーザーに対して一度に権限を指定できます。多数のユーザーグループがある場合、グループを使用することで権限の管理が容易になります。例えば、IAMAdmins という名前のグループを設定して、そのグループに IAM リソースを管理する許可を与えることができます。

ユーザーは、ロールとは異なります。ユーザーは 1 人の人または 1 つのアプリケーションに一意に関連付けられますが、ロールはそれを必要とする任意の人が引き受けるようになっています。ユーザーには永続的な長期の認証情報がありますが、ロールでは一時認証情報が提供されます。詳細については、「IAM ユーザーガイド」の「IAM ユーザーに関するユースケース」を参照してください。

IAM ロール

IAM ロールは、特定のアクセス許可 AWS アカウント を持つ 内の ID です。これは IAM ユーザーに似ていますが、特定のユーザーには関連付けられていません。で IAM ロールを一時的に引き受けるには AWS Management Console、ユーザーから IAM ロール (コンソール) に切り替えることができます。ロールを引き受けるには、 または AWS API オペレーションを AWS CLI 呼び出すか、カスタム URL を使用します。ロールを使用する方法の詳細については、「IAM ユーザーガイド」の「ロールを引き受けるための各種方法」を参照してください。

IAM ロールと一時的な認証情報は、次の状況で役立ちます:

- フェデレーションユーザーアクセス – フェデレーティッド ID に許可を割り当てるには、ロールを作成してそのロールの許可を定義します。フェデレーティッド ID が認証されると、その ID はロールに関連付けられ、ロールで定義されている許可が付与されます。フェデレーションのロールについては、「IAM ユーザーガイド」の「サードパーティー ID プロバイダー (フェデレーション) 用のロールを作成する」を参照してください。IAM Identity Center を使用する場合は、許可セットを設定します。アイデンティティが認証後にアクセスできるものを制御するため、IAM Identity Center は、権限セットを IAM のロールに関連付けます。アクセス許可セットの詳細については、「AWS IAM Identity Center User Guide」の「Permission sets」を参照してください。
- 一時的な IAM ユーザー権限 - IAM ユーザーまたはロールは、特定のタスクに対して複数の異なる権限を一時的に IAM ロールで引き受けることができます。
- クロスアカウントアクセス - IAM ロールを使用して、自分のアカウントのリソースにアクセスすることを、別のアカウントの人物 (信頼済みプリンシパル) に許可できます。クロスアカウントアクセス権を付与する主な方法は、ロールを使用することです。ただし、一部の では AWS のサービス、 (ロールをプロキシとして使用する代わりに) リソースに直接ポリシーをアタッチできます。クロスアカウントアクセスにおけるロールとリソースベースのポリシーの違いについては、「IAM ユーザーガイド」の「IAM でのクロスアカウントのリソースへのアクセス」を参照してください。
- クロスサービスアクセス — 一部の は他の の機能 AWS のサービス を使用します AWS のサービス。例えば、あるサービスで呼び出しを行うと、通常そのサービスによって Amazon EC2 でアプリケーションが実行されたり、Amazon S3 にオブジェクトが保存されたりします。サービスでは、呼び出し元プリンシパルの許可、サービスロール、またはサービスリンクロールを使用してこれを行う場合があります。
  - 転送アクセスセッション (FAS) – IAM ユーザーまたはロールを使用して でアクションを実行すると AWS、プリンシパルと見なされます。一部のサービスを使用する際に、アクションを実行することで、別のサービスの別のアクションがトリガーされることがあります。FAS は、 を呼び出すプリンシパルのアクセス許可と AWS のサービス、ダウンストリームサービス AWS の

サービス へのリクエストをリクエストする を使用します。FAS リクエストは、サービスが他の AWS のサービス またはリソースとのやり取りを完了する必要があるリクエストを受け取った場合にのみ行われます。この場合、両方のアクションを実行するためのアクセス許可が必要です。FAS リクエストを行う際のポリシーの詳細については、「転送アクセスセッション」を参照してください。

- サービスロール - サービスがユーザーに代わってアクションを実行するために引き受ける IAM ロールです。IAM 管理者は、IAM 内からサービスロールを作成、変更、削除することができます。詳細については、「IAM ユーザーガイド」の「AWS のサービスに許可を委任するロールを作成する」を参照してください。
- サービスにリンクされたロール – サービスにリンクされたロールは、 にリンクされたサービスロールの一種です AWS のサービス。サービスは、ユーザーに代わってアクションを実行するロールを引き受けることができます。サービスにリンクされたロールは に表示され AWS アカウント 、 サービスによって所有されます。IAM 管理者は、サービスリンクロールのアクセス許可を表示できますが、編集することはできません。

- Amazon EC2 で実行されているアプリケーション – IAM ロールを使用して、EC2 インスタンスで実行され、 AWS CLI または AWS API リクエストを行うアプリケーションの一時的な認証情報を管理できます。これは、EC2 インスタンス内でのアクセスキーの保存に推奨されます。 AWS ロールを EC2 インスタンスに割り当て、そのすべてのアプリケーションで使用できるようにするには、インスタンスにアタッチされたインスタンスプロファイルを作成します。インスタンスプロファイルにはロールが含まれ、EC2 インスタンスで実行されるプログラムは一時的な認証情報を取得できます。詳細については、「IAM ユーザーガイド」の「Amazon EC2 インスタンスで実行されるアプリケーションに IAM ロールを使用して許可を付与する」を参照してください。

ポリシーを使用したアクセスの管理

でアクセスを制御する AWS には、ポリシーを作成し、ID AWS またはリソースにアタッチします。ポリシーは AWS 、アイデンティティまたはリソースに関連付けられているときにアクセス許可を定義する のオブジェクトです。 は、プリンシパル (ユーザー、ルートユーザー、またはロールセッション) がリクエストを行うときに、これらのポリシー AWS を評価します。ポリシーでの権限により、リクエストが許可されるか拒否されるかが決まります。ほとんどのポリシーは JSON ドキュメント AWS として に保存されます。JSON ポリシードキュメントの構造と内容の詳細については、IAM ユーザーガイドの JSON ポリシー概要を参照してください。

管理者は JSON AWS ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソースに対してどのような条件下でアクションを実行できるかということです。

デフォルトでは、ユーザーやロールに権限はありません。IAM 管理者は、リソースで必要なアクションを実行するための権限をユーザーに付与する IAM ポリシーを作成できます。その後、管理者はロールに IAM ポリシーを追加し、ユーザーはロールを引き受けることができます。

IAM ポリシーは、オペレーションの実行方法を問わず、アクションの許可を定義します。例えば、`iam:GetRole` アクションを許可するポリシーがあるとします。そのポリシーを持つユーザーは、 AWS Management Console、、 AWS CLIまたは AWS API からロール情報を取得できます。

アイデンティティベースのポリシー

アイデンティティベースポリシーは、IAM ユーザーグループ、ユーザーのグループ、ロールなど、アイデンティティにアタッチできる JSON 許可ポリシードキュメントです。これらのポリシーは、ユーザーとロールが実行できるアクション、リソース、および条件をコントロールします。アイデンティティベースポリシーの作成方法については、「IAM ユーザーガイド」の「カスタマー管理ポリシーでカスタム IAM アクセス許可を定義する」を参照してください。

アイデンティティベースのポリシーは、さらにインラインポリシーまたはマネージドポリシーに分類できます。インラインポリシーは、単一のユーザー、グループ、またはロールに直接埋め込まれています。管理ポリシーは、 内の複数のユーザー、グループ、ロールにアタッチできるスタンドアロンポリシーです AWS アカウント。管理ポリシーには、 AWS 管理ポリシーとカスタマー管理ポリシーが含まれます。マネージドポリシーまたはインラインポリシーのいずれかを選択する方法については、「IAM ユーザーガイド」の「管理ポリシーとインラインポリシーのいずれかを選択する」を参照してください。

リソースベースのポリシー

リソースベースのポリシーは、リソースに添付する JSON ポリシードキュメントです。リソースベースのポリシーには例として、IAM ロールの信頼ポリシーや Amazon S3 バケットポリシーがあげられます。リソースベースのポリシーをサポートするサービスでは、サービス管理者はポリシーを使用して特定のリソースへのアクセスを制御できます。ポリシーがアタッチされているリソースの場合、指定されたプリンシパルがそのリソースに対して実行できるアクションと条件は、ポリシーによって定義されます。リソースベースのポリシーでは、プリンシパルを指定する必要があります。プリンシパルには、アカウント、ユーザー、ロール、フェデレーティッドユーザー、または を含めることができます AWS のサービス。

リソースベースのポリシーは、そのサービス内にあるインラインポリシーです。リソースベースのポリシーでは、IAM の AWS マネージドポリシーを使用できません。

アクセスコントロールリスト (ACL)

アクセスコントロールリスト (ACL) は、どのプリンシパル (アカウントメンバー、ユーザー、または ロール) がリソースにアクセスするための許可を持つかを制御します。ACL はリソースベースのポリシーに似ていますが、JSON ポリシードキュメント形式は使用しません。

Amazon S3、 AWS WAF、および Amazon VPC は、ACLs。ACL の詳細については、「Amazon Simple Storage Service デベロッパーガイド」の「アクセスコントロールリスト (ACL) の概要」を参照してください。

その他のポリシータイプ

AWS は、追加のあまり一般的ではないポリシータイプをサポートしています。これらのポリシータイプでは、より一般的なポリシータイプで付与された最大の権限を設定できます。

- アクセス許可の境界 - アクセス許可の境界は、アイデンティティベースポリシーによって IAM エンティティ (IAM ユーザーまたはロール) に付与できる権限の上限を設定する高度な機能です。エンティティにアクセス許可の境界を設定できます。結果として得られる権限は、エンティティのアイデンティティベースポリシーとそのアクセス許可の境界の共通部分になります。`Principal` フィールドでユーザーまたはロールを指定するリソースベースのポリシーでは、アクセス許可の境界は制限されません。これらのポリシーのいずれかを明示的に拒否した場合、権限は無効になります。アクセス許可の境界の詳細については、「IAM ユーザーガイド」の「IAM エンティティのアクセス許可の境界」を参照してください。
- サービスコントロールポリシー (SCPs) – SCPsは、 の組織または組織単位 (OU) の最大アクセス許可を指定する JSON ポリシーです AWS Organizations。 AWS Organizations は、ビジネスが所有する複数の をグループ化して一元管理するためのサービス AWS アカウント です。組織内のすべての機能を有効にすると、サービスコントロールポリシー (SCP) を一部またはすべてのアカウントに適用できます。SCP は、各 を含むメンバーアカウントのエンティティのアクセス許可を制限します AWS アカウントのルートユーザー。Organizations と SCP の詳細については、「AWS Organizations ユーザーガイド」の「サービスコントロールポリシー (SCP)」を参照してください。
- リソースコントロールポリシー (RCP) – RCP は、所有する各リソースにアタッチされた IAM ポリシーを更新することなく、アカウント内のリソースに利用可能な最大数のアクセス許可を設定するために使用できる JSON ポリシーです。RCP は、メンバーアカウントのリソースのアクセス許可を制限し、組織に属しているかどうかにかかわらず AWS アカウントのルートユーザー、 を含む ID の有効なアクセス許可に影響を与える可能性があります。RCP をサポートする のリストなど、Organizations と RCP の詳細については、AWS Organizations 「 ユーザーガイド AWS のサービス 」の「リソースコントロールポリシー (RCPs」を参照してください。 RCPs

- セッションポリシー - セッションポリシーは、ロールまたはフェデレーションユーザーの一時的なセッションをプログラムで作成する際にパラメータとして渡す高度なポリシーです。結果としてセッションの権限は、ユーザーまたはロールのアイデンティティベースポリシーとセッションポリシーの共通部分になります。また、リソースベースのポリシーから権限が派生する場合もあります。これらのポリシーのいずれかを明示的に拒否した場合、権限は無効になります。詳細については、「IAM ユーザーガイド」の「セッションポリシー」を参照してください。

複数のポリシータイプ

1 つのリクエストに複数のタイプのポリシーが適用されると、結果として作成される権限を理解するのがさらに難しくなります。複数のポリシータイプが関係している場合にリクエストを許可するかどうか AWS を決定する方法については、IAM ユーザーガイドの「ポリシー評価ロジック」を参照してください。

AWS マネージド統合の マネージドポリシー

ユーザー、グループ、ロールにアクセス許可を追加するには、自分でポリシーを記述するよりも、AWS 管理ポリシーを使用する方が簡単です。チームに必要な権限のみを提供する IAM カスタマーマネージドポリシーを作成するには時間と専門知識が必要です。すぐに開始するには、 AWS マネージドポリシーを使用できます。これらのポリシーは、一般的なユースケースをターゲット範囲に含めており、 AWS アカウントで利用できます。 AWS 管理ポリシーの詳細については、IAM ユーザーガイドの「 AWS 管理ポリシー」を参照してください。

AWS サービスは、 AWS 管理ポリシーを維持および更新します。 AWS 管理ポリシーのアクセス許可は変更できません。サービスでは新しい機能を利用できるようにするために、 AWS マネージドポリシーに権限が追加されることがあります。この種類の更新はポリシーがアタッチされている、すべてのアイデンティティ (ユーザー、グループおよびロール) に影響を与えます。新しい機能が立ち上げられた場合や、新しいオペレーションが使用可能になった場合に、各サービスが AWS マネージドポリシーを更新する可能性が最も高くなります。サービスは AWS マネージドポリシーからアクセス許可を削除しないため、ポリシーの更新によって既存のアクセス許可が破損することはありません。

さらに、 は、複数の サービスにまたがるジョブ関数の マネージドポリシー AWS をサポートします。例えば、ReadOnlyAccess AWS 管理ポリシーは、すべての AWS サービスとリソースへの読み取り専用アクセスを提供します。サービスが新機能を起動すると、 は新しいオペレーションとリソースの読み取り専用アクセス許可 AWS を追加します。ジョブ機能のポリシーの一覧および詳細に

ついては、「IAM ユーザーガイド」の「AWS のジョブ機能のマネージドポリシー」を参照してください。

AWS マネージドポリシー: AWSIoTManagedIntegrationsFullAccess

`AWSIoTManagedIntegrationsFullAccess` ポリシーを IAM アイデンティティにアタッチできます。

このポリシーは、マネージド統合および関連サービスへのフルアクセス許可を付与します。でこのポリシーを表示するには AWS Management Console、「AWSIoTManagedIntegrationsFullAccess」を参照してください。

アクセス許可の詳細

このポリシーには、以下のアクセス許可が含まれています。

- `iotmanagedintegrations` – このポリシーを追加する IAM ユーザー、グループ、ロールのマネージド統合および関連サービスへのフルアクセスを提供します。
- `iam` – 割り当てられた IAM ユーザー、グループ、ロールが でサービスにリンクされたロールを作成できるようにします AWS アカウント。

```
{
  "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 マネージドポリシー： AWS IoTManagedIntegrationsRolePolicy

`AWS IoTManagedIntegrationsRolePolicy` ポリシーを IAM アイデンティティにアタッチできます。

このポリシーは、ユーザーに代わって Amazon CloudWatch logsとメトリクスを発行するアクセス許可をマネージド統合に付与します。

でこのポリシーを表示するには AWS Management Console、「AWSIoTManagedIntegrationsRolePolicy」を参照してください。

アクセス許可の詳細

このポリシーには、以下のアクセス許可が含まれています。

- `logs` – Amazon CloudWatch ロググループを作成し、グループにログをストリーミングする機能を提供します。
- `cloudwatch` – Amazon CloudWatch メトリクスを発行する機能を提供します。Amazon CloudWatch メトリクスの詳細については、Amazon CloudWatch のメトリクス」を参照してください。

```
{
  "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"
          ]
        }
      }
    }
  ]
}
```

マネージドポリシーへの AWS マネージド統合の更新

このサービスがこれらの変更の追跡を開始してからの AWS 、 マネージド統合の マネージドポリシーの更新に関する詳細を表示します。このページの変更に関する自動アラートについては、 マネージドインテグレーションドキュメント履歴ページの RSS フィードにサブスクライブしてください。

変更	説明	日付
マネージド統合が変更の追跡を開始	マネージド統合は、 AWS マネージドポリシーの変更の追跡を開始しました。	2025 年 3 月 3 日

マネージド統合と IAM の連携方法

IAM を使用して マネージド統合へのアクセスを管理する前に、 マネージド統合で使用できる IAM 機能について説明します。

マネージド統合で使用できる IAM 機能

IAM 機能	マネージド統合のサポート
アイデンティティベースポリシー	はい
リソースベースのポリシー	いいえ
ポリシーアクション	はい
ポリシーリソース	あり
ポリシー条件キー	Yes
ACL	いいえ
ABAC (ポリシー内のタグ)	いいえ
一時的な認証情報	はい
プリンシパル権限	はい
サービスロール	はい
サービスリンクロール	はい

マネージド統合およびその他の AWS のサービスがほとんどの IAM 機能と連携する方法の概要については、IAM ユーザーガイドのAWS 「IAM と連携する のサービス」を参照してください。

マネージド統合のアイデンティティベースのポリシー

アイデンティティベースのポリシーのサポート: あり

アイデンティティベースポリシーは、IAM ユーザーグループ、ユーザーのグループ、ロールなど、アイデンティティにアタッチできる JSON 許可ポリシードキュメントです。これらのポリシーは、ユーザーとロールが実行できるアクション、リソース、および条件をコントロールします。ID ベースのポリシーの作成方法については、「IAM ユーザーガイド」の「カスタマー管理ポリシーでカスタム IAM アクセス許可を定義する」を参照してください。

IAM アイデンティティベースのポリシーでは、許可または拒否するアクションとリソース、およびアクションを許可または拒否する条件を指定できます。プリンシパルは、それが添付されているユーザーまたはロールに適用されるため、アイデンティティベースのポリシーでは指定できません。JSON ポリシーで使用できるすべての要素について学ぶには、「IAM ユーザーガイド」の「IAM JSON ポリシーの要素のリファレンス」を参照してください。

マネージド統合のアイデンティティベースのポリシーの例

マネージド統合のアイデンティティベースのポリシーの例を表示するには、「」を参照してください マネージド統合のアイデンティティベースのポリシーの例。

マネージド統合内のリソースベースのポリシー

リソースベースのポリシーのサポート: なし

リソースベースのポリシーは、リソースに添付する JSON ポリシードキュメントです。リソースベースのポリシーには例として、IAM ロールの信頼ポリシーや Amazon S3 バケットポリシーがあげられます。リソースベースのポリシーをサポートするサービスでは、サービス管理者はポリシーを使用して特定のリソースへのアクセスを制御できます。ポリシーがアタッチされているリソースの場合、指定されたプリンシパルがそのリソースに対して実行できるアクションと条件は、ポリシーによって定義されます。リソースベースのポリシーでは、プリンシパルを指定する必要があります。プリンシパルには、アカウント、ユーザー、ロール、フェデレーティッドユーザー、または を含めることができます AWS のサービス。

クロスアカウントアクセスを有効にするには、アカウント全体、または別のアカウントの IAM エンティティをリソースベースのポリシーのプリンシパルとして指定します。リソースベースのポリシーにクロスアカウントのプリンシパルを追加しても、信頼関係は半分しか確立されない点に注意してください。プリンシパルとリソースが異なる場合 AWS アカウント、信頼されたアカウントの IAM 管理者は、プリンシパルエンティティ (ユーザーまたはロール) にリソースへのアクセス許可も付与す

る必要があります。IAM 管理者は、アイデンティティベースのポリシーをエンティティにアタッチすることで権限を付与します。ただし、リソースベースのポリシーで、同じアカウントのプリンシパルへのアクセス権が付与されている場合は、アイデンティティベースのポリシーをさらに付与する必要はありません。詳細については、「IAM ユーザーガイド」の「IAM でのクロスアカウントリソースアクセス」を参照してください。

マネージド統合のポリシーアクション

ポリシーアクションのサポート:あり

管理者は JSON AWS ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソースに対してどのような条件下でアクションを実行できるかということです。

JSON ポリシーの `Action` 要素にはポリシー内のアクセスを許可または拒否するために使用できるアクションが記述されます。ポリシーアクションの名前は通常、関連付けられた AWS API オペレーションと同じです。一致する API オペレーションのない許可のみのアクションなど、いくつかの例外があります。また、ポリシーに複数のアクションが必要なオペレーションもあります。これらの追加アクションは依存アクションと呼ばれます。

このアクションは関連付けられたオペレーションを実行するためのアクセス許可を付与するポリシーで使用されます。

マネージド統合アクションのリストを確認するには、「サービス認可リファレンス」の「マネージド統合で定義されるアクション」を参照してください。

マネージド統合のポリシーアクションは、アクションの前に次のプレフィックスを使用します。

```
iot-mi
```

単一のステートメントで複数のアクションを指定するには、アクションをカンマで区切ります。

```
"Action": [
      "iot-mi:action1",
      "iot-mi:action2"
         ]
```

マネージド統合のアイデンティティベースのポリシーの例を表示するには、「」を参照してください マネージド統合のアイデンティティベースのポリシーの例。

マネージド統合のポリシーリソース

ポリシーリソースのサポート: あり

管理者は JSON AWS ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソースに対してどのような条件下でアクションを実行できるかということです。

Resource JSON ポリシー要素はアクションが適用されるオブジェクトを指定します。ステートメントにはResource または NotResource 要素を含める必要があります。ベストプラクティスとして、アマゾン リソースネーム (ARN) を使用してリソースを指定します。これは、リソースレベルの許可と呼ばれる特定のリソースタイプをサポートするアクションに対して実行できます。

オペレーションのリスト化など、リソースレベルの権限をサポートしないアクションの場合は、ステートメントがすべてのリソースに適用されることを示すために、ワイルドカード (*) を使用します。

```
"Resource": "*"
```

マネージド統合リソースタイプとその ARNs「マネージド統合で定義されるリソース」を参照してください。 各リソースの ARN を指定できるアクションについては、「 マネージド統合で定義されるアクション」を参照してください。

マネージド統合のアイデンティティベースのポリシーの例を表示するには、「」を参照してくださいマネージド統合のアイデンティティベースのポリシーの例。

マネージド統合のポリシー条件キー

サービス固有のポリシー条件キーのサポート: あり

管理者は JSON AWS ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソースに対してどのような条件下でアクションを実行できるかということです。

Condition 要素 (または Condition ブロック) を使用すると、ステートメントが有効な条件を指定できます。Condition 要素はオプションです。イコールや未満などの 条件演算子 を使用して条件式を作成して、ポリシーの条件とリクエスト内の値を一致させることができます。

1 つのステートメントに複数の Condition 要素を指定する場合、または 1 つの Condition 要素に複数のキーを指定する場合、 AWS では AND 論理演算子を使用してそれらを評価します。1 つの条

件キーに複数の値を指定すると、 は論理ORオペレーションを使用して条件 AWS を評価します。ステートメントの権限が付与される前にすべての条件が満たされる必要があります。

条件を指定する際にプレースホルダー変数も使用できます。例えば IAM ユーザーに、IAM ユーザー名がタグ付けされている場合のみリソースにアクセスできる権限を付与することができます。詳細については、「IAM ユーザーガイド」の「IAM ポリシーの要素: 変数およびタグ」を参照してください。

AWS は、グローバル条件キーとサービス固有の条件キーをサポートしています。すべての AWS グローバル条件キーを確認するには、IAM ユーザーガイドのAWS 「グローバル条件コンテキストキー」を参照してください。

マネージド統合条件キーのリストを確認するには、「サービス認可リファレンス」の「マネージド統合の条件キー」を参照してください。条件キーを使用できるアクションとリソースについては、「 マネージド統合で定義されるアクション」を参照してください。

マネージド統合のアイデンティティベースのポリシーの例を表示するには、「」を参照してくださいマネージド統合のアイデンティティベースのポリシーの例。

マネージド統合ACLs

ACL のサポート: なし

アクセスコントロールリスト (ACL) は、どのプリンシパル (アカウントメンバー、ユーザー、またはロール) がリソースにアクセスするための許可を持つかを制御します。ACL はリソースベースのポリシーに似ていますが、JSON ポリシードキュメント形式は使用しません。

マネージド統合による ABAC

ABAC (ポリシー内のタグ) のサポート: 一部

属性ベースのアクセス制御 (ABAC) は、属性に基づいてアクセス許可を定義する認可戦略です。では AWS、これらの属性はタグと呼ばれます。タグは、IAM エンティティ (ユーザーまたはロール) および多くの AWS リソースにアタッチできます。エンティティとリソースのタグ付けは、ABAC の最初の手順です。その後、プリンシパルのタグがアクセスしようとしているリソースのタグと一致した場合にオペレーションを許可するように ABAC ポリシーをします。

ABAC は、急成長する環境やポリシー管理が煩雑になる状況で役立ちます。

タグに基づいてアクセスを管理するには、`aws:ResourceTag/key-name`、`aws:RequestTag/key-name`、または `aws:TagKeys` の条件キーを使用して、ポリシーの条件要素でタグ情報を提供します。

サービスがすべてのリソースタイプに対して 3 つの条件キーすべてをサポートする場合、そのサービスの値はありです。サービスが一部のリソースタイプに対してのみ 3 つの条件キーのすべてをサポートする場合、値は「部分的」になります。

ABAC の詳細については、「IAM ユーザーガイド」の「ABAC 認可でアクセス許可を定義する」を参照してください。ABAC をセットアップする手順を説明するチュートリアルについては、「IAM ユーザーガイド」の「属性ベースのアクセスコントロール (ABAC) を使用する」を参照してください。

マネージド統合での一時的な認証情報の使用

一時的な認証情報のサポート: あり

一部の AWS のサービス は、一時的な認証情報を使用してサインインすると機能しません。一時的な認証情報 AWS のサービス を使用する方法などの詳細については、IAM ユーザーガイドAWS のサービス の「IAM と連携する 」を参照してください。

ユーザー名とパスワード以外の方法 AWS Management Console を使用して にサインインする場合、一時的な認証情報を使用します。たとえば、会社のシングルサインオン (SSO) リンク AWS を使用して にアクセスすると、そのプロセスによって一時的な認証情報が自動的に作成されます。また、ユーザーとしてコンソールにサインインしてからロールを切り替える場合も、一時的な認証情報が自動的に作成されます。ロールの切り替えに関する詳細については、「IAM ユーザーガイド」の「ユーザーから IAM ロールに切り替える (コンソール)」を参照してください。

一時的な認証情報は、 AWS CLI または AWS API を使用して手動で作成できます。その後、これらの一時的な認証情報を使用してアクセスすることができます AWS。長期的なアクセスキーを使用する代わりに、一時的な認証情報を動的に生成 AWS することをお勧めします。詳細については、「IAM の一時的セキュリティ認証情報」を参照してください。

マネージド統合のクロスサービスプリンシパルアクセス許可

転送アクセスセッション (FAS) のサポート: あり

IAM ユーザーまたはロールを使用して でアクションを実行すると AWS、プリンシパルと見なされます。一部のサービスを使用する際に、アクションを実行することで、別のサービスの別のアクションがトリガーされることがあります。FAS は、 を呼び出すプリンシパルのアクセス許可と AWS のサービス、ダウンストリームサービス AWS のサービス へのリクエストをリクエストする を使用します。FAS リクエストは、サービスが他の AWS のサービス またはリソースとのやり取りを完了する必要があるリクエストを受け取った場合にのみ行われます。この場合、両方のアクションを実行す

るためのアクセス許可が必要です。FAS リクエストを行う際のポリシーの詳細については、「転送アクセスセッション」を参照してください。

マネージド統合のサービスロール

サービスロールのサポート: あり

サービスロールとは、サービスがユーザーに代わってアクションを実行するために引き受ける IAM ロールです。IAM 管理者は、IAM 内からサービスロールを作成、変更、削除できます。詳細については、「IAM ユーザーガイド」の「AWS のサービスに許可を委任するロールを作成する」を参照してください。

> **Warning**
> サービスロールのアクセス許可を変更すると、マネージド統合機能が破損する可能性があります。マネージド統合が指示する場合にのみ、サービスロールを編集します。

マネージド統合のサービスにリンクされたロール

サービスリンクロールのサポート: あり

サービスにリンクされたロールは、 にリンクされたサービスロールの一種です AWS のサービス。サービスは、ユーザーに代わってアクションを実行するロールを引き受けることができます。サービスにリンクされたロールは に表示され AWS アカウント 、 サービスによって所有されます。IAM 管理者は、サービスにリンクされたロールのアクセス許可を表示できますが、編集することはできません。

サービスにリンクされたロールの作成または管理の詳細については、「IAM と提携するAWS のサービス」を参照してください。表の「サービスリンクロール」列に Yes と記載されたサービスを見つけます。サービスにリンクされたロールに関するドキュメントをサービスで表示するには、[はい] リンクを選択します。

マネージド統合のアイデンティティベースのポリシーの例

デフォルトでは、ユーザーとロールには、マネージド統合リソースを作成または変更するアクセス許可はありません。また、、 AWS Command Line Interface （ AWS CLI ） AWS Management Console、または AWS API を使用してタスクを実行することはできません。IAM 管理者は、リソースで必要なアクションを実行するための権限をユーザーに付与する IAM ポリシーを作成できます。その後、管理者はロールに IAM ポリシーを追加し、ユーザーはロールを引き継ぐことができます。

これらサンプルの JSON ポリシードキュメントを使用して、IAM アイデンティティベースのポリシーを作成する方法については、「IAM ユーザーガイド」の「IAM ポリシーを作成する (コンソール)」を参照してください。

各リソースタイプの ARNs「サービス認可リファレンス」の「マネージド統合のアクション、リソース、および条件キー」を参照してください。

トピック

- ポリシーに関するベストプラクティス
- マネージド統合コンソールの使用
- 自分の権限の表示をユーザーに許可する

ポリシーに関するベストプラクティス

ID ベースのポリシーは、アカウント内で誰かがマネージド統合リソースを作成、アクセス、または削除できるかどうかを決定します。これらのアクションを実行すると、 AWS アカウントに料金が発生する可能性があります。アイデンティティベースポリシーを作成したり編集したりする際には、以下のガイドラインと推奨事項に従ってください:

- AWS 管理ポリシーの使用を開始し、最小特権のアクセス許可に移行する – ユーザーとワークロードにアクセス許可の付与を開始するには、多くの一般的なユースケースにアクセス許可を付与するAWS 管理ポリシーを使用します。これらは で使用できます AWS アカウント。ユースケースに固有の AWS カスタマー管理ポリシーを定義することで、アクセス許可をさらに減らすことをお勧めします。詳細については、「IAM ユーザーガイド」の「AWS マネージドポリシー」または「ジョブ機能のAWS マネージドポリシー」を参照してください。
- 最小特権を適用する – IAM ポリシーで許可を設定する場合は、タスクの実行に必要な許可のみを付与します。これを行うには、特定の条件下で特定のリソースに対して実行できるアクションを定義します。これは、最小特権アクセス許可とも呼ばれています。IAM を使用して許可を適用する方法の詳細については、「IAM ユーザーガイド」の「IAM でのポリシーとアクセス許可」を参照してください。
- IAM ポリシーで条件を使用してアクセスをさらに制限する - ポリシーに条件を追加して、アクションやリソースへのアクセスを制限できます。例えば、ポリシー条件を記述して、すべてのリクエストを SSL を使用して送信するように指定できます。条件を使用して、サービスアクションが などの特定の を通じて使用されている場合に AWS のサービス、サービスアクションへのアクセスを許可することもできます AWS CloudFormation。詳細については、「IAM ユーザーガイド」の「IAM JSON ポリシー要素:条件」を参照してください。

- IAM Access Analyzer を使用して IAM ポリシーを検証し、安全で機能的な権限を確保する - IAM Access Analyzer は、新規および既存のポリシーを検証して、ポリシーが IAM ポリシー言語 (JSON) および IAM のベストプラクティスに準拠するようにします。IAM アクセスアナライザーは 100 を超えるポリシーチェックと実用的な推奨事項を提供し、安全で機能的なポリシーの作成をサポートします。詳細については、「IAM ユーザーガイド」の「IAM Access Analyzer でポリシーを検証する」を参照してください。
- 多要素認証 (MFA) を要求する – で IAM ユーザーまたはルートユーザーを必要とするシナリオがある場合は AWS アカウント、セキュリティを強化するために MFA を有効にします。API オペレーションが呼び出されるときに MFA を必須にするには、ポリシーに MFA 条件を追加します。詳細については、「IAM ユーザーガイド」の「MFA を使用した安全な API アクセス」を参照してください。

IAM でのベストプラクティスの詳細については、「IAM ユーザーガイド」の「IAM でのセキュリティのベストプラクティス」を参照してください。

マネージド統合コンソールの使用

マネージド統合コンソールにアクセスするには、最小限のアクセス許可のセットが必要です。これらのアクセス許可により、 のマネージド統合リソースの詳細を一覧表示および表示できます AWS アカウント。最小限必要な許可よりも制限が厳しいアイデンティティベースのポリシーを作成すると、そのポリシーを持つエンティティ (ユーザーまたはロール) に対してコンソールが意図したとおりに機能しません。

AWS CLI または AWS API のみを呼び出すユーザーには、最小限のコンソールアクセス許可を付与する必要はありません。代わりに、実行しようとしている API オペレーションに一致するアクションのみへのアクセスが許可されます。

ユーザーとロールが引き続きマネージド統合コンソールを使用できるようにするには、エンティティにマネージド統合*ConsoleAccess*または*ReadOnly* AWS マネージドポリシーもアタッチします。詳細については、「IAM ユーザーガイド」の「ユーザーへのアクセス許可の追加」を参照してください。

自分の権限の表示をユーザーに許可する

この例では、ユーザーアイデンティティにアタッチされたインラインおよびマネージドポリシーの表示を IAM ユーザーに許可するポリシーの作成方法を示します。このポリシーには、コンソールで、または AWS CLI または AWS API を使用してプログラムでこのアクションを実行するアクセス許可が含まれています。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

マネージド統合のアイデンティティとアクセスのトラブルシューティング

次の情報は、マネージド統合と IAM の使用時に発生する可能性がある一般的な問題の診断と修正に役立ちます。

トピック

- マネージド統合でアクションを実行する権限がない

- iam:PassRole を実行する権限がありません
- 自分の 以外のユーザーに マネージド統合リソース AWS アカウント へのアクセスを許可したい

マネージド統合でアクションを実行する権限がない

アクションを実行する権限がないというエラーが表示された場合は、そのアクションを実行できるようにポリシーを更新する必要があります。

次のエラー例は、mateojackson IAM ユーザーがコンソールを使用して、ある *my-example-widget* リソースに関する詳細情報を表示しようとしたことを想定して、その際に必要なiot-mi:*GetWidget* アクセス許可を持っていない場合に発生するものです。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform: iot-mi:GetWidget on resource: my-example-widget
```

この場合、iot-mi:*GetWidget* アクションを使用して *my-example-widget*リソースへのアクセスを許可するように、mateojackson ユーザーのポリシーを更新する必要があります。

サポートが必要な場合は、 AWS 管理者にお問い合わせください。サインイン認証情報を提供した担当者が管理者です。

iam:PassRole を実行する権限がありません

iam:PassRole アクションを実行する権限がないというエラーが表示された場合は、マネージド統合にロールを渡すことができるようにポリシーを更新する必要があります。

一部の AWS のサービス では、新しいサービスロールまたはサービスにリンクされたロールを作成する代わりに、既存のロールをそのサービスに渡すことができます。そのためには、サービスにロールを渡す権限が必要です。

次の例のエラーは、 という名前の IAM marymajor ユーザーがコンソールを使用してマネージド統合でアクションを実行しようとすると発生します。ただし、このアクションをサービスが実行するには、サービスロールから付与された権限が必要です。メアリーには、ロールをサービスに渡す許可がありません。

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

この場合、Mary のポリシーを更新してメアリーに iam:PassRole アクションの実行を許可する必要があります。

サポートが必要な場合は、 AWS 管理者にお問い合わせください。サインイン資格情報を提供した担当者が管理者です。

自分の 以外のユーザーに マネージド統合リソース AWS アカウント へのアクセスを許可したい

他のアカウントのユーザーや組織外の人が、リソースにアクセスするために使用できるロールを作成できます。ロールの引き受けを委託するユーザーを指定できます。リソースベースのポリシーまたはアクセスコントロールリスト (ACL) をサポートするサービスの場合、それらのポリシーを使用して、リソースへのアクセスを付与できます。

詳細については、以下を参照してください:

- マネージド統合がこれらの機能をサポートしているかどうかを確認するには、「」を参照してください[マネージド統合と IAM の連携方法](#)。
- 所有 AWS アカウント する のリソースへのアクセスを提供する方法については、IAM ユーザーガイドの「所有 AWS アカウント する別の の IAM ユーザーへのアクセスを提供する」を参照してください。
- リソースへのアクセスをサードパーティーに提供する方法については AWS アカウント、IAM ユーザーガイドの「サードパーティー AWS アカウント が所有する へのアクセスを提供する」を参照してください。
- ID フェデレーションを介してアクセスを提供する方法については、「IAM ユーザーガイド」の「外部で認証されたユーザー (ID フェデレーション) へのアクセスの許可」を参照してください。
- クロスアカウントアクセスにおけるロールとリソースベースのポリシーの使用方法の違いについては、「IAM ユーザーガイド」の「IAM でのクロスアカウントのリソースへのアクセス」を参照してください。

マネージド統合でサービスにリンクされたロールを使用する

AWS IoT Device Management のマネージド統合では、 AWS Identity and Access Management (IAM) サービスにリンクされたロールを使用します。サービスにリンクされたロールは、マネージド統合に直接リンクされた一意のタイプの IAM ロールです。サービスにリンクされたロールは、マネージド統合によって事前定義されており、サービスがユーザーに代わって他の AWS サービスを呼び出すために必要なすべてのアクセス許可が含まれています。

サービスにリンクされたロールを使用すると、必要なアクセス許可を手動で追加する必要がなくなるため、マネージド統合の設定が簡単になります。 AWS IoT Device Management のマネージド統合

は、サービスにリンクされたロールのアクセス許可を定義します。特に定義されている場合を除き、マネージド統合のみがそのロールを引き受けることができます。定義される許可は信頼ポリシーと許可ポリシーに含まれており、その許可ポリシーを他の IAM エンティティにアタッチすることはできません。

サービスリンクロールを削除するには、最初に関連リソースを削除する必要があります。これにより、リソースへのアクセス許可が誤って削除されないため、マネージド統合リソースが保護されます。

サービスにリンクされたロールをサポートする他のサービスの詳細については、AWS「IAM と連携するサービス」を参照し、「サービスにリンクされたロール」列で「はい」があるサービスを探します。サービスリンクロールに関するドキュメントをサービスで表示するには、リンクで [はい] を選択します。

マネージド統合のサービスにリンクされたロールのアクセス許可

AWS IoT Device Management のマネージド統合は、AWSServiceRoleForIoTManagedIntegrations という名前のサービスにリンクされたロールを使用します。ユーザーに代わってログとメトリクスを発行する AWS IoT Device Management のマネージド統合アクセス許可を提供します。

AWSServiceRoleForIoTManagedIntegrations サービスにリンクされたロールは、次のサービスを信頼してロールを引き受けます。

- `iotmanagedintegrations.amazonaws.com`

AWSIoTManagedIntegrationsServiceRolePolicy という名前のロールアクセス許可ポリシーにより、マネージド統合は指定されたリソースに対して次のアクションを実行できます。

- アクション: `logs:CreateLogGroup, logs:DescribeLogGroups, logs:CreateLogStream, logs:PutLogEvents, logs:DescribeLogStreams, cloudwatch:PutMetricData` on `all of your managed integrations resources.`

```
{
  "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"
          ]
        }
      }
    }
  ]
}
```

ユーザー、グループ、またはロールにサービスリンクロールの作成、編集、または削除を許可するには、アクセス許可を設定する必要があります。詳細についてはIAM ユーザーガイド の「サービスにリンクされた役割のアクセス許可」を参照してください。

マネージド統合のサービスにリンクされたロールの作成

サービスリンクロールを手動で作成する必要はありません。、`CreateEventLogConfiguration`、または `RegisterCustomEndpoint` API で `PutRuntimeLogConfiguration`、 AWS Management Console、AWS CLIまたは AWS API コマンドを呼び出すなどのイベントタイプが発生すると、マネージド統合によってサービスにリンクされたロールが作成されます。`PutRuntimeLogConfiguration`、、`CreateEventLogConfiguration`または の詳細については`RegisterCustomEndpoint`、PutRuntimeLogConfiguration、CreateEventLogConfiguration、または RegisterCustomEndpoint を参照してください。

このサービスリンクロールを削除した後で再度作成する必要が生じた場合は同じ方法でアカウントにロールを再作成できます。`PutRuntimeLogConfiguration`、、`CreateEventLogConfiguration`または `RegisterCustomEndpoint` API コマンドの呼び出しなどのイベントタイプが発生すると、マネージド統合によってサービスにリンクされたロールが再度作成されます。または、 経由で AWS カスタマーサポートに連絡することもできます AWS Support Center Console。 AWS サポートプランの詳細については、「AWS サポートプランの比較」を参照してください。

IAM コンソールを使用して、 IoT ManagedIntegrations - マネージドロールのユースケースでサービスにリンクされたロールを作成することもできます。 AWS CLI または AWS API で、サービス名を使用して`iotmanagedintegrations.amazonaws.com`サービスにリンクされたロールを作成します。詳細については、「IAM ユーザーガイド」の「サービスリンクロールの作成」を参照してください。このサービスリンクロールを削除しても、同じ方法でロールを再作成できます。

マネージド統合のサービスにリンクされたロールの編集

マネージド統合では、AWSServiceRoleForIoTManagedIntegrations サービスにリンクされたロールを編集することはできません。サービスリンクロールの作成後は、さまざまなエンティティがロールを参照する可能性があるため、ロール名を変更することはできません。ただし、IAM を使用してロールの説明を編集することはできます。詳細については、「IAM ユーザーガイド」の「サービスリンクロールの編集」を参照してください。

マネージド統合のサービスにリンクされたロールの削除

サービスリンクロールを必要とする機能やサービスが不要になった場合は、ロールを削除することをお勧めします。そうすることで、積極的にモニタリングまたは保守されていない未使用のエンティ

ティを排除できます。ただし、手動で削除する前に、サービスリンクロールのリソースをクリーンアップする必要があります。

> **Note**
>
> マネージド統合がリソースを削除しようとしたときに ロールを使用している場合、削除が失敗する可能性があります。その場合は、数分待ってからオペレーションを再試行してください。

IAM を使用してサービスリンクロールを手動で削除するには

IAM コンソール、 AWS CLI、または AWS API を使用して、AWSServiceRoleForIoTManagedIntegrations サービスにリンクされたロールを削除します。詳細については、「IAM ユーザーガイド」の「サービスにリンクされたロールの削除」を参照してください。

マネージド統合サービスにリンクされたロールでサポートされているリージョン

AWS IoT Device Management のマネージド統合は、サービスが利用可能なすべてのリージョンでサービスにリンクされたロールの使用をサポートします。詳細については、「AWS リージョンとエンドポイント」を参照してください。

マネージド統合のコンプライアンス検証

AWS のサービス が特定のコンプライアンスプログラムの範囲内にあるかどうかを確認するには、AWS のサービス 「コンプライアンスプログラムによる対象範囲内」を参照して、関心のあるコンプライアンスプログラムを選択します。一般的な情報については、AWS 「コンプライアンスプログラム」を参照してください。

を使用して、サードパーティーの監査レポートをダウンロードできます AWS Artifact。詳細については、「Downloading Reports in AWS Artifact」を参照してください。

を使用する際のお客様のコンプライアンス責任 AWS のサービス は、お客様のデータの機密性、貴社のコンプライアンス目的、適用される法律および規制によって決まります。 は、コンプライアンスに役立つ以下のリソース AWS を提供します。

- セキュリティのコンプライアンスとガバナンス – これらのソリューション実装ガイドでは、アーキテクチャ上の考慮事項について説明し、セキュリティとコンプライアンスの機能をデプロイする手順を示します。

- HIPAA 対応サービスのリファレンス – HIPAA 対応サービスの一覧が提供されています。すべて AWS のサービス HIPAA の対象となるわけではありません。
- AWS コンプライアンスリソース – このワークブックとガイドのコレクションは、お客様の業界や地域に適用される場合があります。
- AWS カスタマーコンプライアンスガイド – コンプライアンスの観点から責任共有モデルを理解します。このガイドでは、複数のフレームワーク (米国国立標準技術研究所 (NIST)、Payment Card Industry Security Standards Council (PCI)、国際標準化機構 (ISO) など) にわたるセキュリティコントロールを保護し、そのガイダンスに AWS のサービス マッピングするためのベストプラクティスをまとめています。
- 「 デベロッパーガイド」の「ルールによるリソースの評価」 – この AWS Config サービスは、リソース設定が内部プラクティス、業界ガイドライン、および規制にどの程度準拠しているかを評価します。 AWS Config
- AWS Security Hub – これにより AWS のサービス 、 内のセキュリティ状態を包括的に把握できます AWS。Security Hub では、セキュリティコントロールを使用して AWS リソースを評価し、セキュリティ業界標準とベストプラクティスに対するコンプライアンスをチェックします。サポートされているサービスとコントロールの一覧については、Security Hub のコントロールリファレンスを参照してください。
- Amazon GuardDuty – 不審なアクティビティや悪意のあるアクティビティがないか環境をモニタリングすることで AWS アカウント、、ワークロード、コンテナ、データに対する潜在的な脅威 AWS のサービス を検出します。GuardDuty を使用すると、特定のコンプライアンスフレームワークで義務付けられている侵入検知要件を満たすことで、PCI DSS などのさまざまなコンプライアンス要件に対応できます。
- AWS Audit Manager – これにより AWS のサービス 、 AWS 使用状況を継続的に監査し、リスクの管理方法と規制や業界標準への準拠を簡素化できます。

マネージド統合の耐障害性

AWS グローバルインフラストラクチャは、 AWS リージョン およびアベイラビリティーゾーンを中心に構築されています。 は、低レイテンシー、高スループット、および高度に冗長なネットワークで接続された、物理的に分離および分離された複数のアベイラビリティーゾーン AWS リージョン を提供します。アベイラビリティーゾーンでは、ゾーン間で中断することなく自動的にフェイルオーバーするアプリケーションとデータベースを設計および運用することができます。アベイラビリティーゾーンは、従来の単一または複数のデータセンターインフラストラクチャよりも可用性が高く、フォールトトレラントで、スケーラブルです。

AWS リージョン およびアベイラビリティーゾーンの詳細については、AWS 「 グローバルインフラストラクチャ」を参照してください。

AWS グローバルインフラストラクチャに加えて、 のマネージド統合 AWS IoT Device Management には、データの耐障害性とバックアップのニーズをサポートするのに役立ついくつかの機能があります。

マネージド統合のモニタリング

モニタリングは、マネージド統合やその他の AWS ソリューションの信頼性、可用性、パフォーマンスを維持する上で重要な部分です。AWS には、マネージド統合を監視し、問題が発生したときに報告し、必要に応じて自動アクションを実行するための以下のモニタリングツールが用意されています。

- Amazon CloudWatch は、 AWS リソースと で実行されるアプリケーションを AWS リアルタイムでモニタリングします。メトリクスの収集と追跡、カスタマイズしたダッシュボードの作成、および指定したメトリクスが指定したしきい値に達したときに通知またはアクションを実行するアラームの設定を行うことができます。例えば、CloudWatch で Amazon EC2 インスタンスの CPU 使用率などのメトリクスを追跡し、必要に応じて新しいインスタンスを自動的に起動できます。詳細については、「Amazon CloudWatch ユーザーガイド」を参照してください。
- Amazon CloudWatch Logs では、Amazon EC2 インスタンス、CloudTrail、その他ソースから得たログファイルのモニタリング、保存、およびアクセスが可能です。CloudWatch Logs は、ログファイル内の情報をモニタリングし、特定のしきい値が満たされたときに通知します。高い耐久性を備えたストレージにログデータをアーカイブすることも可能です。詳細については、「Amazon CloudWatch Logs ユーザーガイド」を参照してください。
- Amazon EventBridge を使用して AWS サービスを自動化し、アプリケーションの可用性の問題やリソースの変更などのシステムイベントに自動的に対応できます。 AWS サービスからのイベントは、ほぼリアルタイムで EventBridge に配信されます。簡単なルールを記述して、注目するイベントと、イベントがルールに一致した場合に自動的に実行するアクションを指定できます。詳細については、「Amazon EventBridge ユーザーガイド」を参照してください。
- AWS CloudTrail は、 AWS アカウントによって、またはアカウントに代わって行われた API コールおよび関連イベントをキャプチャし、指定した Amazon S3 バケットにログファイルを配信します。が呼び出したユーザーとアカウント AWS、呼び出し元のソース IP アドレス、および呼び出しの発生日時を特定できます。詳細については、「AWS CloudTrail ユーザーガイド」を参照してください。

Amazon CloudWatch とのマネージド統合のモニタリング

CloudWatch を使用してCloudWatch は raw データを収集し、読み取り可能なほぼリアルタイムのメトリクスに加工します。これらの統計は 15 か月間保持されるため、履歴情報にアクセスし、ウェブアプリケーションまたはサービスの動作をより的確に把握できます。また、特定のしきい値を監視す

るアラームを設定し、これらのしきい値に達したときに通知を送信したりアクションを実行したりできます。詳細については、「Amazon CloudWatch ユーザーガイド」を参照してください。

マネージド型統合の場合、*XXX* を監視し、XXX を監視し、*#########*は*##########*することもできます。

次の表に、 マネージド統合のメトリクスとディメンションを示します。

Amazon EventBridge での Managed Integrations イベントのモニタリング

EventBridge で Managed Integrations イベントをモニタリングできます。これにより、独自のアプリケーション、software-as-a-service (SaaS) アプリケーション、および AWS のサービスからリアルタイムのデータのストリームが提供されます。EventBridge は、そのデータを AWS Lambda や Amazon Simple Notification Service などのターゲットにルーティングします。これらのイベントは Amazon CloudWatch Events に表示されるイベントと同じです。Amazon CloudWatch Events は、AWS リソースの変更を記述するシステムイベントのほぼリアルタイムのストリームを提供します。

次の例は、 マネージド統合のイベントを示しています。

トピック

- eventName イベント

eventName イベント

このイベントの例では、

```
{
   "version": "0",
   "id": "01234567-EXAMPLE",
   "detail-type": "ServiceName ResourceType State Change",
   "source": "aws.servicename",
   "account": "123456789012",
   "time": "2019-06-12T10:23:43Z",
   "region": "us-east-2",
   "resources": [
     "arn:aws:servicename:us-east-2:123456789012:resourcename"
   ],
```

```
    "detail": {
      "event": "eventName",
      "detailOne": "something",
      "detailTwo": "12345678-1234-5678-abcd-12345678abcd",
      "detailThree": "something",
      "detailFour": "something"
    }
}
```

を使用した Managed Integrations API コールのログ記録 AWS CloudTrail

マネージド統合は、ユーザーAWS CloudTrail、ロール、または によって実行されたアクションを記録するサービスである と統合されています AWS のサービス。CloudTrail は、マネージド統合のすべての API コールをイベントとしてキャプチャします。キャプチャされた呼び出しには、マネージド統合コンソールからの呼び出しと、マネージド統合 API オペレーションへのコード呼び出しが含まれます。CloudTrail で収集された情報を使用して、マネージド統合に対するリクエスト、リクエスト元の IP アドレス、リクエスト日時などの詳細を確認できます。

各イベントまたはログエントリには、誰がリクエストを生成したかという情報が含まれます。アイデンティティ情報は、以下を判別するのに役立ちます。

- ルートユーザーまたはユーザー認証情報のどちらを使用してリクエストが送信されたか。
- リクエストが IAM Identity Center ユーザーに代わって行われたかどうか。
- リクエストがロールまたはフェデレーションユーザーのテンポラリなセキュリティ認証情報を使用して行われたかどうか。
- リクエストが、別の AWS のサービスによって送信されたかどうか。

CloudTrail は、アカウント AWS アカウント を作成すると でアクティブになり、CloudTrail イベント履歴に自動的にアクセスできます。CloudTrail の [イベント履歴] では、 AWS リージョンで過去 90 日間に記録された 管理イベントの表示、検索、およびダウンロードが可能で、変更不可能な記録を確認できます。詳細については、「AWS CloudTrail ユーザーガイド」の「CloudTrail イベント履歴の使用」を参照してください。[イベント履歴] の閲覧には CloudTrail の料金はかかりません。

AWS アカウント 過去 90 日間のイベントの継続的な記録については、証跡または CloudTrail Lake イベントデータストアを作成します。

CloudTrail 証跡

証跡により、CloudTrail はログファイルを Amazon S3 バケットに配信できます。を使用して作成された証跡はすべてマルチリージョン AWS Management Console です。 AWS CLIを使用する際は、単一リージョンまたは複数リージョンの証跡を作成できます。アカウント AWS リージョン 内のすべての でアクティビティをキャプチャするため、マルチリージョン証跡を作成することをお勧めします。単一リージョンの証跡を作成する場合、証跡の AWS リージョンに記録されたイベントのみを表示できます。証跡の詳細については、「AWS CloudTrail ユーザーガイド」の「AWS アカウントの証跡の作成」および「組織の証跡の作成」を参照してください。

証跡を作成すると、進行中の管理イベントのコピーを 1 つ無料で CloudTrail から Amazon S3 バケットに配信できますが、Amazon S3 ストレージには料金がかかります。CloudTrail の料金の詳細については、「AWS CloudTrail の料金」を参照してください。Amazon S3 の料金に関する詳細については、「Amazon S3 の料金」を参照してください。

CloudTrail Lake イベントデータストア

[CloudTrail Lake] を使用すると、イベントに対して SQL ベースのクエリを実行できます。CloudTrail Lake は、行ベースの JSON 形式の既存のイベントを Apache ORC 形式に変換します。ORC は、データを高速に取得するために最適化された単票ストレージ形式です。イベントは、イベントデータストアに集約されます。イベントデータストアは、高度なイベントセレクタを適用することによって選択する条件に基づいた、イベントのイミュータブルなコレクションです。どのイベントが存続し、クエリに使用できるかは、イベントデータストアに適用するセレクタが制御します。CloudTrail Lake の詳細については、 AWS CloudTrail ユーザーガイドのAWS CloudTrail 「Lake の使用」を参照してください。

CloudTrail Lake のイベントデータストアとクエリにはコストがかかります。イベントデータストアを作成する際に、イベントデータストアに使用する料金オプションを選択します。料金オプションによって、イベントの取り込みと保存にかかる料金、および、そのイベントデータストアのデフォルトと最長の保持期間が決まります。CloudTrail の料金の詳細については、「AWS CloudTrail の料金」を参照してください。

CloudTrail でイベントを管理する

管理イベントは、 のリソースで実行される管理オペレーションに関する情報を提供します AWS アカウント。これらのイベントは、コントロールプレーンオペレーションとも呼ばれます。CloudTrail は、デフォルトで管理イベントをログ記録します。

マネージド統合は、すべてのマネージド統合コントロールプレーンオペレーションを管理イベントとしてログに記録します。統合ログを CloudTrail に管理する Managed Integrations コントロールプレーンオペレーションのリストについては、「Managed integrations API Reference」を参照してください。

イベント例

各イベントは任意の送信元からの単一のリクエストを表し、リクエストされた API オペレーション、オペレーションの日時、リクエストパラメータなどに関する情報を含みます。CloudTrail ログファイルは、パブリック API コールの順序付けられたスタックトレースではないため、イベントは特定の順序で表示されません。

次の例は、 `StartDeviceDiscovery` API オペレーションを示す CloudTrail イベントを示しています。

`StartDeviceDiscovery` API オペレーションで成功した CloudTrail イベント。

```
{
    "eventVersion": "1.09",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROA47CRX4JX4AEXAMPLE",
        "arn": "arn:aws:sts::123456789012:assumed-role/Admin/EXAMPLE",
        "accountId": "222222222222",
        "accessKeyId": "access-key-id",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROA47CRX4JXUNEXAMPLE",
                "arn": "arn:aws:iam::123456789012:role/Admin",
                "accountId": "222222222222",
                "userName": "Admin"
            },
            "attributes": {
                "creationDate": "2025-02-26T20:04:25Z",
                "mfaAuthenticated": "false"
            }
        }
    },
    "eventTime": "2025-02-26T20:11:33Z",
    "eventSource": "gamma-iotmanagedintegrations.amazonaws.com",
```

```
    "eventName": "StartDeviceDiscovery",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "15.248.7.123",
    "userAgent": "aws-sdk-java/2.30.21 md/io#sync md/http#Apache ua/2.1 os/
Mac_OS_X#15.3.1 lang/java#17.0.13 md/OpenJDK_64-Bit_Server_VM#17.0.13+11-LTS md/
vendor#Amazon.com_Inc. md/en_US cfg/auth-source#stat m/D,N",
    "requestParameters": {
        "DiscoveryType": "ZIGBEE",
        "ControllerIdentifier": "554a1e3f7c884e67a21e0cabac3a48e3"
    },
    "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": "717023e159264ec5ba97293e4d884d3a",
        "StartedAt": 1740600693.789,
        "Arn": "arn:aws:iotmanagedintegrations::123456789012:device-
discovery/717023e159264ec5ba97293e4d884d3a"
    },
    "requestID": "29aa09b9-ad0e-42dc-8b7f-565a1a56c020",
    "eventID": "d8d0a6ab-b729-4aa5-8af0-9f605ee90d0f",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "123456789012",
    "eventCategory": "Management"
}
```


`StartDeviceDiscovery` API オペレーションを使用して拒否された CloudTrail イベントにアクセスします。


```
{
    "eventVersion": "1.09",
    "userIdentity": {
        "type": "AssumedRole",
```

```
        "principalId": "AROA47CRX4JX4AEXAMPLE",
        "arn": "arn:aws:sts::123456789012:assumaDMINed-role/EXAMPLEExplicitDenyRole/
EXAMPLE",
        "accountId": "222222222222",
        "accessKeyId": "access-key-id",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROA47CRX4JXUNEXAMPLE",
                "arn": "arn:aws:iam::123456789012:role/EXAMPLEExplicitDenyRole",
                "accountId": "222222222222",
                "userName": "EXAMPLEExplicitDenyRole"
            },
            "attributes": {
                "creationDate": "2025-02-27T21:36:55Z",
                "mfaAuthenticated": "false"
            }
        },
        "invokedBy": "AWS Internal"
    },
    "eventTime": "2025-02-27T21:37:01Z",
    "eventSource": "gamma-iotmanagedintegrations.amazonaws.com",
    "eventName": "StartDeviceDiscovery",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "AWS Internal",
    "userAgent": "AWS Internal",
    "errorCode": "AccessDenied",
    "requestParameters": {
        "DiscoveryType": "CLOUD",
        "ClientToken": "ClientToken",
        "ConnectorAssociationIdentifier": "ConnectorAssociation"
    },
    "responseElements": {
        "message": "User: arn:aws:sts::123456789012:assumed-role/
EXAMPLEExplicitDenyRole/EXAMPLE is not authorized to perform:
 iotmanagedintegrations:StartDeviceDiscovery on resource:
 arn:aws:iotmanagedintegrations:us-east-1:123456789012:/device-discoveries with an
 explicit deny"
    },
    "requestID": "5eabd798-d79c-4d76-a5dd-115be230d77a",
    "eventID": "cc75660c-f628-462a-9e6e-83dab40c5246",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "managementEvent": true,
```

```
    "recipientAccountId": "123456789012",
    "eventCategory": "Management"
}
```

CloudTrail レコードの内容については、「AWS CloudTrail ユーザーガイド」の「CloudTrail record contents」を参照してください。

マネージド統合デベロッパーガイドのドキュメント履歴

次の表に、 マネージド統合のドキュメントリリースを示します。

変更	説明	日付
初回リリース	マネージド統合デベロッパーガイドの初回リリース	2025 年 3 月 3 日