aws

デベロッパーガイド

のマネージド統合 AWS IoT Device Management

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

Table of Contents

マネージド統合とは AWS IoT Device Management

のマネージド統合 AWS IoT Device Management は、IoT ソリューションプロバイダーが何百もの製造元の IoT デバイスの制御と管理を統一するのに役立ちます。マネージド統合を使用すると、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。マネージド型統合では、単一のユーザーインターフェイスと一連の APIs 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

サポート対象のリージョン

のマネージド統合 AWS IoT Device Management は、次のリージョンでサポートされています。

- カナダ (中部)
- 欧州 (アイルランド)

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。

- マネージド統合をセットアップする
- の マネージド統合の使用を開始する AWS IoT Device Management

Managed integrations overview

次の図は、マネージド統合の概要を示しています。

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に、これらの主要な概念と用語の概要を説明します。

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、モノと比較したマネージド AWS IoT Core 型モノです。

- 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。
- Managed integrations Managed Thing: Managed Thing では、デバイスとのやり取り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。

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提供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のためにこれらのデバイスを制御でき、マネージド統合は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。

データモデルの用語

マネージド型統合では、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の通信を行います。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: 複数のノードが連携して完全な機能セットを提供する物理デバイス (ビデオドアベルなど) を表すエンティティ。
- 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、ポーチライトがオン、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。

マネージド統合をセットアップする

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

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、電話またはテキストメッセージを受け取り、電話キーパッドで検証コードを入力します。

   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、 を保護し AWS IAM アイデンティティセンター、 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 マネジメントコンソールとして 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 アイデンティティセンター ユーザーガイド」の「AWS IAM アイデンティティセンターの有効化」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 アイデンティティセンター 「 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 アイデンティティセンター 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 アイデンティティセンター ユーザーガイド」の「Add groups」を参照してください。

の マネージド統合の使用を開始する 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手順の概要を説明します。

トピック

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。各デバイスは、次の 3 つのカテゴリのいずれかに該当します。

- 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のエンドデバイス 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 (C2C) デバイス: 独自のクラウドインフラストラクチャとデバイス制御用のブランドモバイルアプリケーションを維持するベンダーによって設計および管理されるデバイスです。マネージド統合のお客様は、構築済みの C2C コネクタのカタログにアクセスしたり、独自の C2C コネクタを作成したりして、統合インターフェイスを介して複数のサードパーティーベンダークラウドと連携する IoT ソリューションを開発できます。

  エンドユーザーが初めて C2C デバイスの電源をオンにするときは、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が完了すると、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使用できます。

暗号化キーを設定する

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。

Managed integrations API リファレンスガイドの 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、GetDefaultEncryptionConfiguration 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。

オンボーディング手法

オンボーディングのタイプを以下に示します。

直接接続されたデバイスのオンボーディング

直接接続されたデバイスをオンボードする手順プロビジョニング先については、「」を参照してください。

ハブオンボーディング

ハブをオンボードする手順マネージド統合にハブをオンボードするについては、「」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

ハブ接続デバイスをオンボードする手順デバイスをオンボードし、ハブで運用するについては、「」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

サードパーティーのクラウドベン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ドする手順C2C (Cloud-to-Cloud) コネクタを使用するについては、「」を参照してください。

デバイスプロビジョニング

デバイスプロビジョニングは、デバイスの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、デバイスソフトウェア開発キット (SDK) またはハブデバイスを介して間接的にリンクされた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managedThingId。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、マネージド統合のマネージドモノにアタッチされます。マネージドモノは、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managedThingId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。

プロビジョニングされるデバイスのステータスは、オンボーディングフローのステージによって異なります。次のリストでは、各プロビジョニングステータスについて説明します。

- アクティブ: デバイスが検出され、コマンドとコントロールが使用可能になりました。
- DISCOVERED: デバイスは見つかりましたが、コマンドとコントロールはまだ利用できません。
- UNASSOCIATED: マネージドモノが作成されましたが、さらにアクションを検出する必要があります。 AWS クラウド または AWS IoT Managed Integrations Controller (ハブ) からアクセスできない
- PRE_ASSOCIATED: マネージド型モノが作成され、電源が入っているか接続されると自動検出の準備が整います。 AWS クラウド または AWS IoT Managed Integrations Controller (ハブ) からアクセスすることはできません。
- DELETE_IN_PROGRESS: 非同期削除プロセスが開始されました。
- DELETED: デバイスは から削除されました AWS クラウド。
- ISOLATED: 以前に検出またはアクティブ化されたマネージドモノのうち、到達できなくなったもの。たとえば、コネクタの関連付けがすべて削除されたサードパーティークラウドのデバイスなどです。

次のオンボーディングフローは、 マネージド統合を使用してハブをプロビジョニングするためのものです。

マネージド統合にハブをオンボードする: デバイス認証、通信、セットアップを処理するために連携するコアプロビジョナーとプロトコル固有のプラグインをセットアップします。

マネージド統合を使用してハブ接続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): デバイスはサプライチェーンのアップストリームで事前に関連付けられています。たとえば、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て、デバイスをカスタマーアカウントに事前リンクします。
- 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。

> **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合理化された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の安全性を確保し、効率的に実行できます。

トピック

デバイス

最初のオンボーディング中、マネージド統合はマネージドモノと呼ばれる物理デバイスのデジタルツインを作成します。Managed Thing には、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managedThingIDを提供する があります。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、 などのマネージドモノのパブリック 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**
>
> 物理デバイスは、異なる顧客の下で複数回プロビジョニングされる場合、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での Managed Thing の作成から始まり、お客様が DeleteManagedThing API を使用して Managed Thing 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には、 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が保存されます。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およびプロビジョニング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 AWS 実装。類似点がありますが、以下のトピックで説明する微妙な違いもあります。

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 の機能を活用します。

トピック

マネージド統合データモデル

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" },
    "capabilities": Capability[]
}
```

Capability

はデバイスの機能capabilityを表します。

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

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

AWS Matter データモデルの実装

Matter Data Model の AWS 実装は、マネージド型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

デバイスの説明には、 endpointと の 2 つのデータ要素が使用されます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",
        ...
    }
    ```

データモデルスキーマ

マネージド統合は、機能定義とタイプ定義の 2 つのスキーマタイプをサポートします。カスタムデータモデルを作成する場合は、JSON スキーマドキュメントを使用していずれかのタイプのスキーマを定義します。各スキーマドキュメントには 50,000 文字の制限があります。
    ```

機能スキーマ

機能は、エンドポイント内の特定の機能を表す基本的な構成要素です。機能を使用すると、プロパティ、アクション、イベントを使用してデバイスの状態と動作をモデル化できます。プロパティを使用すると、任意の宣言データ型でデバイスの状態属性を柔軟にモデル化できます。アクションとイベントは、実行できるコマンドやレポートできるシグナルなど、デバイスの動作をモデル化します。

以下は、機能スキーマの高レベルの構造を示しています。

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

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。たとえば、ベルを鳴らしたり、ドアにいる人を見たりします。

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、ドアのノックなどのセンサーからのインシデント、アラーム、またはアクティビティを報告するイベントを送信できます。

プロパティ

デバイスの状態の特定の属性を表すエンティティ。たとえば、ベルが鳴っている、またはポークライトがオンになっているなどです。

各機能には、一意の名前空間識別子、バージョン情報、およびその目的の説明が含まれます。スキーマドキュメントでは、セマンティックバージョニングを使用して下位互換性を維持し、新機能を有効にします。

詳細については、「機能定義のスキーマ」を参照してください。

型定義スキーマ

型定義は、再利用性と合成性を可能にする宣言型の構造化データ型です。情報のフォーマット方法と制約方法を定義します。タイプ定義を使用して、IoT ソリューション全体で標準化されたデータ形式を作成します。

各タイプ定義には以下が含まれます。

- 一意の名前空間識別子
- タイトル
- 説明
- データ形式と制約を定義するプロパティ

型は、整数や制限が定義された文字列などの単純なプリミティブでも、列挙型や複数のフィールドを持つカスタムオブジェクトなどの複雑な構造でもかまいません。タイプ定義では、JSON スキーマ構文を使用して、最小値と最大値、文字列の長さ、許容パターンなどの制約を指定します。

詳細については、「タイプ定義のスキーマ」を参照してください。

機能定義のスキーマ

機能は、システム内で機能する方法について明確な契約を提供する宣言型 JSON ドキュメントを使用して文書化されます。

機能の場合、必須要素は $id、、extrinsicId、extrinsicVersionおよび以下のセクションの少なくとも 1 nameつの要素です。

- properties
- actions
- events

機能のオプションの要素は、$ref、title、、descriptionversion、$defsおよび ですextrinsicProperties。機能については、$ref「」を参照してくださいaws.capability。

以下のセクションでは、機能定義に使用されるスキーマについて詳しく説明します。

$id (必須)

$id 要素はスキーマ定義を識別します。次の構造に従う必要があります。

- URI /schema-versions/ プレフィックスで始める
- capability スキーマタイプを含める
- URI パス区切り文字としてスラッシュ (/) を使用する
- フラグメントをピリオドで区切ってスキーマ ID を含める (.)

- @ 文字を使用してスキーマ ID とバージョンを区切る
- ピリオド (.) を使用してバージョンフラグメントを分離し、セムバーバージョンで終わる

スキーマ ID は、3 ~ 12 文字のルート名前空間で始まり、その後にオプションのサブ名前空間と名前が続く必要があります。

セムバーバージョンには、メジャーバージョン (最大 3 桁) 、マイナーバージョン (最大 3 桁) 、オプションの PATCH バージョン (最大 4 桁) が含まれます。

> **Note**
>
> 予約された名前空間aws や matter

Example例 $id

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

$ref

$ref 要素は、システム内の既存の機能を参照します。これは、 $id要素と同じ制約に従います。

> **Note**
>
> タイプ定義または機能は、 $ref ファイルで指定された値で存在する必要があります。

Example$ref の例

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

名前 (必須)

name 要素は、スキーマドキュメント内のエンティティ名を表す文字列です。多くの場合、略語が含まれているため、次のルールに従う必要があります。

- 英数字、ピリオド (.)、スラッシュ (/)、ハイフン (-)、スペースのみを含む
- 文字で始める

- 最大 64 文字

名前要素は、Amazon Web Services コンソールの UI とドキュメントで使用されます。

Example名前の例

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

title

title 要素は、スキーマドキュメントで表されるエンティティの記述文字列です。任意の文字を含めることができ、ドキュメントで使用されます。機能タイトルの最大長は 256 文字です。

Exampleタイトルの例

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

description

description 要素は、スキーマドキュメントで表されるエンティティの詳細な説明を提供します。任意の文字を含めることができ、ドキュメントで使用されます。機能の説明の最大長は 2048 文字です

Example説明の例

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

version

version 要素はオプションです。これは、スキーマドキュメントのバージョンを表す文字列です。以下の制約があります。

- セムバー形式を使用し、次のバージョンフラグメントを . (ピリオド) で区切ります。
  - MAJOR バージョン、最大 3 桁
  - MINOR バージョン、最大 3 桁
  - PATCH バージョン (オプション) 、最大 4 桁
- 長さは 3 ~ 12 文字です。

Exampleバージョン例

```
1.0
```

```
1.12
```

```
1.4.1
```

機能バージョンの使用

機能は、イミュータブルなバージョニングされたエンティティです。すべての変更は、新しいバージョンを作成することが期待されます。システムは、MAJOR.MINOR.PATCH 形式のセマンティックバージョニングを使用します。ここで、

- 下位互換性のない API 変更を行う場合のメジャーバージョンの増加
- 下位互換性のある方法で機能を追加するとマイナーバージョンが増加する
- 機能に影響のない軽微な追加を行うと、パッチバージョンが増加します。

Matter クラスターから派生した機能はバージョン 1.4 からベースライン化され、各 Matter リリースはシステムにインポートされる予定です。Matter バージョンはセムバーのメジャーレベルとマイナーレベルの両方を消費するため、マネージド統合では PATCH バージョンのみを使用できます。

Matter に PATCH バージョンを追加するときは、 Matter がシーケンシャルリビジョンを使用することに注意してください。すべての PATCH バージョンは Matter 仕様に記載されているリビジョンに準拠し、下位互換性がある必要があります。

下位互換性のない問題を修正するには、Connection Standards Alliance (CSA) と協力して仕様の問題を解決し、新しいリビジョンをリリースする必要があります。

AWSマネージド機能は、 の初期バージョンでリリースされました`1.0`。これらを使用すると、3 つのレベルのバージョンをすべて使用できます。

extrinsicVersion (必須)

これは、 AWS IoT システムの外部で管理されるバージョンを表す文字列です。Matter 機能の場合、は にextrinsicVersionマッピングされます。 revision

これは文字列化された整数値として表され、長さは 1 ~ 10 桁の数字です。

Exampleバージョン例

```
7
```

```
1567
```

extrinsicId (必須)

extrinsicId 要素は、Amazon Web Services IoT システムの外部で管理される識別子を表します。Matter 機能の場合、コンテキストに応じて 、fieldId、、clusterIdattributeIdcommandIdeventId、または にマッピングされます。

は、文字列化された 10 進数整数 (1 ~ 10 桁) または文字列化された 16 進数整数 (0x または 0X プレフィックス、1 ~ 8 桁の 16 進数) のいずれかextrinsicIdです。

> **Note**
>
> AWSベンダー ID (VID) は 0x1577 で、Matter の場合は 0 です。システムは、カスタムスキーマがこれらの予約済み VIDs を機能に使用しないようにします。

Example extrinsicIds の例

```
0018
0x001A
0x15771002
```

$defs

$defs セクションは、JSON スキーマで許可されているスキーマドキュメント内で参照できるサブスキーマのマップです。このマップでは、 キーはローカル参照定義で使用され、 値は JSON スキーマを提供します。

> **Note**
>
> システムは、有効なマップ`$defs`であり、各サブスキーマが有効な JSON スキーマである のみを適用します。追加のルールは適用されません。

定義を使用する場合は、以下の制約事項に従ってください。

- 定義名には URI フレンドリ文字のみを使用する
- 各値が有効なサブスキーマであることを確認します。
- スキーマドキュメントのサイズ制限内に収まる任意の数のサブスキーマを含める

extrinsicProperties

`extrinsicProperties` 要素には、外部システムで定義されているが、データモデル内で維持されている一連のプロパティが含まれています。Matter 機能の場合、ZCL クラスター、属性、コマンド、またはイベント内のさまざまなモデル化されていない要素または部分的にモデル化された要素にマッピングされます。

外部プロパティは、次の制約に従う必要があります。

- プロパティ名はスペースや特殊文字を含まない英数字にする必要があります
- プロパティ値は任意の JSON スキーマ値にすることができます
- 最大 20 個のプロパティ

システムは、`extrinsicProperties`、、、 など`cliFunctionName`、さまざまな `access` `apiMaturity` `cli`をサポートしています。これらのプロパティにより、データモデル変換への AWS （ またはその逆の) ACL が容易になります。

> **Note**
>
> 外部プロパティは、機能の `action`、`event`、`property`、および `struct`フィールド要素でサポートされていますが、機能やクラスター自体ではサポートされていません。

システムサポートの外部プロパティ

システムは、ZCL との間の変換extrinsicProperties中に、以下のモデル化されていない、または部分的にモデル化されたクラスター、属性、コマンド、またはイベント属性を追跡します。

access

各アクセスオブジェクトには、以下が含まれています。

- op - 値 read、、writeまたは enumを持つ としてモデル化されたオペレーション invoke
- privilege - 、view、proxy_view、operate、manageまたは の値enumを持つ としてモデル化された特権 administer
- role - 演算子ロールを表す無制限の文字列

apiMaturity

成熟度レベルを表す無制限のプレーン文字列。これは、、stable、、internalまたは の値enumを持つ として ZCL provisionalでモデル化されます。 deprecated

side

、either、serverおよび の値を持つ列挙型としてモデル化 client

ブールプロパティ

以下のプロパティはブールフラグです。

- isFabricScoped
- isFabricSensitive
- mustUseAtomicWrite
- mustUseTimedInvoke

文字列プロパティ

次のプロパティは、無制限の文字列として表されます。

- cli
- cliFunctionName
- functionName
- group
- introducedIn

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

変換に関する考慮事項

ZCL 変換の場合、 extrinsicPropertiesは処理せずにマップに保存されます。検出を使用するカスタムスキーマは ZCL 変換を受けません。ただし、今後カスタムスキーマに ZCL 変換を実装する場合は、すべての無制限のプレーン文字列タイプをモデルextrinsicProperties化し、列挙型、パターン (正規表現) 、長さなどの制約を定義する必要があります。この準備により、変換中にこれらのプロパティを適切に処理できます。

対照的に、 コネクタ変換の AWS については、コネクタ形式ではこれらの詳細が必須ではないため、 extrinsicPropertiesはまったく含まれていません。

プロパティ

プロパティは、 機能のデバイス管理の状態を表します。各状態はキーと値のペアとして定義され、キーは状態の名前を記述し、値は状態の定義を記述します。

プロパティを使用する場合は、次の制約事項に従ってください。

- プロパティ名には英数字のみを使用し、スペースや特殊文字は使用しない
- スキーマドキュメントのサイズ制限内に収まる任意の数のプロパティを含める

プロパティの使用

機能内のプロパティは、マネージド統合を使用するデバイスの特定の状態を表す基本的な要素です。デバイスの現在の状態または設定を表します。これらのプロパティの定義と構造を標準化すること

で、スマートホームシステムはさまざまなメーカーのデバイスが効果的に通信できるようにし、シームレスで相互運用可能なエクスペリエンスを実現します。

機能プロパティの場合、必須要素は extrinsicIdと ですvalue。機能プロパティのオプション要素は、description、retrievable、mutable、reportableおよび ですextrinsicProperties。

値

ビルダーが JSON スキーマ準拠の制約を適用して、このプロパティのデータ型を定義できるようにする無制限の構造。

値を定義するときは、次の制約に従います。

- シンプルな型の場合は、 type と、 maxLengthや などのその他のネイティブ JSON スキーマ制約を使用します。 maximum
- 複合型の場合は、oneOf、allOf、または を使用しますanyOf。システムは notキーワードをサポートしていません
- 任意のグローバルタイプを参照するには、有効な検出可能なリファレンス$refで を使用します。
- null 可能性については、nullable 属性にブールフラグを指定して OpenAPI タイプのスキーマ定義に従います (truenull が許容値である場合) 。

例:

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

取得可能

状態が読み取り可能かどうかを説明するブール値。

状態の可読性の側面は、デバイスの 機能の実装に委ねられます。デバイスは、特定の状態が読み取り可能かどうかを決定します。状態のこの側面は、機能レポートではまだ報告されないため、システム内で強制されません。

例: true または false

Mutable

状態が書き込み可能かどうかを説明するブール値。

状態の書き込み可能性の側面は、デバイスの 機能の実装に委ねられます。デバイスは、特定の状態が書き込み可能かどうかを決定します。状態のこの側面は、機能レポートではまだ報告されないため、システム内で強制されません。

例: `true` または `false`

報告対象

状態に変化があったときに 状態がデバイスによって報告されるかどうかを説明するブール値。

状態のレポート可能性の側面は、デバイスの 機能の実装に委ねられます。デバイスは、特定の状態が報告可能かどうかを決定します。状態のこの側面は、機能レポートではまだ報告されないため、システム内で強制されません。

例: `true` または `false`

アクション

アクションは、リクエスト/レスポンスモデルに従うスキーマ管理オペレーションです。各アクションは、デバイス実装オペレーションを表します。

アクションを実装するときは、次の制約に従ってください。

- アクション配列に一意のアクションのみを含める
- スキーマドキュメントのサイズ制限内に収まる任意の数のアクションを含める

アクションの使用

アクションは、マネージド統合システムでデバイス機能とやり取りして制御するための標準化された方法です。これは、デバイスで実行できる特定のコマンドまたはオペレーションを表し、必要なリクエストまたはレスポンスパラメータをモデル化するための構造化された形式で補完されます。これらのアクションは、ユーザーの意図とデバイスオペレーションの橋渡しとして機能し、さまざまなタイプのスマートデバイス間で一貫した信頼性の高い制御を可能にします。

アクションの場合、必須要素は `name`と ですextrinsicId。オプションの要素は、`description`、`extrinsicProperties`、`request`および ですresponse。

説明

説明の最大長は 1536 文字です。

リクエスト

リクエストセクションはオプションであり、リクエストパラメータがない場合は省略できます。省略すると、システムは の名前を使用するだけでペイロードなしでリクエストを送信できますAction。これは、照明のオン/オフなどの簡単なアクションで使用されます。

複雑なアクションには、追加のパラメータが必要です。たとえば、カメラの映像をストリーミングするリクエストには、使用するストリーミングプロトコルに関するパラメータや、ストリームを特定のディスプレイデバイスに送信するかどうかに関するパラメータが含まれる場合があります。

アクションリクエストの場合、必須要素は ですparameters。オプションの要素は、description、extrinsicId、および ですextrinsicProperties。

リクエストの説明

説明はセクション 3.5 と同じ形式で、最大長は 2048 文字です。

レスポンス

マネージド統合では、SendManagedThingCommand API を介して送信されたアクションリクエストについて、リクエストはデバイスに到達し、非同期レスポンスが返されることを期待します。アクションレスポンスは、このレスポンスの構造を定義します。

アクションリクエストの場合、必須要素は ですparameters。オプションの要素は、name、description、extrinsicId、extrinsicProperties、errorsおよび ですresponseCode。

レスポンスの説明

説明は と同じ形式に従いdescription、最大長は 2048 文字です。

レスポンス名

名前は と同じ形式に従います。詳細については名前 (必須)、以下を参照してください。

- レスポンスの従来の名前は、アクション名Responseに を追加することによって算出されます。
- 別の名前を使用する場合は、このname要素で指定できます。レスポンスで nameが指定されている場合、この値は従来の名前よりも優先されます。

エラー

リクエストの処理中にエラーが発生した場合にレスポンスで提供される一意のメッセージの無制限の配列。

制約:

- メッセージ項目は、次のフィールドを持つ JSON オブジェクトとして宣言されます。
  - code: 英数字と _ (アンダースコア) を含み、長さが 1～64 文字の文字列
  - message: 無制限の文字列値

Exampleエラーメッセージの例

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

Response Code (レスポンスコード)

リクエストの処理方法を示す整数コード。デバイスコードは、HTTP サーバーのレスポンスステータスコード仕様を使用してコードを返し、システム内での統一を可能にすることをお勧めします。

制約: 100～599 の範囲の整数値。

リクエストまたはレスポンスパラメータ

パラメータセクションは、名前とサブスキーマのペアのマップとして定義されます。スキーマドキュメントに収まる場合、任意の数のパラメータをリクエストパラメータ内で定義できます。

パラメータ名に使用できるのは英数字のみです。スペースやその他の文字は使用できません。

パラメータフィールド

の必須要素は extrinsicIdと parameterですvalue。オプションの要素は descriptionおよびですextrinsicProperties。

description 要素は と同じ形式に従いdescription、最大長は 1024 文字です。

extrinsicId および **extrinsicProperties**オーバーライド

extrinsicId および は、 extrinsicId (必須) および と同じ形式extrinsicPropertiesに従います。詳細についてはextrinsicProperties、以下を参照してください。

- リクエストまたはレスポンスで extrinsicIdが指定されている場合、この値はアクションレベルで提供される値よりも優先されます。システムがextrinsicId最初にリクエスト/レスポンスレベルを使用する必要があります。見つからない場合はアクションレベルを使用してください。extrinsicId
- リクエストまたはレスポンスで extrinsicPropertiesが指定されている場合、これらのプロパティはアクションレベルで提供される va 値よりも優先されます。システムはアクションレベルを取得しextrinsicProperties、リクエスト/レスポンスレベルで提供されるキーと値のペアを置き換える必要があります extrinsicProperties

Example extrinsicId および extrinsicProperties オーバーライドの例

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

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

```
   }
}
```

上記の例では、アクションリクエストの有効な値は次のようになります。

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

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

組み込みアクション

すべての機能で、キーワード ReadStateと を使用してカスタムアクションを実行できますUpdateState。これら 2 つのアクションキーワードは、データモデルで定義された機能のプロパティに作用します。

ReadState

状態プロパティの値を読み取るmanagedThingコマンドを に送信します。デバイスの状態を強制的に更新する方法ReadStateとして を使用します。

UpdateState

一部のプロパティを更新するコマンドを送信します。

デバイス状態の同期を強制することは、以下のシナリオで役立ちます。

1. デバイスは一定期間オフラインであり、イベントを出力していませんでした。
2. デバイスはプロビジョニングされたばかりで、クラウドにまだ状態が維持されていません。
3. デバイスの状態がデバイスの実際の状態と同期していません。

ReadState の例

SendManagedThingCommand API を使用して、ライトがオンまたはオフになっているかどうかを確認します。

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

matter.OnOff 機能のすべての状態プロパティを読み取ります。

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

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

UpdateState の例

SendManagedThingCommand API を使用してOnTime照明の を変更します。

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

```
  ]
}
```

イベント

イベントは、デバイスによって実装されるスキーマ管理の単方向シグナルです。

以下の制約に従ってイベントを実装します。

- イベント配列に一意のイベントのみを含める
- スキーマドキュメントのサイズ制限内に収まる任意の数のイベントを含める

マネージド統合システムのイベント

イベントの使用

イベントは、デバイスまたはその周囲への変更について事前に学習するための標準化された方法です。これは、デバイスがクラウドに送信して、デバイスで変更された、またはその環境で検出された何かに関する情報を提供するモデル化されたイベントを表します。これらのイベントはモデル化されているため、お客様はコントロールフローでそれらを使用して、特定のイベントとそのイベント内で提供される詳細に対応できます。

イベントの場合、必須要素は nameと ですextrinsicId。オプションの要素は、description、extrinsicProperties、および ですrequest。

説明

説明は、「」で説明されているのと同じ形式に従いdescription、最大長は 512 文字です。

リクエスト

request セクションはオプションであり、リクエストパラメータがない場合は省略できます。省略すると、システムはイベントの名前を使用するだけでペイロードなしでイベントリクエストを送信するデバイスをサポートします。これは、ポンプでのセンサー障害や、煙または一酸化炭素アラームでアラームがミュートされているなどの単純なイベントで使用されます。

複雑なアクションには、追加のパラメータが必要です。たとえば、カメラの映像をストリーミングするリクエストには、使用するストリーミングプロトコルに関するパラメータや、ストリームを特定のディスプレイデバイスに送信するかどうかに関するパラメータが含まれる場合があります。

イベントリクエストの場合、必須要素は ですparameters。オプションの要素はありません。

レスポンス

イベントレスポンスは現在サポートされていません。

タイプ定義のスキーマ

以下のセクションでは、型定義に使用されるスキーマについて詳しく説明します。

$id

$id 要素はスキーマ定義を識別します。次の構造に従う必要があります。

- URI `/schema-versions/` プレフィックスで始める
- `definition` スキーマタイプを含める
- URI パス区切り文字としてスラッシュ (/) を使用する
- フラグメントをピリオドで区切ってスキーマ ID を含める (`.`)
- @ 文字を使用してスキーマ ID とバージョンを区切る
- ピリオド (.) を使用してバージョンフラグメントを分離し、セムバーバージョンで終わる

スキーマ ID は、3 ~ 12 文字のルート名前空間で始まり、その後にオプションのサブ名前空間と名前が続く必要があります。

セムバーバージョンには、メジャーバージョン (最大 3 桁) 、マイナーバージョン (最大 3 桁) 、オプションの PATCH バージョン (最大 4 桁) が含まれます。

> **Note**
>
> 予約済み名前空間`aws`や `matter`

Example$id の例

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

$ref

$ref 要素は、システム内の既存の型定義を参照します。これは、 `$id`要素と同じ制約に従います。

> **Note**
>
> タイプ定義または機能は、 `$ref` ファイルで指定された値で存在する必要があります。

Example$ref の例

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

名前

name 要素は、スキーマドキュメント内のエンティティ名を表す文字列です。多くの場合、略語が含まれており、次のルールに従う必要があります。

- 英数字、ピリオド (.)、スラッシュ (/)、ハイフン (-)、スペースのみを含む
- 文字で始める
- 最大 192 文字

名前要素は、Amazon Web Services コンソールの UI とドキュメントで使用されます。

Example名前の例

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

title

title 要素は、スキーマドキュメントで表されるエンティティの記述文字列です。任意の文字を含めることができ、ドキュメントで使用されます。

Exampleタイトルの例

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

description

description 要素は、スキーマドキュメントで表されるエンティティの詳細な説明を提供します。任意の文字を含めることができ、ドキュメントで使用されます。

Example説明の例

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

extrinsicId

extrinsicId 要素は、Amazon Web Services IoT システム外で管理される識別子を表します。Matter 機能の場合、コンテキストに応じて 、fieldId、、clusterIdattributeIdcommandIdeventId、または にマッピングされます。

は、文字列化された 10 進数整数 (1 ~ 10 桁) または文字列化された 16 進数整数 (0x または 0X プレフィックス、1 ~ 8 桁の 16 進数) のいずれかextrinsicIdです。

> **Note**
>
> AWSベンダー ID (VID) は 0x1577 で、Matter の場合は 0 です。システムは、カスタムスキーマがこれらの予約済み VIDs を機能に使用しないようにします。

Example extrinsicIds の例

```
0018
0x001A
0x15771002
```

extrinsicProperties

extrinsicProperties 要素には、外部システムで定義されているが、データモデル内で維持されている一連のプロパティが含まれています。Matter 機能の場合、ZCL クラスター、属性、コマンド、またはイベント内のさまざまなモデル化されていない要素または部分的にモデル化された要素にマッピングされます。

外部プロパティは、次の制約に従う必要があります。

- プロパティ名は英数字で、スペースや特殊文字は使用できません
- プロパティ値は任意の JSON スキーマ値にすることができます
- 最大 20 個のプロパティ

システムは、extrinsicProperties、、、 などcliFunctionName、さまざまな access apiMaturity cliをサポートしています。これらのプロパティにより、データモデル変換への AWS （ またはその逆の) ACL が容易になります。

> **Note**
>
> 外部プロパティは、機能の action、event、property、および structフィールド要素でサポートされていますが、機能やクラスター自体ではサポートされていません。

システムサポートの外部プロパティ

システムは、ZCL との間の変換extrinsicProperties中に、以下のモデル化されていない、または部分的にモデル化されたクラスター、属性、コマンド、またはイベント属性を追跡します。

access

各アクセスオブジェクトには、以下が含まれています。

- op - 値 read、、writeまたは enumを持つ としてモデル化されたオペレーション invoke
- privilege - 、view、proxy_view、operate、manageまたは の値enumを持つ としてモデル化された特権 administer
- role - 演算子ロールを表す無制限の文字列

apiMaturity

成熟度レベルを表す無制限のプレーン文字列。これは、、stable、、internalまたは の値enumを持つ として ZCL provisionalでモデル化されます。 deprecated

side

、either、serverおよび の値を持つ列挙型としてモデル化 client

ブールプロパティ

以下のプロパティはブールフラグです。

- isFabricScoped
- isFabricSensitive
- mustUseAtomicWrite
- mustUseTimedInvoke

文字列プロパティ

次のプロパティは、無制限の文字列として表されます。

- cli
- cliFunctionName
- functionName
- group
- introducedIn
- manufacturerCode
- noDefaultImplementation
- presentIf
- priority
- removedIn
- reportableChange
- reportMinInterval
- reportMaxInterval
- restriction
- storage

変換に関する考慮事項

ZCL 変換の場合、 extrinsicPropertiesは処理せずにマップに保存されます。検出を使用するカスタムスキーマは ZCL 変換を受けません。ただし、今後カスタムスキーマに ZCL 変換を実装する場合は、すべての無制限のプレーン文字列タイプをモデルextrinsicProperties化し、列挙型、パターン (正規表現) 、長さなどの制約を定義する必要があります。この準備により、変換中にこれらのプロパティを適切に処理できます。

対照的に、 からコネクタ AWS への変換の場合、コネクタ形式ではこれらの詳細は必須ではないため、 extrinsicPropertiesはまったく含まれません。

機能スキーマドキュメントでの型定義の構築と使用

スキーマ内のすべての要素は、型定義に解決されます。これらの型定義は、プリミティブ型定義 (ブール値、文字列、数値など) または名前空間型定義 (便宜上、プリミティブ型定義から構築された型定義) のいずれかです。

カスタムスキーマを定義する場合、プリミティブ定義と名前空間タイプ定義の両方を使用できます。

目次

プリミティブ型の定義

プリミティブ型定義は、マネージド統合で定義されたすべての型定義の構成要素です。カスタム型定義を含むすべての名前空間定義は、 `$ref`キーワードまたは `type`キーワードを使用してプリミティブ型定義に解決されます。

すべてのプリミティブ型は `nullable`キーワードを使用して null にでき、 `type`キーワードを使用してすべてのプリミティブ型を識別できます。

ブール値

ブール型はデフォルト値をサポートします。

サンプル定義：

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

整数タイプのサポート

整数型は以下をサポートします。

- default値
- maximum値
- minimum値
- exclusiveMaximum値
- exclusiveMinimum値
- multipleOf値

x が検証対象の値である場合、次の条件が満たされている必要があります。

- x ≥ minimum
- x > exclusiveMinimum
- x < exclusiveMaximum

> **Note**
>
> 小数部がゼロの数値は整数と見なされますが、浮動小数点数は拒否されます。
>
> ```
> 1.0 // Schema-Compliant
> 3.1415926 // NOT Schema-Compliant
> ```

minimum と の両方、exclusiveMinimumまたは maximumと の両方を指定できますがexclusiveMaximum、両方を同時に使用することはお勧めしません。

サンプル定義：

```
{
    "type" : "integer",
    "default" : 2,
    "nullable" : true,
    "maximum" : 10,
    "minimum" : 0,
    "multipleOf": 2
}
```

代替定義：

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

数字

整数や浮動小数点数など、任意の数値型に数値型を使用します。

数値タイプは以下をサポートします。

- default値
- maximum値
- minimum値
- exclusiveMaximum値
- exclusiveMinimum値
- multipleOf 値。複数の は浮動小数点数にすることができます。

x が検証対象の値である場合、次の条件が満たされている必要があります。

- x ≥ minimum
- x > exclusiveMinimum
- x < exclusiveMaximum

minimum と の両方、exclusiveMinimumまたは maximumと の両方を指定できますがexclusiveMaximum、両方を同時に使用することはお勧めしません。

サンプル定義：

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

代替定義：

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

文字列

文字列タイプは以下をサポートします。

- default値
- maxLength および minLength値を含む長さ制約 (負以外の数値である必要があります)
- pattern 正規表現の値

正規表現を定義すると、式が文字列内の任意の場所と一致する場合、文字列は有効です。たとえば、正規表現は、文字列「p」だけでなく、「apple」などの p を含む任意の文字列pに一致します。明確にするために、特に理由がない限り、正規表現を ^...$ (など^p$) で囲むことをお勧めします。

サンプル定義：

```
{
```

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

Null

Null 型は 1 つの値のみを受け入れます。 `null`

サンプル定義：

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

配列

配列タイプは以下をサポートします。

- `default` — デフォルト値として使用されるリスト。
- `items` — すべての配列要素に適用される JSON 型定義。
- 長さの制約 (負以外の数値である必要があります)
  - `minItems`
  - `maxItems`
- `pattern` 正規表現の値
- `uniqueItems` — 配列内の要素を一意にする必要があるかどうかを示すブール値
- `prefixItems` — 各項目がドキュメントの配列の各インデックスに対応するスキーマである配列。つまり、最初の要素が入力配列の最初の要素を検証し、2 番目の要素が入力配列の 2 番目の要素を検証する配列などです。

サンプル定義：

```
{
   "type": "array",
   "default": ["1", "2"],
   "items" : {
        "type": "string",
        "pattern": "^([a-zA-Z0-9_ -/]+)$"
```

```
    },
    "minItems" : 1,
    "maxItems": 4,
    "uniqueItems" : true,
}
```

配列検証の例：

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

タプル検証を使用した代替定義：

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

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

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

オブジェクト

オブジェクトタイプは以下をサポートしています。

- プロパティの制約
  - properties — propertiesキーワードを使用して、オブジェクトのプロパティ (キーと値のペア) を定義します。の値は オブジェクトpropertiesで、各キーはプロパティの名前、各値はそのプロパティの検証に使用されるスキーマです。properties キーワード内のプロパティ名のいずれにも一致しないプロパティは、このキーワードによって無視されます。

- required — デフォルトでは、 propertiesキーワードで定義されたプロパティは必要ありません。ただし、 requiredキーワードを使用して必要なプロパティのリストを指定できます。required キーワードは、0 個以上の文字列の配列を取ります。これらの文字列はそれぞれ一意である必要があります。
- propertyNames — このキーワードにより、プロパティ名の RegEx パターンを制御できます。たとえば、オブジェクトのすべてのプロパティに、特定の規則に従って名前を付けるように強制できます。
- patternProperties — 正規表現をスキーマにマッピングします。プロパティ名が指定された正規表現と一致する場合、プロパティ値は対応するスキーマに対して検証する必要があります。たとえば、 を使用して、特定の種類のプロパティ名を指定して、値が特定のスキーマと一致するようにpatternProperties指定します。
- additionalProperties — このキーワードは、追加のプロパティの処理方法を制御します。追加のプロパティは、プロパティキーワードに名前がリストされていないプロパティ、またはの正規表現のいずれかに一致するプロパティですpatternProperties。デフォルトでは、追加のプロパティが許可されます。このフィールドを に設定するfalseと、追加のプロパティは許可されません。
- unevaluatedProperties — このキーワードは、サブスキーマで宣言されたプロパティを認識できる点additionalPropertiesを除いて、 に似ています。 は、スキーマの処理時に正常に検証されたプロパティを収集し、それらをプロパティの許可されたリストとして使用することでunevaluatedProperties機能します。これにより、条件付きでプロパティを追加するなど、より複雑な操作を実行できます。詳細については、以下の例を参照してください。

- anyOf — このキーワードの値は空でない配列である必要があります。配列の各項目は、有効な JSON スキーマである必要があります。インスタンスは、このキーワードの値で定義された少なくとも 1 つのスキーマに対して正常に検証された場合、このキーワードに対して正常に検証されます。
- oneOf — このキーワードの値は空でない配列である必要があります。配列の各項目は、有効な JSON スキーマである必要があります。インスタンスは、このキーワードの値で定義された 1 つのスキーマに対して正常に検証された場合、このキーワードに対して正常に検証されます。

必須の例：

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

```
// Schema Compliant
{
  "test": 4
}

// NOT Schema Compliant
{}
```

PropertyNames の例：

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

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

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

PatternProperties の例：

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

// Schema Compliant
{ "S_25": "This is a string" }
{ "I_0": 42 }

// NOT Schema Compliant
{ "S_0": 42 } // Value must be a string
```

```
{ "I_42": "This is a string" } // Value must be an integer
```

AdditionalProperties の例：

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

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

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

UnevaluatedProperties の例：

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

// Schema Compliant
{
  "standard_field": "some value",
  "@id": 123,
```

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

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

AnyOf の例：

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

// Schema Compliant
"short"
12

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

OneOf の例：

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

// Schema Compliant
10
9
```

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

名前空間型の定義

名前空間型定義は、プリミティブ型から構築された型です。これらのタイプは、*namespace*.*typename*. マネージド統合が aws および matter名前空間の下に事前定義されたタイプを提供する形式に従う必要があります。予約済み awsおよび 名前空間以外のカスタムタイプには、任意のmatter名前空間を使用できます。

使用可能な名前空間型の定義を見つけるには、Typeフィルターを に設定して ListSchemaVersions API を使用しますdefinition。

matter タイプ

フィルターを に設定matterし、Namespaceフィルターを Type に設定して、ListSchemaVersions API を使用してmatter名前空間でデータ型を検索しますdefinition。

aws タイプ

フィルターを に設定awsし、Namespaceフィルターを Type に設定して ListSchemaVersions API を使用してaws、名前空間でデータ型を検索しますdefinition。

ビットマップタイプ定義

ビットマップには 2 つの必須プロパティがあります。

- type は オブジェクトである必要があります
- properties は、各ビット定義を含むオブジェクトである必要があります。各ビットは、 プロパティextrinsicIdと を持つオブジェクトですvalue。各ビットの値は、最小値が 0、最大値が 1 以上の整数である必要があります。

サンプルビットマップ定義：

```
{
    "title" : "Sample Bitmap Type",
    "description" : "Type definition for SampleBitmap.",
```

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

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

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

列挙型の定義

列挙型には 3 つのプロパティが必要です。

- type は オブジェクトである必要があります
- enum は、少なくとも 1 つの項目を持つ一意の文字列の配列である必要があります
- extrinsicIdMap は、列挙値であるプロパティを持つオブジェクトです。各プロパティの値は、列挙値に対応する外部識別子である必要があります。

列挙型定義の例：

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

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

// NOT Schema Compliant
"NotAnEnumValue"
```

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、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

コマンドリクエストは、デバイスに送信されるコマンドです。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、 SendManagedThingCommand API がマネージド統合によって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。

へのレスポンスSendManagedThingCommandは traceIdであり、これを使用してtraceId、可能な限りコマンド配信および関連するコマンドレスポンスワークフローを追跡できます。

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に通知されます。

マネージド統合リソースのタグ付け

リソースの管理と整理に役立つように、オプションでこれらの各リソースに独自のメタデータをタグ形式で割り当てることができます。このセクションでは、タグとその作成方法について説明します。

ベーシックタグ

タグを使用して、さまざまな方法 (目的、所有者、環境など) でマネージド統合リソースを分類できます。同じ型のリソースが多い場合に役立ちます。割り当てたタグに基づいてリソースをすばやく識別できます。タグはそれぞれ、1 つのキーとオプションの値で構成され、どちらもユーザーが定義します。たとえば、デバイスをタイプ別に追跡しやすくするために、モノのタイプに対して一連のタグを定義できます。リソースの種類ごとのニーズを合わせて一連のタグキーを作成することをお勧めします。一貫性のあるタグキーセットを使用することで、リソースの管理が容易になります。

追加または適用したタグに基づいて、リソースを検索およびフィルター処理できます。また、「IAM ポリシーでのタグの使用」で説明しているように、タグを使用してリソースへのアクセスを制御することもできます。

使いやすいように、 AWS マネジメントコンソールのタグエディタは、タグを作成および管理するための一元化された方法を提供します。詳細については、「 マネジメントコンソールの使用」の「タグエディタの使用 AWS」を参照してください。

AWS CLI と マネージド統合 API を使用してタグを操作することもできます。次のコマンドの `Tags`フィールドを使用して、タグを作成するときに、マネージドモノ、プロビジョニングプロファイル、認証情報ロッカー、over-the-air (OTA) タスクにタグを関連付けることができます。

- CreateManagedThing
- CreateProvisioningProfile
- CreateCredentialLocker
- CreateOtaTask
- CreateAccountAssociation

以下のコマンドを使用して、タグ付けがサポートされている既存のリソースに対してタグを追加、変更、または削除できます。

- TagResource

- ListTagsForResource
- UntagResource

タグのキーと値は編集でき、タグはリソースからいつでも削除できます。タグの値を空の文字列に設定することはできますが、タグの値を null に設定することはできません。特定のリソースについて既存のタグと同じキーを持つタグを追加した場合、古い値は新しい値によってオーバーライドされます。リソースを削除すると、リソースに関連付けられているすべてのタグも削除されます。

タグの制約と制限

タグには以下のような基本制限があります。

- リソースあたりのタグの最大数: 50
- キーの最大長: 127 文字 UnicodeUTF-8)
- 値の最大長: 255 文字 (Unicode) (UTF-8)
- タグのキーと値は大文字と小文字が区別されます。
- タグの名前または値に「aws:」プレフィックスは使用しないでください。 AWS 使用のために予約されています。このプレフィックスが含まれるタグの名前または値は編集または削除できません。このプレフィックスを持つタグは、リソースあたりのタグ数の制限にはカウントされません。
- 複数のサービス間およびリソース間でタグ付けスキーマを使用する場合、他のサービスでも許可される文字に制限が適用されることがあるため注意ください。使用できる文字は、UTF-8 で表現できる文字、スペース、および数字と、特殊文字 +、-、=、.、_、:、/、@ です。

IAM ポリシーでのタグの使用

マネージド統合 API アクションに使用する IAM ポリシーで、タグベースのリソースレベルのアクセス許可を適用できます。これにより、ユーザーがどのリソースを作成、変更、または使用できるかを制御しやすくなります。IAM ポリシーの以下の条件コンテキストのキーと値とともに Condition 要素 (Condition ブロックとも呼ばれる) を使用して、リソースのタグに基づいてユーザーアクセス (アクセス許可) を制御できます。

- 特定のタグを持つリソースに対してユーザーアクションを許可または拒否するには、aws:ResourceTag/*tag-key*: *tag-value* を使用します。
- タグが許可されているリソースを作成または変更する API リクエストを作成する場合に、特定のタグが使用されている (または、使用されていない) ことを要求するには、aws:RequestTag/*tag-key*: *tag-value* を使用します。

- タグが許可されているリソースを作成または変更する API リクエストを作成する場合に、特定の一連のタグが使用されている または、使用されていないことを要求するには、`aws:TagKeys: [tag-key, ...]` を使用します。

> **Note**
>
> IAM ポリシーの条件コンテキストキーと値は、タグ付けできるリソースの識別子が必須パラメータであるマネージド統合アクションにのみ適用されます。例えば、GetCustomEndpoint の使用は、このリクエストではタグ付け可能なリソース (マネージド型モノ、プロビジョニングプロファイル、認証情報ロッカー、over-the-airタスク) が参照されないため、条件コンテキストキーと値に基づいて許可または拒否されません。タグ付け可能なマネージド統合リソースとそれらがサポートする条件キーの詳細については、「 の AWS IoT マネージド統合機能のアクション、リソース、および条件キー AWS IoT Device Management」を参照してください。

タグの使用の詳細については、[AWS Identity and Access Management User Guide](ユーザーガイド)の[Controlling Access Using Tags](タグを使用したアクセスの制御)を参照してください。そのガイドの[IAM JSON Policy Reference](IAM JSON ポリシーリファレンス)セクションには、IAM での JSON ポリシーの要素、変数、および評価ロジックの詳細な構文、説明、および例が記載されています。

次のポリシー例では、 `CreateManagedThing`アクションに 2 つのタグベースの制限を適用します。このポリシーによって制限されている IAM ユーザー

- 「env=prod」というタグが付いたマネージド型モノを作成することはできません (例では、「」行を参照してください`"aws:RequestTag/env" : "prod"`)。
- 既存のタグ「env=prod」を持つマネージドモノを変更またはアクセスすることはできません (例では、「」行を参照してください`"aws:ResourceTag/env" : "prod"`)。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": "iotmanagedintegrations:CreateManagedThing",
```

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

次のように、タグ値を1 つのリストとして指定して、1 つのタグキーに対して複数のタグ値を指定することもできます。

```
          "StringEquals" : {
```

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

> **Note**
>
> タグに基づいてリソースへのユーザーのアクセスを許可または拒否する場合は、ユーザーが同じリソースに対してそれらのタグを追加または削除することを明示的に拒否することを検討する必要があります。そうしないと、ユーザーはそのリソースのタグを変更することで、制限を回避してリソースにアクセスできてしまいます。

マネージド統合通知

マネージド統合通知は、デバイスからの更新と主要なインサイトを提供します。通知には、コネクタイベント、デバイスコマンド、ライフサイクルイベント、OTA (Over-the-Air) 更新、エラーレポートが含まれます。これらのインサイトは、自動化されたワークフローを作成したり、すぐにアクションを実行したり、トラブルシューティングのためにイベントデータを保存したりするための実用的な情報を提供します。

現在、マネージド統合通知の送信先としてサポートされているのは Amazon Kinesis データストリームのみです。通知を設定する前に、まず Amazon Kinesis データストリームを設定し、マネージド統合によるデータストリームへのアクセスを許可する必要があります。

通知用に Amazon Kinesis をセットアップする

Amazon Kinesis のセットアップ手順

- ステップ 1: Amazon Kinesis データストリームを作成する
- ステップ 2: アクセス許可ポリシーを作成する
- ステップ 3: IAM ダッシュボードに移動し、ロールを選択する
- ステップ 4: カスタム信頼ポリシーを使用する
- ステップ 5: アクセス許可ポリシーを適用する
- ステップ 6: ロール名を入力する

マネージド統合通知用に Amazon Kinesis を設定するには、次の手順に従います。

ステップ 1: Amazon Kinesis データストリームを作成する

Amazon Kinesis Data Stream は、大量のデータをリアルタイムで取り込み、永続的に保存して、データをアプリケーションで使用できるようにすることができます。

Amazon 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。

ステップ 2: アクセス許可ポリシーを作成する

マネージド統合が Kinesis データストリームにアクセスできるようにするアクセス許可ポリシーを作成します。

アクセス許可ポリシーを作成するには

- アクセス許可ポリシーを作成するには、以下のポリシーをコピーし、「JSON エディタを使用してポリシーを作成する」で説明されているステップに従います。

  JSON

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

ステップ 3: IAM ダッシュボードに移動し、ロールを選択する

IAM ダッシュボードを開き、ロールをクリックします。

IAM ダッシュボードに移動するには

- IAM ダッシュボードを開き、ロールをクリックします。

  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。

ステップ 4: カスタム信頼ポリシーを使用する

カスタム信頼ポリシーを使用して、マネージド統合に Kinesis データストリームへのアクセスを許可できます。

カスタム信頼ポリシーを使用するには

- 新しいロールを作成し、カスタム信頼ポリシーを選択します。Next をクリックします。

  次のポリシーでは、マネージド統合がロールを引き受けることを許可し、 Conditionステートメントは混乱した代理問題を防ぐのに役立ちます。

  JSON

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

ステップ 5: アクセス許可ポリシーを適用する

ステップ 2 で作成したアクセス許可ポリシーをロールに追加します。

アクセス許可ポリシーを追加するには

- アクセス許可の追加ページで、ステップ 2 で作成したアクセス許可ポリシーを検索して追加します。Next をクリックします。

ステップ 6: ロール名を入力する

- ロール名を入力し、ロールの作成をクリックします。

マネージド統合通知を設定する

通知のセットアップ手順

- ステップ 1: CreateDestination API を呼び出すアクセス許可をユーザーに付与する
- ステップ 2: CreateDestination API を呼び出す
- ステップ 3: CreateNotificationConfiguration API を呼び出す

マネージド統合通知を設定するには、次の手順に従います。

ステップ 1: CreateDestination API を呼び出すアクセス許可をユーザーに付与する

- **`CreateDestination`** API を呼び出すアクセス許可をユーザーに付与する

  次のポリシーは、ユーザーが CreateDestination API を呼び出すための要件を定義します。

  マネージド統合へのパスロールアクセス許可を取得するには、「 ユーザーガイド」の「 AWS サービスにロールを渡すアクセス許可をユーザーに付与する」を参照してください。 AWS Identity and Access Management

  JSON

```
{
      "Version":"2012-10-17",
      "Statement":[
         {
            "Effect":"Allow",
            "Action":"iam:PassRole",

 "Resource":"arn:aws:iam::123456789012:role/ROLE_CREATED_IN_PREVIOUS_STEP",
            "Condition":{
               "StringEquals":{
```

```
 "iam:PassedToService":"iotmanagedintegrations.amazonaws.com"
                }
            }
        },
        {
            "Effect":"Allow",
            "Action":"iotmanagedintegrations:CreateDestination",
            "Resource":"*"
        }
    ]
}
```

ステップ 2: CreateDestination API を呼び出す

- **`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、通知がルーティングされる通知先を作成します。`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": "string"
    "RoleArn": "arn:aws:iam::accountID:role/ROLE_CREATED_IN_PREVIOUS_STEP"
}
```

> **Note**
>
> `ClientToken` はべき等性トークンです。同じクライアントトークンとパラメータを使用して最初に正常に完了したリクエストを再試行すると、それ以上のアクションを実行せずに再試行が成功します。

ステップ 3: CreateNotificationConfiguration API を呼び出す

- **`CreateNotificationConfiguration`** API を呼び出す

  最後に、CreateNotificationConfiguration API を使用して、選択したイベントタイプを Kinesis データストリームで表される送信先にルーティングする通知設定を作成します。`DestinationName` パラメータでは、最初に `CreateDestination` API を呼び出したとき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": "string"
}
```

マネージド統合でモニタリングされるイベントタイプ

マネージド統合通知でモニタリングされるイベントタイプは次のとおりです。

- `DEVICE_COMMAND`
  - SendManagedThingCommand API コマンドのステータス。有効な値は`succeeded`か`failed`のどちらかです。

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

```
                "result":"failed"
            }
}
```

- DEVICE_COMMAND_REQUEST
  - 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、WebRTC」を参照してください。

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

- DEVICE_DISCOVERY_STATUS
  - デバイスの検出ステータス。

```
{
      "version":"0",
      "messageId":"6a7e8feb-b491-4cf7-a9f1-bf3703467718",
```

```
      "messageType":"DEVICE_DISCOVERY_STATUS",
      "source":"aws.iotmanagedintegrations",
      "customerAccountId":"123456789012",
      "timestamp":"2017-12-22T18:43:48Z",
      "region":"ca-central-1",
      "resources":[
        "arn:aws:iotmanagedintegrations:ca-central-1:123456789012:managed-
thing/6a7e8feb-b491-4cf7-a9f1-bf3703467718"
      ],
      "payload":{
        "deviceCount": 1,
        "deviceDiscoveryId": "123",
        "status": "SUCCEEDED"
      }
}
```

- DEVICE_EVENT
  - 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":[
        "/quit/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": "ca-central-1",
      "resources": [
        "arn:aws:iotmanagedintegrations:ca-central-1:123456789012:managed-thing/
d5c280b423a042f3933eed09cf408657"
        ],
      "payload": {
        "deviceDetails": {
          "id": "d5c280b423a042f3933eed09cf408657",
          "arn": "arn:aws:iotmanagedintegrations:ca-central-1:123456789012:managed-
thing/d5c280b423a042f3933eed09cf408657",
          "createdAt": "2024-11-14T19:55:57.515841147Z",
          "updatedAt": "2024-11-14T19:55:57.515841559Z"
        },
        "status": "UNCLAIMED"
      }
}
```

- DEVICE_OTA
  - 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:ca-central-1:123456789012:managed-
thing/61889008880012345678"
```

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

Cloud-to-Cloud (C2C) コネクタ

cloud-to-cloud コネクタを使用すると、サードパーティーのデバイスと 間の双方向通信を作成および促進できます AWS。

トピック

cloud-to-cloud (C2C) コネクタとは

cloud-to-cloudコネクタは、 をサードパーティーのクラウドプロバイダーのエンドポイント AWS クラウド に安全にリンクする構築済みのソフトウェアパッケージです。C2C コネクタを使用すると、ソリューションプロバイダーは AWS IoT Device Management のマネージド統合を活用して、サードパーティーのクラウドに接続されているデバイスを制御できます。

マネージド統合には、 AWS お客様が統合するコネクタを表示および選択できるコネクタのカタログが含まれています。詳細については、C2C コネクタカタログとはを参照してください。

マネージド統合では、すべてのコネクタを AWS Lambda 関数として実装する必要があります。

C2C コネクタカタログとは

AWS IoT Device Management コネクタカタログのマネージド統合は、AWS IoT Device Management とサードパーティーのクラウドプロバイダーのマネージド統合間の双方向通信を容易にする C2C コネクタのコレクションです。コネクタは、 AWS マネジメントコンソール または で表示できます AWS CLI。

コンソールを使用してマネージド統合コネクタカタログを表示するには

1. マネージド統合コンソールを開く

2. 左側のナビゲーションペインで、マネージド統合を選択します。
3. マネージド統合コンソールの左側のナビゲーションペインで、カタログを選択します。

AWS Lambda C2C コネクタとして機能する

すべての C2C コネクタ Lambda 関数は、マネージド統合とサードパーティープラットフォーム上の対応するアクションとの間でコマンドとイベントを変換して転送します。Lambda の詳細については、「 とは AWS Lambda」を参照してください。

たとえば、エンドユーザーがサードパーティー OEM によって製造されたスマート電球を所有しているとします。C2C コネクタを使用すると、エンドユーザーはマネージド統合プラットフォームを介してこのライトをオンまたはオフにするコマンドを発行できます。その後、このコマンドはコネクタでホストされている Lambda 関数に転送され、リクエストはサードパーティープラットフォームに対する API コールに変換され、デバイスのオン/オフが切り替わります。

`CreateCloudConnector` API を呼び出すときは、Lambda 関数が必要です。Lambda 関数にデプロイされるコードは、「」で説明されているように、すべてのインターフェイスと機能を実装する必要がありますC2C (Cloud-to-Cloud) コネクタを構築する。

マネージド統合コネクタのワークフロー

デベロッパーは、 のマネージド統合に C2C コネクタを登録する必要があります AWS IoT Device Management。この登録プロセスにより、顧客がコネクタを使用するためにアクセスできる論理コネクタリソースが作成されます。

> **Note**
>
> C2C コネクタは、AWS IoT Device Management がコネクタを記述するためにマネージド統合内で作成されるメタデータのセットです。

次の図は、モバイルアプリケーションからクラウド接続デバイスにコマンドを送信するときの C2C コネクタの役割を示しています。C2C コネクタは、AWS IoT Device Management のマネージド統合とサードパーティーのクラウドプラットフォーム間の変換レイヤーとして機能します。

C2C (cloud-to-cloud) コネクタを使用するためのガイドライン

作成する C2C コネクタはお客様のコンテンツであり、お客様がアクセスする別の顧客によって作成された C2C コネクタはサードパーティーのコンテンツです。 AWS は、マネージド統合の一部として C2C コネクタを作成または管理しません。

C2C コネクタは、他のマネージド型統合の顧客と共有できます。その場合は、サービスプロバイダー AWS として、これらの C2C コネクタと関連する連絡先情報を AWS コンソールに一覧表示することを に許可し、他の AWS 顧客が連絡する可能性があることを理解します。お客様は、お客様に C2C コネクタへのアクセスを許可し、別の AWS お客様の C2C コネクタへのアクセスに適用される条件について単独で責任を負います。

C2C (Cloud-to-Cloud) コネクタを構築する

以下のセクションでは、AWS IoT Device Management のマネージド統合用の C2C (Cloud-to-Cloud) コネクタを構築する手順について説明します。

トピック

前提条件

C2C (Cloud-to-Cloud) コネクタを作成する前に、以下が必要です。

- C2C コネクタを AWS アカウント ホストし、マネージド統合を通じて登録するための 。詳細については、「 の作成 AWS アカウント」を参照してください。
- コネクタを構築するときは、特定の IAM アクセス許可が必要です。を使用するには
- コネクタの対象となるサードパーティーのクラウドプロバイダーが OAuth 2.0 認可をサポートしていることを確認します。詳細については、「アカウントリンクの OAuth 2.0 要件」を参照してください。

さらに、コネクタをテストするには、コネクタの開発者に以下が必要です。

- C2C コネクタに関連付けるサードパーティークラウドからのクライアント ID
- C2C コネクタに関連付けるサードパーティークラウドからのクライアントシークレット
- OAuth 2.0 認可 URL
- OAuth 2.0 トークン URL

- サードパーティー API に必要な API キー
- がホストする OAuth コールバック URL のサードパーティー API 登録または許可リストに必要な API キー AWS。OAuth リダイレクト URL を明示的に許可リストに登録するサードパーティーもあれば、ユーザーがログインして OAuth URL を登録できるワークフローを持つサードパーティーもあります。特定のサードパーティーと相談して、マネージド統合 OAuth リダイレクトエンドポイントを許可リストに登録するために必要なものを理解してください。

必要なアクセス許可

コネクタを構築するときは、特定の IAM アクセス許可が必要です。アクション の`iotmanagedintegrations:`アクセス許可に加えて、次のアクセス許可が必要です。

- CreateAccountAssociation、CreateConnectorDestination、GetAccountAssociation、StartAccountAssociat 必須 `secretsmanager:GetSecretValue`
- CreateCloudConnector には が必要です `lambda:Invoke`

アクセス`iotmanagedintegrations:`許可とアクションの詳細については、AWS 「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。

C2C コネクタの要件

開発する C2C コネクタは、AWS IoT Device Management のマネージド統合とサードパーティーベンダークラウド間の双方向通信を容易にします。コネクタは、エンドユーザーに代わってアクションを実行するために、AWS IoT Device Management のマネージド統合用のインターフェイスを実装する必要があります。これらのインターフェイスは、エンドユーザーデバイスを検出し、AWS IoT Device Management のマネージド統合から送信されるデバイスコマンドを開始し、アクセストークンに基づいてユーザーを識別する機能を提供します。デバイスオペレーションをサポートするために、コネクタは AWS IoT Device Management のマネージド統合と関連するサードパーティープラットフォーム間のリクエストおよびレスポンスメッセージの変換を管理する必要があります。

C2C コネクタの要件は次のとおりです。

- サードパーティーの認可サーバーは、OAuth 2.0 標準および に記載されている設定に準拠している必要がありますOAuth 設定要件。
- Matter Data Model の AWS 実装から識別子を解釈するには C2C コネクタが必要であり、 Matter Data Model の AWS 実装に準拠するレスポンスとイベントを出力する必要があります。詳細については、AWS Matter データモデルの実装を参照してください。
- C2C コネクタは、AWS IoT Device Management APIs のマネージド統合をSigV4認証で呼び出すことができる必要があります。SendConnectorEvent API で送信される非同期イベントの場合、コネクタの登録に使用したのと同じ AWS アカウント 認証情報を使用して、関連する SendConnectorEvent リクエストに署名する必要があります。
- コネクタは、AWS.ActivateUser、、AWS.SendCommand、および AWS.DiscoverDevicesAWS.DeactivateUserオペレーションを実装する必要があります。
- C2C コネクタは、デバイスコマンドレスポンスまたはデバイス検出に関連するサードパーティーイベントを受信すると、それらを SendConnectorEvent API とのマネージド統合に転送する必要があります。これらのイベントと SendConnectorEvent API の詳細については、SendConnectorEvent」を参照してください。

> **Note**
>
> SendConnectorEvent API はマネージド統合 SDK の一部であり、リクエストを手動で構築して署名する代わりに使用されます。

アカウントリンクの OAuth 2.0 要件

すべての C2C コネクタは、エンドユーザーを認証するために OAuth 2.0 認可サーバーに依存しています。このサーバーを通じて、エンドユーザーはサードパーティーアカウントを顧客のデバイスプラットフォームにリンクします。アカウントリンクは、エンドユーザーが C2C コネクタでサポートされているデバイスを使用するために必要な最初のステップです。アカウントリンクと OAuth 2.0 のさまざまなロールの詳細については、「」を参照してくださいアカウントリンクロール。

C2C コネクタは認可フローをサポートするために特定のビジネスロジックを実装する必要はありませんが、C2C コネクタに関連付けられた OAuth2.0 認可サーバーは を満たす必要がありますOAuth 設定要件。

> **Note**
>
> のマネージド統合は、認可コードフローを持つ OAuth 2.0 AWS IoT Device Management のみをサポートします。詳細については、「RFC 6749」を参照してください。

アカウントリンクは、マネージド統合とコネクタがアクセストークンを使用してエンドユーザーのデバイスにアクセスできるようにするプロセスです。このトークンは、コネクタが API コールを通じてエンドユーザーのデータとやり取りできるように、エンドユーザーのアクセス許可を持つ AWS IoT Device Management のマネージド統合を提供します。詳細については、「アカウントリンクワークフロー」を参照してください。

これらの機密トークンはログに記録しないことをお勧めします。ただし、ログに保存されている場合は、CloudWatch Logs データ保護ポリシーを使用してログ内のトークンをマスクすることをお勧めします。詳細については、「Help protect sensitive log data with masking」を参照してください。

のマネージド統合 AWS IoT Device Management では、アクセストークンは直接取得されません。これは、認可コード付与タイプを通じて取得されます。まず、AWS IoT Device Management のマネージド統合は認可コードを取得する必要があります。次に、コードをアクセストークンと更新トークンと交換します。更新トークンは、古いアクセストークンの有効期限が切れたときに新しいアクセストークンをリクエストするために使用されます。アクセストークンと更新トークンの両方の有効期限が切れている場合は、アカウントリンクフローを再度実行する必要があります。これを行うには、`StartAccountAssociationRefresh` API オペレーションを使用します。

> **Important**
>
> 発行されるアクセストークンはユーザーごとにスコープする必要がありますが、OAuth クライアントごとにスコープすることはできません。トークンは、クライアント内のすべてのユーザーのすべてのデバイスへのアクセスを提供するべきではありません。
> 認可サーバーは、次のいずれかを実行する必要があります。
>
> - JWT トークンなどの抽出可能なエンドユーザー (リソース所有者) ID を含むアクセストークンを発行します。
> - 発行された各アクセストークンのエンドユーザー ID を返します。

OAuth 設定要件

次の表は、AWS IoT Device Management のマネージド統合でアカウントリンクを実行するために必要な OAuth 認可サーバーからのパラメータを示しています。

OAuth サーバーパラメータ

フィールド	必須	[Comment] (コメント)
`clientId`	あり	アプリケーションのパブリック識別子。これは認証フローを開始するのに使用され、パブリックに共有できます。
`clientSecret`	あり	特にアクセストークンの認可コードを交換するときに、認可サーバーでアプリケーションを認証するために使用されるシークレットキー。機密に保持し、公開しないでください。
`authorizationType`	あり	この認可設定でサポートされている認可のタイプ。現在、サポートされている値は OAuth 2.0」のみです。
`authUrl`	あり	サードパーティーのクラウドプロバイダーの承認 URL。
`tokenUrl`	あり	サードパーティーのクラウドプロバイダーのトークン URL。
`tokenEndpointAuthenticationScheme`	あり	「HTTP_BASIC」または「REQUEST_BODY_CREDENTIALS」のいずれかの認証スキーム。HTTP_BASIC は、クライアント認証情報が認可

		ヘッダーに含まれていることをシグナルし、ラダーはリクエスト本文に含まれていることをシグナルします。

アクセストークンの文字列値が UTF-8 文字セットでエンコードされた Base64 になるように、使用する OAuth サーバーを設定する必要があります。

アカウントリンクロール

C2C コネクタを作成するには、OAuth 2.0 認可サーバーとアカウントのリンクが必要です。詳細については、「アカウントリンクワークフロー」を参照してください。

OAuth 2.0 では、アカウントリンクを実装する際に次の 4 つのロールを定義します。

1. 認可サーバー
2. リソース所有者 (エンドユーザー)
3. リソースサーバー
4. クライアント

以下は、これらの OAuth ロールのそれぞれを定義します。

認可サーバー

認可サーバーは、サードパーティークラウド内のエンドユーザーの ID を識別して認証するサーバーです。このサーバーによって提供されるアクセストークンは、 AWS エンドユーザーのカスタマープラットフォームアカウントとそのサードパーティープラットフォームアカウントをリンクできます。このプロセスはアカウントリンクと呼ばれます。

認可サーバーは、以下を提供することでアカウントリンクをサポートします。

- エンドユーザーがシステムにログインするためのログインページを表示します。これは通常、認可エンドポイントと呼ばれます。
- システム内のエンドユーザーを認証します。
- エンドユーザーを識別する認可コードを生成します。
- AWS IoT Device Management のマネージド統合に認可コードを渡します。
- AWS IoT Device Management のマネージド統合から認可コードを受け取り、AWS IoT Device Management のマネージド統合がシステム内のエンドユーザーのデータにアクセスするために

使用できるアクセストークンを返します。これは通常、トークン URI またはエンドポイントと呼ばれる別の URI を介して完了します。

> **Important**
>
> 認可サーバーは、AWS IoT Device Management Connector のマネージド統合で使用する OAuth 2.0 認可コードフローをサポートする必要があります。AWS IoT Device Management の マネージド統合は、コード交換の証明キー (PKCE) による認可コードフローもサポートしています。
> 認可サーバーは次のいずれかである必要があります。
>
> - JWT トークンなど、抽出可能なエンドユーザーまたはリソース所有者 ID を含むアクセストークンを発行する
> - 発行された各アクセストークンのエンドユーザー ID を返すことができる
>
> そうしないと、コネクタは必要なAWS.ActivateUserオペレーションをサポートできなくなります。これにより、マネージド統合でのコネクタの使用が防止されます。

コネクタ開発者または所有者が独自の認可サーバーを維持していない場合、使用する認可サーバーはコネクタ開発者のサードパーティープラットフォームによって管理されるリソースに対する認可を提供する必要があります。つまり、認可サーバーからマネージド統合によって受信されるトークンは、デバイス (リソース) に意味のあるセキュリティ境界を提供する必要があります。たとえば、エンドユーザートークンは別のエンドユーザーデバイスでのコマンドを許可しません。トークンによって提供されるアクセス許可は、プラットフォーム内のリソースにマッピングされます。Lights" の例を考えてみましょう。エンドユーザーがコネクタとアカウントリンクフローを開始すると、認可サーバーの前にある Lights" ログインページにリダイレクトされます。ログインしてクライアントにアクセス許可を付与すると、コネクタに Lights" アカウント内のリソースへのアクセスを許可するトークンが提供されます。

リソース所有者 (エンドユーザー)

リソース所有者は、アカウントリンクを実行して、AWS IoT Device Management の マネージド統合に、アカウントに関連付けられたリソースへのカスタマーアクセスを許可します。例えば、エンドユーザーが Lights" モバイルアプリケーションにオンボードしたスマート電球を考えてみましょう。リソース所有者とは、デバイスを購入してオンボーディングしたエンドユーザーアカウントを指します。この例では、リソース所有者は Lights" OAuth2.0 アカウントとしてモデル化されています。リソース所有者として、このアカウントはコマンドを発行し、デバイスを管理するためのアクセス許可を提供します。

リソースサーバー

これは、 (デバイスデータ) にアクセスするための認可を必要とする保護されたリソースをホストするサーバーです。お客様は、エンドユーザーに代わって保護されたリソースにアクセス AWS する必要があり、アカウントリンク後の AWS IoT Device Management コネクタのマネージド統合を通じてアクセスします。以前からのスマート電球を例として考えた場合、リソースサーバーは Lights" が所有するクラウドベースのサービスであり、オンボーディング後に電球を管理します。リソースサーバーを通じて、リソース所有者はスマート電球のオン/オフなどのコマンドをスマート電球に発行できます。保護されたリソースは、エンドユーザーのアカウントと、エンドユーザーがアクセス許可を付与した可能性のある他のアカウント/エンティティにのみアクセス許可を提供します。

クライアント

このコンテキストでは、クライアントは C2C コネクタです。クライアントは、エンドユーザーに代わってリソースサーバー内のリソースへのアクセスが付与されるアプリケーションとして定義されます。アカウントリンクプロセスは、コネクタ、クライアントを表し、サードパーティークラウド内のエンドユーザーのリソースへのアクセスをリクエストします。

コネクタは OAuth クライアントですが、AWS IoT Device Management のマネージド統合はコネクタに代わってオペレーションを実行します。例えば、AWS IoT Device Management のマネージド統合は、アクセストークンの取得を認可サーバーにリクエストします。コネクタは、リソースサーバーの保護されたリソース (デバイスデータ) にアクセスする唯一のコンポーネントであるため、引き続きクライアントと見なされます。

エンドユーザーによってオンボードされたスマート電球について考えてみましょう。顧客プラットフォームと Lights" 認可サーバー間のアカウントリンクが完了すると、コネクタ自体がリソースサーバーと通信して、エンドユーザーのスマート電球に関する情報を取得します。その後、コネクタはエンドユーザーからコマンドを受信できます。これには、Lights" リソースサーバーを介して、ユーザーに代わって照明をオンまたはオフにすることが含まれます。したがって、コネクタをクライアントとして指定します。

アカウントリンクワークフロー

AWS IoT Device Management プラットフォームのお客様が C2C コネクタを介してサードパーティープラットフォーム上のエンドユーザーのデバイスとやり取りするためのマネージド統合の場合、次のワークフローを通じてアクセストークンを取得します。

1. ユーザーがカスタマーアプリケーションを通じてサードパーティーデバイスのオンボーディングを開始すると、AWS IoT Device Management のマネージド統合は認可 URI と AssociationId を返します。
2. アプリケーションのフロントエンドは AssociationId を保存し、エンドユーザーをサードパーティープラットフォームのログインページにリダイレクトします。

   - エンドユーザーがサインインします。エンドユーザーは、クライアントにデバイスデータへのアクセスを許可します。
3. サードパーティープラットフォームは認可コードを作成します。エンドユーザーは、リダイレクトリクエストにアタッチされたコードを含む AWS IoT Device Management プラットフォームコールバック URI のマネージド統合にリダイレクトされます。
4. マネージド統合は、このコードをサードパーティーのプラットフォームトークン URI と交換します。
5. トークン URI は認可コードを検証し、エンドユーザーに関連付けられた OAuth2.0 アクセストークンと更新トークンを返します。
6. マネージド統合は、 `AWS.ActivateUser`オペレーションで C2C コネクタを呼び出して、アカウントリンクフローを完了し、UserId を取得します。
7. マネージド統合は、成功した認証ページの OAuthRedirectUrl (コネクタポリシー設定から) をカスタマーアプリケーションに返します。

   > **Note**
   >
   > 障害が発生した場合、AWS IoT Device Management のマネージド統合は、エラーの詳細を顧客アプリケーションに提供する URL にエラーと error_description クエリパラメータを追加します。
8. カスタマーアプリケーションは、エンドユーザーを OAuthRedirectUrl にリダイレクトします。この時点で、アプリケーションフロントエンドは最初のステップからの関連付けの AssociationId を認識します。

   デバイスを検出してコマンドを送信するコマンドなど、C2C コネクタを介して AWS IoT Device Management のマネージド統合からサードパーティーのクラウドプラットフォームに対して行われる後続のすべてのリクエストには、OAuth2.0 アクセストークンが含まれます。

次の図は、アカウントリンクの主要コンポーネント間の関係を示しています。

C2C コネクタインターフェイスオペレーションの実装

のマネージド統合は、コネクタとして認定するために処理 AWS Lambda する必要がある 4 つのオペレーション AWS IoT Device Management を定義します。C2C コネクタは、以下の各オペレーションを実装する必要があります。

1. AWS.ActivateUser - AWS IoT Device Management サービスのマネージド統合は、この API を呼び出して、提供された OAuth2.0 トークンに関連付けられたグローバルに一意のユーザー識別子を取得します。このオペレーションは、オプションでアカウントリンクプロセスの追加要件を実行するために使用できます。
2. AWS.DiscoverDevices - AWS IoT Device Management サービス用の マネージド統合は、この API をコネクタに呼び出してユーザーのデバイスを検出します。
3. AWS.SendCommand - AWS IoT Device Management サービス用の マネージド統合は、この API をコネクタに呼び出して、ユーザーのデバイス用のコマンドを送信します。
4. AWS.DeactivateUser - AWS IoT Device Management サービス用の マネージド統合では、この API をコネクタに呼び出して、ユーザーのアクセストークンを非アクティブ化し、認可サーバーでリンクを解除します。

のマネージド統合は、 AWS IoT Device Management 常に アクションを通じて JSON 文字列ペイロードを使用して Lambda 関数を呼び出します AWS Lambda invokeFunction。リクエストオペレーションには、すべてのリクエストペイロードに operationNameフィールドを含める必要があります。詳細については、 AWS Lambda 「 API リファレンス」の「呼び出し」を参照してください。

各呼び出しタイムアウトは 2 秒に設定され、呼び出しが失敗すると 5 回再試行されます。

コネクタに実装する Lambda は、リクエストペイロードoperationNameから を解析し、対応する機能を実装してサードパーティークラウドにマッピングします。

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

```
        );
    }

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

> **Note**
>
> コネクタの開発者はactivateUserManager.activateUser(request)、前の例に示す deviceDiscoveryManager.listDevices(request)、sendCommandManager.sendCommand(および deactivateUser.deactivateUserオペレーションを実装する必要があります。

次の例では、すべての必須インターフェイスへの共通フィールドが存在するマネージド統合からの汎用コネクタリクエストについて詳しく説明します。この例では、リクエストヘッダーとリクエストペイロードの両方があることがわかります。リクエストヘッダーは、すべてのオペレーションインターフェイスで共通です。

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

デフォルトのリクエストヘッダー

デフォルトのヘッダーフィールドは次のとおりです。

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

コネクタによってホストされる API は、次のヘッダーパラメータを処理する必要があります。

デフォルトのヘッダーとフィールド

フィールド	必須/オプション	[Description] (説明)
`header:auth`	あり	コネクタの登録中に C2C コネクタビルダーによって提供される認可情報。
`header:auth:token`	あり	サードパーティーのクラウドプロバイダーによって生成され、 にリンクされたユーザーの認可トークン`connectorAssociationID` 。
`header:auth:type`	あり	必要な認可のタイプ。

> **Note**
>
> コネクタへのすべてのリクエストには、エンドユーザーのアクセストークンがアタッチされます。エンドユーザーとマネージド統合の顧客との間のアカウントリンクがすでに発生していると仮定できます。

リクエストペイロード

一般的なヘッダーに加えて、すべてのリクエストにはペイロードがあります。このペイロードにはオペレーションタイプごとに一意のフィールドがありますが、各ペイロードには、常に存在する一連のデフォルトフィールドがあります。

リクエストペイロードフィールド:

- operationName: 指定されたリクエストのオペレーション。、AWS.ActivateUser、AWS.SendCommand、 AWS.DiscoverDevicesのいずれかの値に相当しますAWS.DeactivateUser。
- operationVersion: すべてのオペレーションは、時間の経過とともに進化し、サードパーティーコネクタの安定したインターフェイス定義を提供するようにバージョン管理されています。マネージド統合は、すべてのリクエストのペイロードでバージョンフィールドを渡します。
- connectorId: リクエストが送信されたコネクタの ID。

デフォルトのレスポンスヘッダー

すべてのオペレーションは、AWS IoT Device Management のマネージド統合ACKに で応答し、C2C コネクタがリクエストを受信して処理を開始したことを確認します。以下は、前述のレスポンスの一般的な例です。

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

すべてのオペレーションレスポンスには、次の共通ヘッダーが必要です。

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

次の表に、デフォルトのレスポンスヘッダーを示します。

デフォルトのレスポンスヘッダーとフィールド

フィールド	必須/オプション	[Comment] (コメント)

header:responseCode	あり	リクエストの実行ステータスを示す値の ENUM。

このドキュメントで説明されているさまざまなコネクタインターフェイスと API スキーマには、responseMessageまたは Messageフィールドがあります。これは、C2C コネクタ Lambda がリクエストとその実行に関するコンテキストで応答するために使用されるオプションのフィールドです。以外のステータスコードが発生するエラーには、エラーを説明するメッセージ値を含める200ことをお勧めします。

SendConnectorEvent API を使用して C2C コネクタオペレーションリクエストに応答する

のマネージド統合では、 AWS.SendCommandおよび AWS.DiscoverDevicesオペレーションごとにコネクタが非同期的に動作すること AWS IoT Device Management を想定しています。つまり、これらのオペレーションに対する最初のレスポンスは、C2C コネクタがリクエストを受け取ったことを「承認」するだけです。

SendConnectorEvent API を使用すると、コネクタは、以下のリストから AWS.DiscoverDevices および AWS.SendCommandオペレーションのイベントタイプと、プロアクティブデバイスイベント (手動でオンまたはオフになっているライトなど) を に送信することが期待されます。これらのイベントタイプとそのユースケースの詳細な説明については、AWS.DiscoverDevices オペレーションを実装する「」、AWS.SendCommand オペレーションを実装する「」、「」を参照してくださいSendConnectorEvent API を使用してデバイスイベントを送信する。

例えば、C2C コネクタがDiscoverDevicesリクエストを受信した場合、AWS IoT Device Management のマネージド統合では、上記で定義されたレスポンス形式で同期的に応答することが期待されます。次に、DEVICE_DISCOVERY イベントに対してAWS.DiscoverDevices オペレーションを実装する、 で定義されたリクエスト構造を使用して SendConnectorEventAPI を呼び出す必要があります。API SendConnectorEventでの 呼び出しは、C2C コネクタの Lambda AWS アカウント 認証情報にアクセスできる任意の場所で実行できます。AWS IoT Device Management のマネージド統合がこのイベントを受信するまで、デバイス検出フローは成功しません。

> **Note**
>
> または、必要に応じて C2C コネクタの Lambda 呼び出しレスポンスの前に SendConnectorEvent API コールが発生する場合があります。ただし、このフローはソフトウェア開発の非同期モデルと矛盾します。

- SendConnectorEvent - コネクタは、AWS IoT Device Management API のこのマネージド統合を呼び出して、AWS IoT Device Management のマネージド統合にデバイスイベントを送信します。マネージド統合で受け入れられるイベントは 3 種類のみです。
  - 「DEVICE_DISCOVERY」 — このイベントオペレーションは、特定のアクセストークン用にサードパーティークラウド内で検出されたデバイスのリストを送信するために使用されます。
  - 「DEVICE_COMMAND_RESPONSE」 – このイベントオペレーションは、コマンド実行の結果として特定のデバイスイベントを送信するために使用されます。
  - 「DEVICE_EVENT」 – このイベントオペレーションは、ユーザーベースのコマンドの直接の結果ではないデバイスから発生したイベントに使用されます。これは、デバイスの状態の変更や通知をプロアクティブに報告するための一般的なイベントタイプとして機能します。

AWS.ActivateUser オペレーションを実装する

AWS.ActivateUser オペレーションは、AWS IoT Device Management のマネージド統合で、エンドユーザーの OAuth2.0 トークンからユーザー識別子を取得するために必要です。のマネージド統合 AWS IoT Device Management は、リクエストヘッダー内に OAuth トークンを渡し、コネクタがレスポンスペイロードにグローバルに一意のユーザー識別子を含めることを期待します。このオペレーションは、アカウントリンクフローが成功した後に発生します。

次のリストは、AWS.Activateユーザーフローを成功させるためのコネクタの要件の概要を示しています。

- C2C コネクタ Lambda は、AWS IoT Device Management のマネージド統合からのAWS.ActivateUserオペレーションリクエストメッセージを処理できます。
- C2C コネクタ Lambda は、提供された OAuth2.0 トークンから一意のユーザー識別子を決定できます。通常、JWT トークンの場合はトークン自体から抽出することも、トークンによって認可サーバーからリクエストすることもできます。

AWS.ActivateUser のワークフロー

1. のマネージド統合は、次のペイロードを使用して C2C コネクタ Lambda を AWS IoT Device Management 呼び出します。

```
{
   "header": {
        "auth": {
            "token": "ashriu32yr97feqy7afsaf",
            "type": "OAuth2.0"
        }
   },
   "payload": {
        "operationName": "AWS.ActivateUser",
        "operationVersion": "1.0.0",
        "connectorId": "Your-Connector-ID",
}
}
```

2. C2C コネクタは、トークンから、またはサードパーティーのリソースサーバーにクエリを実行して、AWS.ActivateUserレスポンスに含めるユーザー ID を決定します。
3. C2C コネクタは、デフォルトのペイロードと userIdフィールド内の対応するユーザー識別子を含むAWS.ActivateUserオペレーション Lambda 呼び出しに応答します。

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

AWS.DiscoverDevices オペレーションを実装する

デバイス検出は、エンドユーザーが所有する物理デバイスのリストを、AWS IoT Device Management のマネージド統合で維持されているエンドユーザーデバイスのデジタル表現と整合させます。これは、AWS IoT Device Management のユーザーとマネージド統合の間でアカウン

トリンクが完了した後にのみ、エンドユーザーが所有するデバイスで AWS 顧客によって実行されます。デバイス検出は、AWS IoT Device Management のマネージド統合がコネクタを呼び出してデバイス検出リクエストを開始する非同期プロセスです。C2C コネクタは、検出されたエンドユーザーデバイスのリストを、マネージド統合によって生成された参照識別子 (と呼ばれますdeviceDiscoveryId) とともに非同期的に返します。

次の図は、AWS IoT Device Management のエンドユーザーとマネージド統合間のデバイス検出ワークフローを示しています。

AWS.DiscoverDevices ワークフロー

1. お客様は、エンドユーザーに代わってデバイス検出プロセスを開始します。
2. のマネージド統合は、お客様が生成 AWS したデバイス検出リクエストdeviceDiscoveryIdに という参照識別子 AWS IoT Device Management を生成します。
3. のマネージド統合は、エンドユーザーaccessTokenの有効な OAuth と を含むAWS.DiscoverDevicesオペレーションインターフェイスを使用して、C2C コネクタにデバイス検出リクエスト AWS IoT Device Management を送信しますdeviceDiscoveryId。
4. コネクタは、 DEVICE_DISCOVERY イベントに含まれる deviceDiscoveryId を保存します。このイベントには、検出されたエンドユーザーのデバイスのリストも含まれ、DEVICE_DISCOVERYイベントとして SendConnectorEvent API を使用して AWS IoT Device Management のマネージド統合に送信する必要があります。
5. C2C コネクタはリソースサーバーを呼び出して、エンドユーザーが所有するすべてのデバイスを取得します。
6. C2C コネクタ Lambda は Lambda 呼び出し (invokeFunction) に応答し、ACK 応答を AWS IoT Device Management のマネージド統合に返し、AWS.DiscoverDevicesオペレーションの初期応答として機能します。マネージド統合は、開始されたデバイス検出プロセスに ACK で顧客に通知します。
7. リソースサーバーは、エンドユーザーが所有および運用するデバイスのリストを送信します。
8. コネクタは、各エンドユーザーデバイスを、 を含む AWS IoT Device Management に必要なデバイス形式のマネージド統合ConnectorDeviceId、ConnectorDeviceNameおよび各デバイスの機能レポートに変換します。
9. C2C コネクタは、検出されたデバイスの所有者もUserId提供します。デバイスリストの一部として、またはリソースサーバーの実装に応じて別の呼び出しで、リソースサーバーから取得できます。

10. 次に、C2C コネクタは認証情報を使用し、オペレーションパラメータを「DEVICE_DISCOVERY」に設定 AWS アカウント してSendConnectorEvent、SigV4 経由で AWS IoT Device Management API のマネージド統合 を呼び出します。AWS IoT Device Management のマネージド統合に送信されるデバイスのリスト内の各デバイスは、connectorDeviceId、connectorDeviceName、 などのデバイス固有のパラメータで表されますcapabilityReport。

    - リソースサーバーのレスポンスに基づいて、AWS IoT Device Management のマネージド統合を適宜通知する必要があります。

      たとえば、リソースサーバーがエンドユーザーの検出されたデバイスのリストにページ分割されたレスポンスを持っている場合、ポーリングごとに statusCodeパラメータを にして、個々のDEVICE_DISCOVERYオペレーションイベントを送信できます3xx。デバイス検出がまだ進行中の場合は、ステップ 5、6、7 を繰り返します。

11. のマネージド統合は、 がエンドユーザーのデバイスを検出したことに関する通知を顧客 AWS IoT Device Management に送信します。

12. C2C コネクタが値 200 で更新された statusCodeパラメータでDEVICE_DISCOVERYオペレーションイベントを送信すると、マネージド統合はデバイス検出ワークフローの完了をお客様に通知します。

> **Important**
>
> 必要に応じて、ステップ 7 ~ 11 をステップ 6 の前に実行できます。たとえば、サードパーティープラットフォームにエンドユーザーデバイスを一覧表示する API がある場合、C2C コネクタ Lambda が一般的な ACK で応答SendConnectorEventする前に、DEVICE_DISCOVERY イベントを で送信できます。

デバイス検出の C2C コネクタ要件

次のリストは、デバイス検出を成功させるための C2C コネクタの要件の概要を示しています。

- C2C コネクタ Lambda a は、AWS IoT Device Management のマネージド統合からのデバイス検出リクエストメッセージを処理して、AWS.DiscoverDevicesオペレーションを処理できます。
- C2C コネクタは、コネクタの登録 AWS アカウント に使用される の認証情報を使用して、SigV4 経由で AWS IoT Device Management APIs のマネージド統合を呼び出すことができます。

デバイス検出プロセス

次の手順では、AWS IoT Device Management の C2C コネクタとマネージド統合を使用したデバイス検出プロセスの概要を説明します。 AWS IoT Device Management

デバイス検出プロセス

1. マネージド統合は、デバイス検出をトリガーします。

   - 次の JSON ペイロードDiscoverDevicesを使用して POST リクエストを に送信します。

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

2. Connector は検出を承認します。

   - コネクタは、次の JSON レスポンスを含む確認を送信します。

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

3. コネクタは Device Discovery Event を送信します。

- 次の JSON ペイロード/connector-event/*{your_connector_id}*を使用して POST リクエストを に送信します。

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

DISCOVER_DEVICES イベントの CapabilityReport を作成する

上記のイベント構造に示すように、 DISCOVER_DEVICES イベントで報告されたすべてのデバイスは、`AWS.DiscoverDevices`オペレーションへの応答として機能し、対応するデバイスの機能を記述するために CapbilityReport が必要です。CapabilityReport」は、AWS IoT Device Management デバイス機能のマネージド統合を Matter 準拠の形式で指示します。「CapabilityReport」には、次のフィールドを指定する必要があります。

- `nodeId`、文字列: 以下を含むデバイスノードの識別子 `endpoints`
- `version`、文字列: コネクタ開発者によって設定されたこのデバイスノードのバージョン
- `endpoints`、List<Cluster>: このデバイスエンドポイントでサポートされている Matter データモデルの AWS 実装のリスト。
  - `id`、文字列: コネクタ開発者によって設定されたエンドポイント識別子
  - `deviceTypes`、List<String>: このエンドポイントがキャプチャするデバイスタイプのリスト、つまり「カメラ」。
  - `clusters`、List<Cluster>: このエンドポイントがサポートする Matter データモデルの AWS 実装のリスト。
    - `id`、文字列: Matter 標準で定義されているクラスター識別子。
    - `revision`、整数: Matter 標準で定義されているクラスターリビジョン番号。
    - `attributes`、Map<String、Object>: 属性識別子とそれに対応する現在のデバイス状態値のマップ。問題標準で定義された識別子と有効な値。
      - `id`、文字列: Matter データモデルの AWS 実装で定義される属性 ID。
      - `value`、オブジェクト: 属性 ID で定義された属性の現在の値。「値」のタイプは、属性によって異なる場合があります。`value` フィールドは各属性に対してオプションであり、検出中にコネクタ Lambda が現在の状態を判断できる場合にのみ含める必要があります。
    - `commands`、List<String>: Matter 標準で定義されているように、このクラスターをサポートしているコマンド IDs のリスト。
    - `events`、List<String>: Matter 標準で定義されているように、このクラスターでサポートされているイベント IDs のリスト。

サポートされている機能の現在のリストと AWS Matter Data Model の対応する実装については、Data Model ドキュメントの最新リリースを参照してください。

AWS.SendCommand オペレーションを実装する

このAWS.SendCommandオペレーションにより、AWS IoT Device Management のマネージド統合は、エンドユーザーが開始したコマンドを AWS 顧客経由でリソースサーバーに送信できます。リソースサーバーは、複数のタイプのデバイスをサポートする場合があります。各タイプには独自のレスポンスモデルがあります。コマンド実行は、AWS IoT Device Management のマネージド統合がtraceId」でコマンド実行のリクエストを送信する非同期プロセスです。このリクエストは、コネクタがSendConnectorEvent」 API を介してマネージド統合に返送されるコマンドレスポンスに含めます。AWS IoT Device Management の マネージド統合では、リソースサーバーがコマンドが受信されたことを確認するレスポンスを返すことを想定していますが、必ずしもコマンドが実行されたことを示すとは限りません。

次の図は、エンドユーザーが家の照明をオンにしようとする例を含むコマンド実行フローを示しています。

デバイスコマンド実行ワークフロー

1. エンドユーザーは、 AWS 顧客のアプリケーションを使用して照明をオンにするコマンドを送信します。
2. 顧客は、エンドユーザーのデバイス情報を使用して、AWS IoT Device Management のマネージド統合にコマンド情報を中継します。
3. マネージド統合は、コマンドレスポンスをサービスに送り返すときにコネクタが使用する「traceId」を生成します。
4. AWS IoT Device Management の マネージド統合は、 AWS.SendCommandオペレーションインターフェイスを使用してコマンドリクエストをコネクタに送信します。
    - このインターフェイスで定義されるペイロードは、デバイス識別子、 Matter endpoints/clusters/commandsとして定式化されたデバイスコマンド、エンドユーザーのアクセストークン、およびその他の必須パラメータで構成されます。
5. コネクタは、コマンドレスポンスに含まれる `traceId` を保存します。
    - コネクタは、マネージド統合コマンドリクエストをリソースサーバーの適切な形式に変換します。
6. コネクタは、提供されたエンドユーザーのアクセストークンUserIdから取得し、 コマンドに関連付けます。

a. `UserId` は、別の呼び出しを使用してリソースサーバーから取得するか、JWT および同様のトークンの場合はアクセストークンから抽出できます。

b. 実装は、リソースサーバーとアクセストークンの詳細によって異なります。

7. コネクタは、リソースサーバーを「ターンオン」エンドユーザーのライトに呼び出します。
8. リソースサーバーはデバイスとやり取りします。

a. コネクタは、リソースサーバーが コマンドを配信した AWS IoT Device Management のマネージド統合にリレーし、最初の同期コマンドレスポンスとして ACK で応答します。

b. マネージド統合は、それをカスタマーアプリケーションに中継します。

9. デバイスがライトをオンにすると、そのデバイスイベントはリソースサーバーによってキャプチャされます。
10. リソースサーバーは、デバイスイベントをコネクタに送信します。
11. コネクタは、リソースサーバーによって生成されたデバイスイベントをマネージド統合 DEVICE_COMMAND_RESPONSE イベントオペレーションタイプに変換します。
12. コネクタは、 オペレーションを「DEVICE_COMMAND_RESPONSE」として `SendConnectorEvent` API を呼び出します。

- 最初のリクエストで AWS IoT Device Management のマネージド統合によって`traceId`提供される をアタッチします。

13. マネージド統合は、エンドユーザーのデバイス状態の変化を顧客に通知します。
14. お客様は、デバイスのライトがオンになったことをエンドユーザーに通知します。

> **Note**
>
> リソースサーバー設定によって、失敗したデバイスコマンドのリクエストおよびレスポンスメッセージを処理するロジックが決まります。これには、コマンドに同じ referenceId を使用したメッセージの再試行が含まれます。

デバイスコマンド実行の C2C コネクタ要件

次のリストは、デバイスコマンドの実行を成功させるための C2C コネクタの要件の概要を示しています。

- C2C コネクタ Lambda は、AWS IoT Device Management のマネージド統合から の`AWS.SendCommand`オペレーションリクエストメッセージを処理できます。

- C2C コネクタは、リソースサーバーに送信されたコマンドを追跡し、適切なtraceId」にマッピングする必要があります。
- C2C コネクタの登録 AWS アカウント に使用される AWS 認証情報を使用して、SigV4 経由で AWS IoT Device Management サービス API の マネージド統合を呼び出すことができます。

1. マネージド統合は、コネクタにコマンドを送信します (前の図のステップ 4 を参照) 。

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

2. C2C コネクタ ACK コマンド (コネクタが AWS IoT Device Management Service のマネージド統合に ACK を送信する前の図のステップ 7 を参照) 。

   ```
   {
        "header":{
   ```

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

3. コネクタは Device Command Response イベントを送信します (前の図のステップ 11 を参照) 。

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

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

> **Note**
>
> コマンド実行の結果としてのデバイス状態の変更は、対応する DEVICE_COMMAND_RESPONSE イベントが SendConnectorEvent API を介して受信されるまで、AWS IoT Device Management のマネージド統合に反映されません。つまり、マネージド統合が前のステップ 3 からイベントを受信するまで、コネクタ呼び出しレスポンスが成功を示すかどうかにかかわらず、デバイスの状態は更新されません。

AWS.SendCommand リクエストに含まれる問題「エンドポイント」の解釈

マネージド統合では、デバイス検出中に報告されたデバイス機能を使用して、デバイスが受け入れることができるコマンドを決定します。すべてのデバイス機能は Matter データモデルの AWS 実装によってモデル化されるため、すべての受信コマンドは特定のクラスター内の「 コマンド」フィールドから取得されます。コネクタは、「endpoints」フィールドを解析し、対応する Matter コマンドを決定し、正しいコマンドがデバイスに到達するように変換する責任があります。通常、これは Matter データモデルを関連する API リクエストに変換することを意味します。

コマンドが実行されると、コネクタは Matter Data Model の AWS 実装によって定義されたどの「属性」が結果として変更されたかを決定します。これらの変更は、API で送信された API DEVICE_COMMAND_RESPONSE イベントを介して AWS IoT Device Management のマネージド統合に報告されますSendConnectorEvent。

次のAWS.SendCommandペイロードの例に含まれている「endpoints」フィールドを考えてみましょう。

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

このオブジェクトから、コネクタは以下を判断できます。

1. エンドポイントとクラスター情報を設定します。
   a. エンドポイントをid「1」に設定します。

      > **Note**
      > デバイスが複数のエンドポイントを定義し、単一のクラスター (オン/オフなど) が複数の機能を制御できる場合 (ライトのオン/オフ、ストロボのオン/オフなど) 、この ID はコマンドを正しい機能にルーティングするために使用されます。

   b. クラスターを「0x0202」 (ファンコントロールクラスター) idに設定します。
2. コマンド情報を設定します。
   a. コマンド識別子を「0xff01」 (で定義される状態の更新コマンド) に設定します AWS。
   b. 含まれている属性識別子を、リクエストで指定された値で更新します。
3. 属性を更新します。
   a. 属性識別子を「0x0000」 (ファンコントロールクラスターのFanMode 属性) に設定します。
   b. 属性値を「3」 (高速ファン) に設定します。

マネージド統合では、Matter データモデルの AWS 実装で厳密に定義されていない 2 つの「カスタム」コマンドタイプ、ReadState コマンドと UpdateState コマンドが定義されています。Matter で定義されたクラスター属性を取得して設定するために、マネージド統合は UpdateState (id: 0xff01) または ReadState (id: 0xff02) に関連するコマンド IDs と、更新または読み取りが必要な属性の対応するパラメータを含むAWS.SendCommandリクエストをコネクタに送信します。これらのコマンドは、 Matter Data Model の対応する AWS 実装から変更可能 (更新可能) または取得可能 (読み取り可能) に設定された属性の任意のデバイスタイプに対して呼び出すことができます。

SendConnectorEvent API を使用してデバイスイベントを送信する

デバイス開始イベントの概要

SendConnectorEvent API は AWS.SendCommandおよび AWS.DiscoverDevicesオペレーションに非同期で応答するために使用されますが、デバイス開始イベントをマネージド統合に通知するためにも使用されます。デバイス開始イベントは、ユーザーが開始したコマンドなしでデバイスによって生成されたイベントとして定義できます。これらのデバイスイベントには、デバイスの状態の変更、

モーション検出、バッテリーレベルなどが含まれますが、これらに限定されません。これらのイベントは、 SendConnectorEvent API と オペレーション DEVICE_EVENT を使用してマネージド統合に送信できます。

次のセクションでは、これらのイベントのワークフローをさらに説明する例として、ホームにインストールされたスマートカメラを使用します。

デバイスイベントワークフロー

1. カメラは、リソースサーバーに送信されるイベントを生成するモーションを検出します。
2. リソースサーバーはイベントを処理し、C2C コネクタに送信します。
3. コネクタは、このイベントを AWS IoT Device Management DEVICE_EVENTインターフェイスのマネージド統合に変換します。
4. C2C コネクタは、オペレーションが「DEVICE_EVENT」に設定された SendConnectorEvent API を使用して、このデバイスイベントをマネージド統合に送信します。
5. マネージド統合は、関連する顧客を識別し、このイベントを顧客に中継します。
6. 顧客はこのイベントを受け取り、ユーザー識別子を介してユーザーに表示されます。

SendConnectorEvent API オペレーションの詳細については、AWS IoT Device Management API リファレンスガイドSendConnectorEventの マネージド統合の「」を参照してください。

デバイス開始イベントの要件

以下は、デバイス主導イベントの要件です。

- C2C コネクタリソースは、リソースサーバーから非同期デバイスイベントを受信できる必要があります
- C2C コネクタリソースは、C2C コネクタの登録 AWS アカウント に使用される の AWS 認証情報を使用して、SigV4 経由で AWS IoT Device Management サービス API の マネージド統合を呼び出すことができる必要があります。

次の例は、SendConnectorEvent API を介してデバイス発信イベントを送信するコネクタを示しています。

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

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

次の例から、次のようになります。

- これは、ID が 1 のデバイスエンドポイントから取得されます。
- このイベントに関連するデバイス機能は、ファンコントロールの問題クラスターに関連するクラスター ID が 0x0202 です。
- 変更された属性の ID は 0x000 で、クラスター内のファンモード列挙型に関連しています。High の値に関連する値 3 に更新されました。
- connectorId は作成時にクラウドサービスによって返されるパラメータであるため、コネクタは GetCloudConnector を使用してクエリを実行し、 でフィルタリングする必要がありますlambdaARN。Lambda 自体ARNは Lambda.get_function_url_config API を使用してクエリされます。これにより、Lambda で CloudConnectorIdに動的にアクセスでき、以前のように静的に設定されることはありません。

AWS.DeactivateUser オペレーションを実装する

ユーザー非アクティブ化の概要

提供されたユーザーアクセストークンの非アクティブ化は、顧客が AWS 顧客アカウントを削除する場合、またはエンドユーザーが AWS 顧客のシステムからシステム内のアカウントのリンクを解除

する場合に必要です。どちらのユースケースのマネージド統合でも、C2C コネクタを使用してこのワークフローを容易にする必要があります。

次の図は、システムからエンドユーザーアカウントをリンク解除する方法を示しています。

ユーザー非アクティブ化ワークフロー

1. ユーザーは、 AWS 顧客のアカウントと C2C コネクタに関連付けられたサードパーティー認可サーバー間のリンク解除プロセスを開始します。
2. お客様は、AWS IoT Device Management のマネージド統合を通じてユーザーの関連付けの削除を開始します。
3. マネージド統合は、 `AWS.DeactivateUser`オペレーションインターフェイスを使用してコネクタへのリクエストを介して非アクティブ化プロセスを開始します。
   - /user のアクセストークンは、リクエストの ヘッダーに含まれます。
4. C2C コネクタはリクエストを受け入れ、認可サーバーを呼び出してトークンとトークンが提供するアクセスを取り消します。
   - たとえば、リンクされていないユーザーアカウントからのイベントは、 の実行後にマネージド統合に送信されなくなります`AWS.DeactivateUser`。
5. 認可サーバーはアクセスを取り消し、C2C コネクタにレスポンスを返します。
6. C2C コネクタは、AWS IoT Device Management のマネージド統合に、ユーザーのアクセストークンが取り消されたという ACK を送信します。
7. マネージド統合は、リソースサーバーに関連付けられたエンドユーザーが所有するすべてのリソースを削除します。
8. マネージド統合は、システムに関連するすべての関連付けが削除されたことを示す ACK を顧客に送信します。
9. お客様は、アカウントがプラットフォームからリンク解除されたことをエンドユーザーに通知します。

AWS.DeactivateUser の要件

- C2C コネクタ Lambda 関数は、 マネージド統合から `AWS.DeactivateUser` オペレーションを処理するためのリクエストメッセージを受け取ります。

- C2C コネクタは、提供された OAuth2.0 トークンと、認可サーバー内のユーザーの対応する更新トークンを取り消す必要があります。

コネクタが受け取るAWS.DeactivateUserリクエストの例を次に示します。

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

         }
     },
     "payload":{
         "operationName": "AWS.DeactivateUser"
         "operationVersion": "1.0"
         "connectorId": "Your-connector-Id"
     }
 }
```

C2C コネクタを呼び出す

AWS Lambda では、リソースベースのポリシーを使用して、Lambda を呼び出すことができるユーザーを承認できます。AWS IoT Device Management のマネージド統合は であるため AWS のサービス、リソースポリシーを介して C2C コネクタ Lambda を呼び出すには、マネージド統合を許可する必要があります。

少なくとも以下の最小限のアクセス許可を持つリソースポリシーを C2C コネクタ Lambda にアタッチします。これにより、Lambda 関数の呼び出し権限とのマネージド統合が提供されます。このポリシーには、 の使いやすさを意図したユーザーconnectorIdのみに制限するのに役立つConditionキーが含まれています。

JSON

```
{
  "Version":"2012-10-17",
  "Id": "default",
  "Statement": [
    {
```

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

IAM ロールにアクセス許可を追加する

すべてのマネージド統合 APIs呼び出すために AWS sigV4 認証が必要です。SigV4 は AWS アカウント 、認証情報を使用して AWS API リクエストを認証するプロトコルに署名しています。マネージド統合 APIs の呼び出しに使用する IAM ロールには、APIs を正常に呼び出すための次のアクセス許可が必要です。

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

これらのアクセス許可の追加の詳細については、 にお問い合わせください サポート。

その他のリソース

C2C コネクタを登録するには、以下が必要です。

- 登録するコネクタを指定する Lambda ARN。

C2C コネクタを手動でテストする

C2C コネクタをend-to-end手動でテストするには、顧客とエンドユーザーの両方をシミュレートする必要があります。

次のリソースが必要です。

- テストするコネクタを指定する AWS Lambda ARN。
- クラウドプラットフォームから OAuth 2.0 ユーザーアカウントをテストします。
- AWS IoT Device Management のマネージド統合に登録されているコネクタ。詳細については、「C2C (Cloud-to-Cloud) コネクタを使用する」を参照してください。

C2C (Cloud-to-Cloud) コネクタを使用する

C2C コネクタは、リクエストとレスポンスメッセージの翻訳を管理し、マネージド統合とサードパーティーベンダークラウド間の通信を可能にします。これにより、さまざまなデバイスタイプ、プラットフォーム、プロトコル間での統合制御が容易になり、サードパーティーのデバイスのオンボーディングと管理が可能になります。

次の手順では、C2C コネクタを使用するステップを一覧表示します。

C2C コネクタを使用する手順:

1. CreateCloudConnector

   マネージド統合とサードパーティーベンダークラウド間の双方向通信を有効にするようにコネクタを設定します。

   コネクタを設定するときは、次の詳細を指定します。

   - 名前: コネクタのわかりやすい名前を選択します。
   - 説明: コネクタの目的と機能の概要を提供します。

- AWS Lambda ARN: コネクタに電力を供給する AWS Lambda 関数の Amazon リソースネーム (ARN) を指定します。

サードパーティーベンダー APIsしてコネクタを作成する AWS Lambda 関数を構築およびデプロイします。次に、マネージド統合内で CreateCloudConnector API を呼び出し、登録用の AWS Lambda 関数 ARN を指定します。 AWS Lambda 関数が、マネージド統合でコネクタを作成するのと同じ AWS アカウントにデプロイされていることを確認します。統合を識別するための一意のコネクタ ID が割り当てられます。

CreateCloudConnector API リクエストとレスポンスの例:

```
Request:

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

Response:

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

作成フロー:

> **Note**
>
> この手順では、必要に応じて GetCloudConnector、UpdateCloudConnector、DeleteCloudConnector、および ListCloudConnectors APIsを使用します。

2. CreateConnectorDestination

送信先を設定して、コネクタがサードパーティーベンダークラウドとの安全な接続を確立するために必要な設定と認証情報を提供します。Destinations を使用して、認可 URL、認証スキーム、認証情報の場所などの OAuth 2.0 認可の詳細など、マネージド統合にサードパーティーの認証情報を登録します AWS Secrets Manager。

前提条件

ConnectorDestination を作成する前に、以下を行う必要があります。

- CreateCloudConnector API を呼び出してコネクタを作成します。関数が返す ID は、CreateConnectorDestination API コールで使用されます。
- コネクタtokenUrlの 3P プラットフォームの を取得します。(authCode を accessToken と交換できます)。
- コネクタの 3P プラットフォームの authUrl を取得します。(エンドユーザーはユーザー名とパスワードを使用して認証できます)。
- アカウントのシークレットマネージャーで clientSecret (3P プラットフォームから) clientIdと を使用します。

CreateConnectorDestination API リクエストとレスポンスの例:

```
Request:

{
    "Name": "CreateConnectorDestination",
    "Description": "CreateConnectorDestination",
    "AuthType": "OAUTH",
    "AuthConfig": {
        "oAuth": {
            "authUrl": "https://xxxx.com/oauth2/authorize",
            "tokenUrl": "https://xxxx/oauth2/token",
            "scope": "testScope",
            "tokenEndpointAuthenticationScheme": "HTTP_BASIC",
            "oAuthCompleteRedirectUrl": "about:blank",
            "proactiveRefreshTokenRenewal": {
                "enabled": false,
                "DaysBeforeRenewal": 30
            }
        }
```

```
    },
    "CloudConnectorId": "<connectorId>", // The connectorID instance from response
 of Step 1.
    "SecretsManager": {
        "arn": "arn:aws:secretsmanager:*****:secret:*******",
        "versionId": "********"
    },
    "ClientToken": "***"
}

Response:

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

クラウド送信先の作成フロー:

> **Note**
>
> この手順では、必要に応じて GetCloudConnector、UpdateCloudConnector、DeleteCloudConnector、および ListCloudConnectors APIsを使用します。

3. CreateAccountAssociation

関連付けは、エンドユーザーのサードパーティーのクラウドアカウントとコネクタの送信先との関係を表します。関連付けを作成し、エンドユーザーをマネージド統合にリンクすると、そのデバイスは一意の関連付け ID を介してアクセスできるようになります。この統合により、デバイスの検出、コマンドの送信、イベントの受信の 3 つの主要な機能が有効になります。

前提条件

AccountAssociation を作成する前に、以下を完了する必要があります。

- CreateConnectorDestination API を呼び出して送信先を作成します。関数が返す ID は、CreateAccountAssociation API コールで使用されます。
- CreateAccountAssociation API を呼び出します。

CreateAccountAssociation API リクエストとレスポンスの例:

```
Request:

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

Response:

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

> **Note**
>
> この手順では、必要に応じて GetCloudConnector、UpdateCloudConnector、DeleteCloudConnector、および ListCloudConnectors APIsを使用します。

AccountAssociation には、GetAccountAssociation API と ListAccountAssociations APIs。これらの APIs、関連付けの状態を示します。StartAccountAssociationRefresh API を使用すると、更新トークンの有効期限が切れたときに AccountAssociation 状態を更新できます。

4. デバイス検出

各マネージド型モノは、シリアル番号やデータモデルなど、デバイス固有の詳細にリンクされます。データモデルはデバイスの機能を記述し、電球、スイッチ、サーモスタット、または別のタイプのデバイスであるかどうかを示します。3P デバイスを検出し、3P デバイス用の managedThing を作成するには、以下のステップを順番に実行する必要があります。

a. StartDeviceDiscovery API を呼び出して、デバイス検出プロセスを開始します。

   StartDeviceDiscovery API リクエストとレスポンスの例:

```
Request:

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

Response:

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

b. GetDeviceDiscovery API を呼び出して、検出プロセスのステータスを確認します。

c. ListDiscoveredDevices API を呼び出して、検出されたデバイスを一覧表示します。

ListDiscoveredDevices API リクエストとレスポンスの例:

```
Request:

//Empty body

Response:

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

d. CreateManagedThing API を呼び出して、検出リストからマネージド統合にインポートするデバイスを選択します。

CreateManagedThing API リクエストとレスポンスのサンプル:

```
Request:

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

Response:

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

e. GetManagedThing API を呼び出して、新しく作成されたこの を表示しますmanagedThing。ステータスは になりますUNASSOCIATED。

f. RegisterAccountAssociation API を呼び出して、これを特定の managedThingに関連付けますaccountAssociation。RegisterAccountAssociation API が正常に終了すると、 はASSOCIATED状態managedThingに変更されます。

RegisterAccountAssociation API リクエストとレスポンスの例:

```
Request:

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

Response:

{
```

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

5. 3P デバイスにコマンドを送信する

新しくオンボーディングされたデバイスを制御するには、SendManagedThingCommand API を使用し、以前に作成した関連付け ID と、デバイスでサポートされている機能に基づくコントロールアクションを使用します。コネクタは、アカウントリンクプロセスから保存された認証情報を使用して、サードパーティーのクラウドで認証し、オペレーションに関連する API コールを呼び出します。

SendManagedThingCommand API リクエストとレスポンスの例:

```
Request:

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

Response:
```

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

3P デバイスフローにコマンドを送信します。

6. コネクタがマネージド統合にイベントを送信する

SendConnectorEvent API は、コネクタからマネージド統合への 4 種類のイベントをキャプチャします。これは、Operation Type パラメータの次の列挙値で表されます。

- DEVICE_COMMAND_RESPONSE: コマンドに応答してコネクタが送信する非同期レスポンス。
- DEVICE_DISCOVERY: デバイス検出プロセスに応答して、コネクタは検出されたデバイスのリストをマネージド統合に送信し、SendConnectorEvent API を使用します。
- DEVICE_EVENT: 受信したデバイスイベントを送信します。
- DEVICE_COMMAND_REQUEST: デバイスから開始されたコマンドリクエスト。例えば、WebRTC ワークフローなどです。

コネクタは、オプションの `userId`パラメータを指定して、SendConnectorEvent API を使用してデバイスイベントを転送することもできます。

- を持つデバイスイベントの場合`userId`:

  SendConnectorEvent API リクエストとレスポンスの例:

```
Request:

{
    "UserId": "*****",
    "Operation": "DEVICE_EVENT",
    "OperationVersion": "1.0",
    "StatusCode": 200,
    "ConnectorId": "****",
    "ConnectorDeviceId": "***",
    "TraceId": "***",
    "MatterEndpoint": {
        "id": "**",
```

```
        "clusters": [{
            .....
            }
        }]
    }
}

Response:

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

- のないデバイスイベントの場合userId:

  SendConnectorEvent API リクエストとレスポンスの例:

```
Request:

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

Response:

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

特定の managedThingとアカウントの関連付け間のリンクを削除するには、登録解除メカニズムを使用します。

DeregisterAccountAssociation API リクエストとレスポンスの例:

```
Request:

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

Response:

HTTP/1.1 200 // Empty body
```

イベントフローの送信:

7. コネクタのステータスを「Listed」に更新して、他のマネージド統合のお客様が表示できるようにします。

   デフォルトでは、コネクタはプライベートであり、コネクタを作成した AWS アカウントにのみ表示されます。コネクタを他のマネージド統合のお客様に表示することを選択できます。

   コネクタを他のユーザーと共有するには、コネクタの詳細ページの AWS マネジメントコンソール 「表示可能にする」オプションを使用して、レビュー AWS のためにコネクタ ID を に送信します。承認されると、コネクタは同じ 内のすべてのマネージド統合ユーザーが使用できます AWS リージョン。さらに、コネクタの関連 AWS Lambda 関数のアクセスポリシーを変更することで、特定の AWS アカウント IDs へのアクセスを制限できます。コネクタを他のお客様が使用できるようにするには、他の AWS アカウントから可視コネクタへの Lambda 関数に対する IAM アクセス許可を管理します。

   コネクタを他のマネージド統合の顧客に表示する前に、コネクタの共有とアクセス許可を管理する AWS のサービス 用語と組織のポリシーを確認してください。

マネージド統合 Hub SDK

このセクションのトピックを使用して、マネージド統合 Hub SDK を使用して IoT ハブデバイスをオンボードおよび制御する方法について説明します。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

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、ユーザー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公開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

認証モードとラジオ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。

Hub SDK クライアント APIs

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

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アップストリームで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、この方法を使用できます。

デバイスコントロール

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。この機能により、状態変更のイベントが出力され、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)、および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、異なる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。

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。

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

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

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検証

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**
>
> iotmi.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 ディレクトリにある に従ってください。

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して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。

```
IoT-managed-integrations-Hub-SDK-aarch64-v1.0.0.tgz
###package/
    ###greengrass/
        ###artifacts/
        ###recipes/
    ###hubSystemdSetup/
        ### REAME.md
        ### copy_to_hub.sh
```

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

の初期セットアップ

SDK パッケージを抽出する

```
tar -xzf managed-integrations-Hub-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。

マネージド統合にハブをオンボードする

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設定します。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。

ハブ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。

ハブ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、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。

ハブ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

デバイスがマネージド統合とデータを交換するために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

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

  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 リファレンスガイド」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。マネージド統合にデバイスをオンボードするには、これらの認証情報が必要です。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

```
aws iot-managed-integrations create-provisioning-profile \
    --provisioning-type 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: コマンドとコントロールをサポートしていないハブCONTROLLERの場合は、この値を に設定します。それ以外の場合は に設定しますDEVICE。
- AuthenticationMaterialType: この値を に設定しますWIFI_SETUP_QR_BAR_CODE。
- AuthenticationMaterial: 次のフィールドを含めます。UPC または のいずれかを使用できますEANが、両方を使用することはできません。
  - 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
  - EAN: このデバイスの国際記事番号

> **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",
 "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PRE_ASSOCIATED前に である必要があります。ProvisioningStatus の詳細については、「デバイスプロビジョニング」を参照してください。GetManagedThing API を使用して、マネージドモノが存在し、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。

  > **Important**
  >
  > 証明書とプライベートキーを PEM 形式で保存する場合は、改行文字を正しく処理して適切な形式にします。PEM エンコードファイルの場合、エスケープされた改行を保存するだけでは後で正しく取得されないため、改行文字は実際の行区切り文字に置き換え(`\n`)る必要があります。

  > **Note**
  >
  > 安全な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、デバイスの認証、プロビジョニングステータスの管理、接続設定の保存を行います。

> **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: 前の ProvisioningProfileの名前。
    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. UPCCreateManagedThing API コールから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で同じ)

```
{
    "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

Key	値	顧客によって追加されましたか?	コメント
iot_provisioning_method	FLEET_PROVISIONING	可	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	可	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	可	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 使用したの名前と同じである必要があります。	可	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。	可	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。

Key	値	顧客によって追加されましたか?	コメント
SN	デバイスのシリアル番号。例えば、AIDACKCEVSQ6C2EXAMPLE 。	可	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。
UPC	デバイスユニバーサル製品コード。例えば、841667145075 。	可	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。	不可	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。	可	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 。	不可	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 。	不可	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。	不可	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。

Key	値	顧客によって追加されましたか?	コメント
mqtt_keep_alive_interval	範囲は 30 ~ 1200 で、単位は秒単位です。デフォルト値は 300 です。	可	これを使用して、MQTT 接続のキープアライブ間隔を設定します。
event_manager_upper_bound	デフォルト値は 500 です。	不可	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
```

デバイスをオンボードし、ハブで運用する

マネージド型モノを作成してハブに接続することで、マネージド型統合ハブにデバイスをオンボードするようにセットアップします。デバイスは、シンプルなセットアップまたはユーザーガイドによるセットアップのいずれかを通じてハブにオンボードできます。

トピック

- デバイスのオンボードと運用のための簡単なセットアップ

- デバイスをオンボードして運用するためのユーザーガイドのセットアップ

デバイスのオンボードと運用のための簡単なセットアップ

マネージド型モノを作成してハブに接続することで、マネージド型統合ハブにデバイスをオンボードするようにセットアップします。このセクションでは、シンプルなセットアップを使用してデバイスのオンボーディングプロセスを完了する手順について説明します。

前提条件

デバイスのオンボードを試みる前に、以下の手順を実行します。

- ハブデバイスをマネージド統合ハブにオンボードします。
- Managed Integrations コマンドリファレンス AWS CLI から最新バージョンの をインストールする AWS CLI
- DEVICE_Life_CYCLE イベント通知をサブスクライブします。

セットアップステップ

- ステップ 1: 認証情報ロッカーを作成する
- ステップ 2: 認証情報ロッカーをハブに追加する
- ステップ 3: 認証情報を使用してマネージド型モノを作成する。
- ステップ 4: デバイスをプラグインし、そのステータスを確認します。
- ステップ 5: デバイス機能を取得する
- ステップ 6: マネージドモノにコマンドを送信する
- ステップ 7: ハブからマネージド型モノを削除する

ステップ 1: 認証情報ロッカーを作成する

デバイスの認証情報ロッカーを作成します。

認証情報ロッカーを作成するには

- `create-credential-locker` コマンドを使用します。このコマンドを実行すると、Wi-Fi セットアップキーペアとデバイス証明書を含むすべての製造リソースの作成がトリガーされます。

create-credential-locker 例

```
aws iot-managed-integrations create-credential-locker \
  --name "DEVICE_NAME"
```

レスポンス:

```
{
  "Id": "LOCKER_ID"
  "Arn": "arn:aws:iotmanagedintegrations:AWS_REGION:AWS_ACCOUNT_ID:credential-
locker/LOCKER_ID
  "CreatedAt": "2025-06-09T13:58:52.977000+08:00"
}
```

詳細については、「 マネージド統合コマンドリファレンス」のcreate-credential-locker コマンド AWS CLI 」を参照してください。

ステップ 2: 認証情報ロッカーをハブに追加する

認証情報ロッカーをハブに追加します。

認証情報ロッカーをハブに追加するには

- 次のコマンドを使用して、認証情報ロッカーをハブに追加します。

```
aws iotmi --region AWS_REGION --endpoint AWS_ENDPOINT update-managed-thing \
--identifier "HUB_MANAGED_THING_ID" --credential-locker-id "LOCKER_ID"
```

ステップ 3: 認証情報を使用してマネージド型モノを作成する。

デバイスの認証情報を使用してマネージド型モノを作成します。各デバイスには、独自のマネージド型モノが必要です。

マネージド型モノを作成するには

- `create-managed-thing` コマンドを使用して、デバイスのマネージド型モノを作成します。

  create-managed-thing 例

```
#ZWAVE:
aws iot-managed-integrations create-managed-thing --role DEVICE \
--authentication-material '900137947003133...' \ #auth material from zwave qr code
--authentication-material-type ZWAVE_QR_BAR_CODE \
--credential-locker-id ${locker_id}

#ZIGBEE:
aws iot-managed-integrations create-managed-thing --role DEVICE \
--authentication-material 'Z:286...$I:A4DC00.' \ #auth material from zigbee qr code
--authentication-material-type ZIGBEE_QR_BAR_CODE \
--credential-locker-id ${locker_id}
```

> **Note**
>
> Z ウェーブデバイスと Zigbee デバイスには個別のコマンドがあります。

レスポンス:

```
{
  "Id": "DEVICE_MANAGED_THING_ID"
  "Arn": "arn:aws:iotmanagedintegrations:AWS_REGION:AWS_ACCOUNT_ID:managed-
thing/DEVICE_MANAGED_THING_ID"
  "CreatedAt": "2025-06-09T13:58:52.977000+08:00"
}
```

詳細については、create-managed-thing コマンド AWS CLI 」を参照してください。

ステップ 4: デバイスをプラグインし、そのステータスを確認します。

デバイスをプラグインし、そのステータスを確認します。

- `get-managed-thing` コマンドを使用して、デバイスのステータスを確認します。マネージドモノの ProvisioningStatus はアクティブである必要があります。ProvisioningStatus の詳細については、「デバイスプロビジョニング」を参照してください。

  get-managed-thing 例

```
#KINESIS NOTIFICATION:
```

```
{
    "version": "1.0.0",
    "messageId": "4ac684bb7f4c41adbb2eecc1e7991xxx",
    "messageType": "DEVICE_LIFE_CYCLE",
    "source": "aws.iotmanagedintegrations",
    "customerAccountId": "12345678901",
    "timestamp": "2025-06-10T05:30:59.852659650Z",
    "region": "us-east-1",
    "resources": ["XXX"],
    "payload": {
        "deviceDetails": {
            "id": "1e84f61fa79a41219534b6fd57052XXX",
            "arn": "XXX",
            "createdAt": "2025-06-09T06:24:34.336120179Z",
            "updatedAt": "2025-06-10T05:30:59.784157019Z"
        },
        "status": "ACTIVATED"
    }
}
aws iot-managed-integrations get-managed-thing \
--identifier :"DEVICE_MANAGED_THING_ID"
```

レスポンス:

```
{
  "Id": "DEVICE_MANAGED_THING_ID"
  "Arn": "arn:aws:iotmanagedintegrations:AWS_REGION:AWS_ACCOUNT_ID:managed-
thing/MANAGED_THING_ID"
  "CreatedAt": "2025-06-09T13:58:52.977000+08:00"
}
```

詳細については、get-managed-thing コマンド AWS CLI 」を参照してください。

ステップ 5: デバイス機能を取得する

get-managed-thing-capabilities コマンドを使用してエンドポイント ID を取得し、デバイスに対して可能なアクションのリストを表示します。

デバイスの機能を取得するには

- `get-managed-thing-capabilities` コマンドを使用して、エンドポイント ID を書き留めます。

  get-managed-thing-capabilties 例

  ```
  aws iotmi get-managed-thing-capabilities \
  --identifier "DEVICE_MANAGED_THING_ID"
  ```

  レスポンス:

  ```
  {
      "ManagedThingId": "1e84f61fa79a41219534b6fd57052cbc",
      "CapabilityReport": {
          "version": "1.0.0",
          "nodeId": "zw.FCB10009+06",
          "endpoints": [
              {
                  "id": "ENDPOINT_ID"
                  "deviceTypes": [
                      "On/Off Switch"
                  ],
                  "capabilities": [
                      {
                          "id": "matter.OnOff@1.4",
                          "name": "On/Off",
                          "version": "6",
                          "properties": [
                              "OnOff"
                          ],
                          "actions": [
                              "Off",
                              "On"
                          ],
                          "events": []
                      }
                      ...
  }
  ```

  詳細については、「 マネージド統合コマンドリファレンス」のget-managed-thing-capabilities AWS CLI 」コマンドを参照してください。

ステップ 6: マネージドモノにコマンドを送信する

send-managed-thing-command コマンドを使用して、管理対象モノにトグルアクションコマンドを送信します。

マネージドモノにコマンドを送信するには

- send-managed-thing-command コマンドを使用して、マネージドモノにコマンドを送信します。

  send-managed-thing-command 例

```
json=$(jq -cr '.|@json') <<EOF
[
  {
    "endpointId": "1",
    "capabilities": [
      {
        "id": "matter.OnOff@1.4",
        "name": "On/Off",
        "version": "1",
        "actions": [
          {
            "name": "Toggle",
            "parameters": {}
          }
        ]
      }
    ]
  }
]
EOF
aws iot-managed-integrations send-managed-thing-command \
--managed-thing-id "DEVICE_MANAGED_THING_ID" --endpoints "ENDPOINT_ID"
```

> **Note**
>
> この例では、jq cli を に使用しますが、endpointId文字列全体を渡すこともできます。

レスポンス:

```
{
"TraceId": "TRACE_ID"
}
```

詳細については、「 マネージド統合コマンドリファレンス」のsend-managed-thing-command AWS CLI 」コマンドを参照してください。

ステップ 7: ハブからマネージド型モノを削除する

マネージドモノを削除してハブをクリーンアップします。

マネージドモノを削除するには

- `delete-managed-thing` コマンドを使用して、デバイスハブからマネージドモノを削除します。

  delete-managed-thing 例

  ```
  aws iot-managed-integrations delete-managed-thing \
  --identifier "DEVICE_MANAGED_THING_ID"
  ```

  詳細については、「 マネージド統合コマンドリファレンス」のdelete-managed-thing コマンド AWS CLI 」を参照してください。

  > **Note**
  >
  > デバイスが DELETE_IN_PROGRESS 状態でスタックしている場合は、 `--force`フラグを に追加しますdelete-managed-thing command。

  > **Note**
  >
  > Z 波デバイスの場合は、 コマンドの実行後にデバイスをペアリングモードにする必要があります。

デバイスをオンボードして運用するためのユーザーガイドのセットアップ

デバイスをハブに接続するマネージド型モノを作成して、マネージド型統合ハブにオンボードするようにデバイスをセットアップします。このセクションでは、ユーザーガイドの設定を使用してデバイスのオンボーディングプロセスを完了する手順について説明します。

前提条件

デバイスのオンボードを試みる前に、以下の手順を実行します。

- ハブデバイスをマネージド統合ハブにオンボードします。
- Managed Integrations コマンドリファレンス AWS CLI から最新バージョンの をインストールする AWS CLI
- DEVICE_DISCOVERY-STATUS イベント通知をサブスクライブします。

ユーザーガイドの設定手順

- 前提条件: Z Wave デバイスでペアリングモードを有効にする
- ステップ 1: デバイス検出を開始する
- ステップ 2: 検出ジョブ ID をクエリする
- ステップ 3: デバイスのマネージドモノを作成する
- ステップ 4: マネージドモノをクエリする
- ステップ 5: マネージド型のモノの機能を取得する
- ステップ 6: マネージドモノにコマンドを送信する
- ステップ 7: マネージドモノの状態を確認する
- ステップ 8: ハブからマネージド型モノを削除する

前提条件: Z Wave デバイスでペアリングモードを有効にする

Z 波デバイスでペアリングモードを有効にします。ペアリングモードは Z-Wave デバイスごとに異なる可能性があるため、デバイスの手順を参照してペアリングモードを適切に設定してください。これは通常、ユーザーが押さなければならないボタンです。

ステップ 1: デバイス検出を開始する

ハブのデバイス検出を開始して、デバイスのオンボードに使用される検出ジョブ ID を取得します。

デバイス検出を開始するには

- start-device-discovery コマンドを使用して、検出ジョブ ID を取得します。

  start-device-discovery 例

```
#For Zigbee
  aws iot-managed-integrations start-device-discovery --discovery-type ZIGBEE \
  --controller-identifier HUB_MANAGED_THING_ID

  #For Zwave
  aws iot-managed-integrations start-device-discovery --discovery-type ZWAVE \
  --controller-identifier HUB_MANAGED_THING \
  --authentication-material-type ZWAVE_INSTALL_CODE \
  --authentication-material 13333

  #For Cloud
  aws iot-managed-integrations start-device-discovery --discovery-type CLOUD \
  --account-association-id C2C_ASSOCIATION_ID \

  #For Custom
  aws iot-managed-thing start-device-discovery --discovery-type CUSTOM \
  --controller-identifier HUB_MANAGED_THING_ID \
  --custom-protocol-detail NAME : NON_EMPTY_STRING \
```

  レスポンス:

```
{
      "Id": DISCOVERY_JOB_ID,
      "StartedAt": "2025-06-03T14:43:12.726000-07:00"
  }
```

> **Note**
>
> Z ウェーブデバイスと Zigbee デバイスには個別のコマンドがあります。

  詳細については、「 マネージド AWS CLI integrationsCommand リファレンス」のstart-device-discovery API」を参照してください。

ステップ 2: 検出ジョブ ID をクエリする

`list-discovered-devices` コマンドを使用して、デバイスの認証マテリアルを取得します。

検出ジョブ ID をクエリするには

- list-discovered-devices コマンドで検出ジョブ ID を使用して、デバイスの認証マテリアルを取得します。

```
aws iot-managed-integrations list-discovered-devices --identifier DISCOVERY_JOB_ID
```

レスポンス:

```
"Items": [
    {
        "DeviceTypes": [],
        "DiscoveredAt": "2025-06-03T14:43:37.619000-07:00",
        "AuthenticationMaterial": AUTHENTICATION_MATERIAL
    }
]
```

ステップ 3: デバイスのマネージドモノを作成する

`create-managed-thing` コマンドを使用して、デバイスのマネージド型モノを作成します。各デバイスには、独自のマネージド型モノが必要です。

マネージド型モノを作成するには

- `create-managed-thing` コマンドを使用して、デバイスのマネージド型モノを作成します。

  create-managed-thing 例

```
aws iot-managed-integrations create-managed-thing \
  --role  DEVICE --authentication-material-type DISCOVERED_DEVICE \
  --authentication-material "AUTHENTICATION_MATERIAL"
```

  レスポンス:

```
{
    "Id": "DEVICE_MANAGED_THING_ID"
```

```
    "Arn": "arn:aws:iotmanagedintegrations:AWS_REGION:AWS_ACCOUNT_ID:managed-
thing/DEVICE_MANAGED_THING_ID"
    "CreatedAt": "2025-06-09T13:58:52.977000+08:00"
  }
```

詳細については、create-managed-thing コマンド AWS CLI 」を参照してください。

ステップ 4: マネージドモノをクエリする

コマンドを使用して、マネージド型モノがアクティブ化されているかどうかを確認できますget-managed-thing。

マネージドモノをクエリするには

- get-managed-thing コマンドを使用して、マネージドモノのプロビジョニングステータスがに設定されているかどうかを確認しますACTIVATED。プロビジョニングステータスの詳細については、「デバイスプロビジョニング」を参照してください。

  get-managed-thing 例

  ```
  aws iot-managed-integrations get-managed-thing \
    --identifier "DEVICE_MANAGED_THING_ID"
  ```

  レスポンス:

  ```
  {
        "Id": "DEVICE_MANAGED_THING_ID",
        "Arn": "arn:aws:iotmanagedintegrations:AWS_REGION:AWS_ACCOUNT_ID:managed-
  thing/DEVICE_MANAGED_THING_ID,
        "Role": "DEVICE",
        "ProvisioningStatus": "ACTIVATED",
        "MacAddress": "MAC_ADDRESS",
        "ParentControllerId": "PARENT_CONTROLLER_ID",
        "CreatedAt": "2025-06-03T14:46:35.149000-07:00",
        "UpdatedAt": "2025-06-03T14:46:37.500000-07:00",
        "Tags": {}
    }
  ```

  詳細については、get-managed-thing コマンド AWS CLI 」を参照してください。

ステップ 5: マネージド型のモノの機能を取得する

を使用して、マネージドモノの使用可能なアクションのリストを表示できますget-managed-thing-capabiltiies。

デバイスの機能を取得するには

- get-managed-thing-capabilities コマンドを使用してエンドポイント ID を取得します。また、可能なアクションのリストも書き留めます。

  get-managed-thing-capabilities 例

```
aws iot-managed-integrations get-managed-thing-capabilities \
  --identifier "DEVICE_MANAGED_THING_ID"
```

  レスポンス:

```
{
      "ManagedThingId": "DEVICE_MANAGED_THING_ID",
      "CapabilityReport": {
          "version": "1.0.0",
          "nodeId": "zb.539D+4A1D",
          "endpoints": [
              {
                  "id": "1",
                  "deviceTypes": [
                      "Unknown Device"
                  ],
                  "capabilities": [
                      {
                          "id": "matter.OnOff@1.4",
                          "name": "On/Off",
                          "version": "6",
                          "properties": [
                              "OnOff",
                              "OnOff",
                              "OnTime",
                              "OffWaitTime"
                          ],
                          "actions": [
                              "Off",
                              "On",
```

```
                            "Toggle",
                            "OffWithEffect",
                            "OnWithRecallGlobalScene",
                            "OnWithTimedOff"
                        ],
                        ...
  }
```

詳細については、「 get-managed-thing-capabilities AWS CLI」コマンドを参照してください。

ステップ 6: マネージドモノにコマンドを送信する

send-managed-thing-command コマンドを使用して、トグルアクションコマンドをマネージドモノに送信できます。

トグルアクションを使用して、マネージドモノにコマンドを送信します。

- send-managed-thing-command コマンドを使用して、トグルアクションコマンドを送信します。

  send-managed-thing-command 例

```
json=$(jq -cr '.|@json') <<EOF
  [
    {
      "endpointId": "1",
      "capabilities": [
        {
          "id": "matter.OnOff@1.4",
          "name": "On/Off",
          "version": "1",
          "actions": [
            {
              "name": "Toggle",
              "parameters": {}
            }
          ]
        }
      ]
    }
  ]
  EOF
```

```
  aws iot-managed-integrations send-managed-thing-command \
  --managed-thing-id ${device_managed_thing_id} --endpoints ENDPOINT_ID
```

> **Note**
>
> この例では、jq cli を に使用しますが、endpointId文字列全体を渡すこともできます。

レスポンス:

```
{
  "TraceId": TRACE_ID
  }
```

詳細については、「 マネージド統合コマンドリファレンス」のsend-managed-thing-command AWS CLI 」コマンドを参照してください。

ステップ 7: マネージドモノの状態を確認する

マネージドモノの状態をチェックして、トグルアクションが成功したことを確認します。

マネージドモノのデバイスの状態を確認するには

- get-managed-thing-state コマンドを使用して、成功したトグルアクションを検証します。

  get-managed-thing-state 例

```
  aws iot-managed-integrations get-managed-thing-state --managed-thing-
id DEVICE_MANAGED_THING_ID
```

  レスポンス:

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

```
            "endpointId": "1",
            "capabilities": [
                {
                    "id": "matter.OnOff@1.4",
                    "name": "On/Off",
                    "version": "1.4",
                    "properties": [
                        {
                            "name": "OnOff",
                            "value": {
                                "propertyValue": true,
                                "lastChangedAt": "2025-06-03T21:50:39.886Z"
                            }
                        }
                    ]
                }
            ]
        }
    ]
}
```


詳細については、「 マネージド統合コマンドリファレンス」のget-managed-thing-state」コマンドを参照してください。 AWS CLI

ステップ 8: ハブからマネージド型モノを削除する

マネージドモノを削除してハブをクリーンアップします。

マネージドモノを削除するには

- deletedelete-managed-thing コマンドを使用して、マネージド型モノを削除します。

  delete-managed-thing 例

  ```
  aws iot-managed-integrations delete-managed-thing \
    --identifier MANAGED_THING_ID
  ```

  詳細については、「 マネージド統合コマンドリファレンス」のdelete-managed-thing コマンド AWS CLI 」を参照してください。

> **Note**
>
> デバイスが DELETE_IN_PROGRESS状態でスタックしている場合は、 delete-managed-thing コマンドに --forceフラグを追加します。

> **Note**
>
> Z 波デバイスの場合は、 コマンドの実行後にデバイスをペアリングモードにする必要があります。

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

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"
  }
}
```

カスタムプロトコルプラグイン

カスタムプロトコルプラグインを使用して、独自の IoT プロトコルを AWS IoT Device Management エコシステムのマネージド統合に統合できます。明確に定義された SDK インターフェイスを使用すると、マネージド統合とハブ SDK コンポーネントとの完全互換性を維持しながら、デバイスのオンボード、機能の定義、リアルタイム制御フローの処理を行うことができます。

次のリストでは、カスタムプロトコルプラグインの主な機能について説明します。

データモデルのカスタマイズ

独自の AWS データモデルスキーマを定義し、プロビジョニングフロー中にマネージド統合にアップロードします。これらのスキーマは、後でワークフローで使用できます。

柔軟なプラグイン実装

- を使用して独自のプラグインコンポーネントを構築しますHub SDK クライアント。
- プロビジョニングや制御など、さまざまな機能用に個別のプラグインを実装するか、両方の統合クライアントを作成します。
- マネージド統合アセットとミドルウェアスタックなどの独自のアセットとの間に明確な境界を維持し、分離された開発しやすいコードロジックを実装します。

下位互換性

既存のお客様は、既存のラジオタイプをそのまま動作させながら、新しいカスタムプロトコルをスムーズにオンボードできます。

次の図は、カスタムプロトコルプラグインアーキテクチャを示しています。

Hub SDK クライアント

Hub SDK クライアントライブラリは、マネージド統合 Hub SDK と、同じハブで実行されている独自のプロトコルスタックとの間のインターフェイスとして機能します。これにより、一連のパブリック APIs公開され、プロトコルスタックと Device Hub SDK コンポーネントとのやり取りが容易になります。ユースケースには、カスタムプラグインコントロール、カスタムプラグインプロビジョナー、ローカルコントローラーが含まれます。

トピック

- マネージド統合 Hub SDK を取得する
- Hub SDK ツールキットについて
- Hub SDK クライアントを使用してカスタムアプリケーションを作成する
- カスタムアプリケーションの実行
- Hub SDK クライアント API リファレンス
- データ型

マネージド統合 Hub SDK を取得する

Hub SDK クライアントには、 マネージド統合 SDK が付属しています。ハブ SDK にアクセスするには、 マネージド統合コンソールからお問い合わせください。

Hub SDK ツールキットについて

ダウンロードすると、アプリケーションで使用できるすべてのパブリックヘッダーファイルと.soファイルを含むIotMI-DeviceSDK-Toolkitフォルダが表示されます。マネージド統合チームには、デモmain.cpp用の例と、直接実行できる bin/ のデモアプリケーションバイナリも用意されています。オプションで、これをアプリケーションの開始点として使用できます。

Hub SDK クライアントを使用してカスタムアプリケーションを作成する

次の手順を使用して、カスタムアプリケーションを作成します。

1. ヘッダーファイル (.h) と共有オブジェクトファイル (.so) をアプリケーションに含めます。

   アプリケーションには、パブリックヘッダーファイル (.h) と共有オブジェクトファイル (.so) を含める必要があります。.so ファイルの場合は、lib フォルダに配置することができます。最終的なレイアウトは次のようになります。

   ```
   ### include
   #   ### iotmi_device_sdk_client
   #   #   ### iotmi_device_sdk_client_common_types.h
   #   ### iotmi_device_sdk_client.h
   #   ### iotshd_status.h
   ### lib
   #   ### libiotmi_devicesdk_client_module.so
   #   ### libiotmi_log_c.so
   ```

2. メインアプリケーションで Hub SDK クライアントを作成します。

   a. メインアプリケーションでは、リクエストの処理に使用する前に、まず Hub SDK クライアントを初期化する必要があります。クライアントは、 を使用して簡単に構築できますclientId。

   b. クライアントを取得したら、マネージド統合 Device SDK に接続できます。

      以下は、Hub SDK クライアントの作成方法と接続方法の例です。

      ```
      #include <cstdlib>
      #include <string>
      #include "iotshd_status.h"
      #include "iotmi_device_sdk_client.h"

      auto client = std::make_unique<DeviceSDKClient>(your_own_clientId);
      ```

```
iotmi_statusCode_t status = client->connect();
```

> **Note**
>
> *your_own_clientId* は、ユーザーガイド付きセットアップで start-device-discovery で指定するもの、または簡易セットアッププロビジョニングフローで create-managed-thing で指定するものと同じである必要があります。

3. 次の手順を実行して、パブリッシュおよびサブスクライブします。

   a. 接続が確立されたら、マネージド統合 Hub SDK から受信タスクをサブスクライブできるようになりました。受信タスクは、制御タスクでもプロビジョニングタスクでもかまいません。また、受信したタスクに対して独自のコールバック関数を定義し、独自のトレース目的で独自のカスタムコンテキストを定義する必要があります。

```
    // subscribe to provisioning tasks
    iotmi_statusCode_t status = client->iotmi_provision_subscribe_to_tasks(
    example_subscriber_callback, custom_context);

    // subscribe to control tasks
    iotmi_statusCode_t status = client->iotmi_control_subscribe_to_tasks(
    example_subscriber_callback, custom_context);
```

   b. 接続が確立されたら、アプリケーションからマネージドインテグレーション Hub SDK にリクエストを発行できるようになりました。独自のタスクメッセージタイプを定義し、ビジネス目的に応じてペイロードを変更できます。リクエストには、サブスクライブフローと同様に、コントロールリクエストとプロビジョニングリクエストの両方を含めることができます。最後に、 のアドレスを割り当てrspPayloadて、 マネージド統合 Hub SDK から同期された方法でレスポンスを取得できます。

```
// publish control request
  iotmi_client_request_t api_payload = {
            .messageType = C2MIMessageType::C2MI_CONTROL_EVENT,
            .reqPayload = (uint8_t *)"define_your_req_payload",
            .rspPayload = (uint8_t *)calloc(1000, sizeof(uint8_t))
  };

  status = client->iotmi_control_publish_request(&api_payload);

  // publish provision request
```

```
iotmi_client_request_t api_payload = {
          .messageType = C2MIMessageType::C2MI_DEVICE_ONBOARDED,
          .reqPayload = (uint8_t *)"define_your_req_payload",
          .rspPayload = (uint8_t *)calloc(1000, sizeof(uint8_t))
};
status = client->iotmi_provision_publish_request(&api_payload);
```

4. 独自の を構築しCMakeLists.txt、そこからアプリケーションを構築します。最終出力は、次のような実行可能バイナリにすることができます。 MyFirstApplication

カスタムアプリケーションの実行

カスタムアプリケーションを実行する前に、次の手順を実行してハブをセットアップし、マネージド統合 Hub SDK を起動します。

- 「」のオンボーディング手順に従ってくださいマネージド統合にハブをオンボードする。
- 「」に記載されているインストールプロセスを完了しますマネージド統合 Hub SDK をインストールして検証する。

前提条件が満たされたら、カスタムアプリケーションを実行できます。例:

```
./MyFirstApplication
```

> **Important**
>
> にリストされている開始スクリプトをスクリプトスクリプトを使用して Hub SDK をデプロイするで手動で更新して、独自のアプリケーションを起動する必要があります。順序は重要です。順序は変更しないでください。
> 以下を更新します。変更
>
> ```
> ./IotMI-DeviceSDK-Toolkit/bin/DeviceSDKClientDemo >> $LOGS_DIR/
> logDeviceSDKClientDemo_logs.txt &
> ```
>
> 次のように変更します。
>
> ```
> ./MyFirstApplication >> $LOGS_DIR/MyFirstApplication_logs.txtt &
> ```

Hub SDK クライアント API リファレンス

Hub SDK クライアント (DeviceSDKClient クラス) は、カスタムアプリケーションがマネージド統合 Device SDK とやり取りするためのインターフェイスを提供します。このクライアントでは、以下を実行できます。

- マネージド統合コンポーネントからプロビジョニング関連タスクとコントロール関連タスクをサブスクライブします。
- プロビジョニング関連およびコントロール関連のリクエストをマネージド統合コンポーネントに発行します。

AWS IoT Device Management APIs」を参照してください。 AWS Lambda

トピック

クライアントの初期化

の使用を開始するにはDeviceSDKClient、クライアント ID を使用して初期化します。

```
iotmi_statusCode_t DeviceSDKClient(const std::string& clientId)
```

これにより、指定された で新しいDeviceSDKClientインスタンスが作成されますclientId。は、マネージド統合に登録する と一致するclientId必要があります。

パラメータ

clientId (文字列) - このインスタンスのクライアント ID。

```
connect()
```

DeviceSDKClient インスタンスをマネージド統合に接続します。

戻り値

- IOTMI_STATUS_OK - 接続は成功しました。
- IOTMI_STATUS_CUSTOM_PLUGIN_CONNECTION_ERROR - マネージド統合への接続中にエラーが発生しました。

タスクサブスクリプションのプロビジョニング

これらの方法を使用して、マネージド統合コンポーネントからプロビジョニング関連のタスクをサブスクライブします。

```
iotmi_statusCode_t
 iotmi_provision_subscribe_to_tasks(DeviceSDKClient_SubscriberCallback callback, char*
 context)
```

マネージド統合コンポーネントから、デバイスのオンボーディングやプロビジョニング解除などのプロビジョニング関連のタスクをサブスクライブします。

パラメータ

- callback (DeviceSDKClient_SubscriberCallback) - タスクを受信したときに実行されるコールバック関数。
- context (char*) - コールバック関数に渡されるカスタムコンテキスト。

戻り値

- IOTMI_STATUS_OK - サブスクリプションは成功しました。
- IOTMI_STATUS_CUSTOM_PLUGIN_CLIENT_NOT_CONNECTED - DeviceSDKClient インスタンスがマネージド統合に接続されていません。
- IOTMI_STATUS_CUSTOM_PLUGIN_SUBSCRIBE_ERROR - タスクのサブスクライブ中にエラーが発生しました。

プロビジョニングタスクの公開

これらの方法を使用して、マネージド統合コンポーネントにプロビジョニング関連のリクエストを発行します。

```
iotmi_statusCode_t iotmi_provision_publish_request(DataModel::iotmi_client_request_t
 request)
```

マネージド統合コンポーネントにプロビジョニング関連のリクエストを発行します。デバイスのオンボードイベントやプロビジョニング解除ステータスなど

パラメータ

`request` (DataModel::iotmi_client_request_t) - 詳細を含むリクエスト構造へのポインタ。

戻り値

- `IOTMI_STATUS_OK` - リクエストが正常に発行されました。
- `IOTMI_STATUS_CUSTOM_PLUGIN_CLIENT_NOT_CONNECTED` - `DeviceSDKClient`インスタンスがマネージド統合に接続されていません。
- `IOTMI_STATUS_INVALID_PARAMETER` - リクエスト内の 1 つ以上のパラメータが無効です。
- `IOTMI_STATUS_INVALID_JSON_OBJECT` - リクエストペイロードが有効な JSON オブジェクトではありません。
- `IOTMI_STATUS_NO_MEMORY` - メモリ割り当てエラーが発生しました。

タスクサブスクリプションの制御

これらの方法を使用して、マネージド統合コンポーネントからコントロール関連のタスクをサブスクライブします。

```
iotmi_statusCode_t iotmi_control_subscribe_to_tasks(DeviceSDKClient_SubscriberCallback
 callback, char context)
```

マネージド統合コンポーネントからコントロール関連のタスク (デバイスコントロールリクエストなど) をサブスクライブします。

パラメータ

- `callback` (DeviceSDKClient_SubscriberCallback) - タスクを受信したときに実行されるコールバック関数。
- `context` (char) - コールバック関数に渡されるカスタムコンテキスト。

戻り値

- `IOTMI_STATUS_OK` - サブスクリプションは成功しました。

- IOTMI_STATUS_CUSTOM_PLUGIN_CLIENT_NOT_CONNECTED - DeviceSDKClientインスタンスがマネージド統合に接続されていません。
- IOTMI_STATUS_CUSTOM_PLUGIN_SUBSCRIBE_ERROR - タスクのサブスクライブ中にエラーが発生しました。

タスクの発行を制御する

これらのメソッドを使用して、管理統合コンポーネントにコントロール関連のリクエストを発行します。

```
iotmi_statusCode_t iotmi_control_publish_request(DataModel::iotmi_client_request_t
 request)
```

管理統合コンポーネントにコントロール関連のリクエストを発行します。例えば、未承諾イベント、コマンドリクエスト、デバイス状態クエリなどです。

パラメータ

`request` (DataModel::iotmi_client_request_t) - 詳細を含むリクエスト構造へのポインタ。

戻り値

- IOTMI_STATUS_OK - リクエストが正常に発行されました。
- IOTMI_STATUS_CUSTOM_PLUGIN_CLIENT_NOT_CONNECTED - DeviceSDKClient インスタンスがマネージド統合に接続されていません。
- IOTMI_STATUS_INVALID_PARAMETER - リクエスト内の 1 つ以上のパラメータが無効です。
- IOTMI_STATUS_INVALID_JSON_OBJECT - リクエストペイロードが有効な JSON オブジェクトではありません。
- IOTMI_STATUS_NO_MEMORY - メモリ割り当てエラーが発生しました。

ログ記録機能

これらの方法を使用して、 マネージド統合が提供するログ記録機能を実装します。

ロガーの初期化

```
void iotmi_devicesdk_log_init(const char* logger_name)
```

ログ記録機能を使用する前に、ロガーを初期化する必要があります。

パラメータ

`logger_name` - 指定したロガー名。デフォルト値は次のとおりです。 `MyApplication`

マクロのログ記録

`LOGGER_LOGD(...)`

DEBUG レベルのログ記録には、アプリケーションでこのマクロを使用します。

`LOGGER_LOGI(...)`

INFO レベルのログ記録には、アプリケーションでこのマクロを使用します。

`LOGGER_LOGW(...)`

アプリケーションでこのマクロを WARN レベルのログ記録に使用します。

`LOGGER_LOGE(...)`

アプリケーションでこのマクロを ERROR レベルのログ記録に使用します。

> **Note**
>
> ログ記録機能の詳細については、「Hub logging documentation」を参照してください。カスタムプロトコルプラグインは、 マネージド統合が提供するすべてのログ記録機能を完全にサポートします。

その他の API

```
std::string get_client_id()
```

DeviceSDKClient インスタンスに関連付けられたクライアント ID を返します。

戻り値

クライアント ID。

データ型

このセクションでは、カスタムプロトコルプラグインに使用されるデータ型を定義します。

iotmi_client_request_t

マネージド統合コンポーネントに発行されるリクエストを表します。

messageType

メッセージのタイプ (CommonTypes::C2MIMessageType)。次のリストは有効な値を示しています。

- C2MI_DEVICE_ONBOARDED: 関連するペイロードを含むデバイスのオンボーディングメッセージを示します。
- C2MI_DE_PROVISIONING_PRE_ASSOCIATED_COMPLETE: 事前関連付けられたデバイスのプロビジョニング解除タスク完了通知を示します。
- C2MI_DE_PROVISIONING_ACTIVATED_COMPLETE: アクティブ化されたデバイスのプロビジョニング解除タスク完了通知を示します。
- C2MI_DE_PROVISIONING_COMPLETE_RESPONSE: プロビジョニング解除タスクの完了レスポンスを示します。
- C2MI_CONTROL_EVENT: デバイスの状態が変化する可能性のあるコントロールイベントを示します。
- C2MI_CONTROL_SEND_COMMAND: ローカルコントローラーからのコントロールコマンドを示します。
- C2MI_CONTROL_SEND_DEVICE_STATE_QUERY: ローカルコントローラーからのコントロールデバイスの状態クエリを示します。

reqPayload

リクエストペイロード。通常は JSON 形式の文字列です。

rspPayload

マネージド統合コンポーネントによって入力されたレスポンスペイロード。

iotmi_client_event_t

マネージド統合コンポーネントから受信したイベントを表します。

event_id

イベントの一意の識別子。

length

イベントデータの長さ。

データ

を含むイベントデータへのポインタmessageType。次のリストは、可能な値を示しています。

- C2MI_PROVISION_UGS_TASK: UGS フローのプロビジョニングタスクを示します。
- C2MI_PROVISION_SS_TASK: SimpleSetup フローのプロビジョニングタスクを示します。
- C2MI_DE_PROVISION_PRE_ASSOCIATED_TASK: 事前関連付けられたデバイスのプロビジョニング解除タスクを示します。
- C2MI_DE_PROVISION_ACTIVATED_TASK: アクティブ化されたデバイスのプロビジョニング解除タスクを示します。
- C2MI_DEVICE_ONBOARDED_RESPONSE: デバイスのオンボーディングレスポンスを示します。
- C2MI_CONTROL_TASK: コントロールタスクを示します。
- C2MI_CONTROL_EVENT_NOTIFICATION: ローカルコントローラーのコントロールイベント通知を示します。

ctx

イベントに関連付けられたカスタムコンテキスト。

ハブコントロール

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。ハブコントロールを使用すると、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、 というラベルが付けられた Hub SDK 内の別のパッケージとして提供されますiot-managed-integrations-hub-control-x.x.x。

トピック

- 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、以下が必要です。

- Hub SDK バージョン 0.4.0 以降にオンボードされたハブ。
- から End Device SDK の最新バージョンをダウンロードします AWS マネジメントコンソール。
- 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 の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。

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

End Device 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の C 機能 APIs。

```
struct DeviceState
{
  struct iotmiDev_Agent *agent;
  struct iotmiDev_Endpoint *endpointLight;
  /* This simulates the HW state of OnOff */
  bool hwState;
};

/* This implementation for OnOff getter just reads
   the state from the DeviceState */
iotmiDev_DMStatus exampleGetOnOff(bool *value, void *user)
{
   struct DeviceState *state = (struct DeviceState *)(user);
  *value = state->hwState;
  return iotmiDev_DMStatusOk;
}
```

c. 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);
}
```

例: ハブコントロールを構築する

ハブコントロールは、Hub SDK パッケージの一部として提供されます。ハブコントロールのサブパッケージには というラベルが付けられiot-managed-integrations-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
```

データモデルマネージド統合データモデルについては、「」を参照してください。のステップ 5 に従ってエンドポイントEnd Device SDK の使用を開始するを設定し、エンドユーザーと iot-managed-integrations 間の通信を管理します。

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

次の表は、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ています。

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

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、「ハブ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

```
aws iot-managed-integrations put-runtime-log-configuration \
    --managed-thing-id MANAGED_THING_ID \
    --runtime-log-configurations LogLevel=DEBUG,UploadLog=TRUE
```

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。

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/RGB ライト	OnOff、LevelControl、ColorControl
スマートプラグ	OnOff

Zigbee デバイスタイプ	サポートされる機能
スマートスイッチ	OnOff
LED ストリップ	OnOff、LevelControl、ColorControl
水バルブ	OnOff
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
ガレージドアコントローラー	OnOff、LevelControl

Z-Wave デバイスタイプ	サポートされる機能
エネルギーメーター	ElectricalEnergyMeasurement、ElectricalPowerMeasurement
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

Raspberry Pi でマネージド統合を実行する

Note

Raspberry Pi AWS IoT での Hub SDK のこの実装は、学習とテストのみを目的としており、本番環境での使用を意図していません。このデモでは、開発を容易にするために以下の設定を行います。
AWS 認証情報ストレージ: デモのみを目的として、認証情報と証明書は、テストと開発を容易にするために、アクセス可能な場所に保存されます。本番環境では AWS Secrets Manager、 や Systems Manager Parameter Store などの安全なストレージソリューションを使用する必要があります。保管時の暗号化を実装し、 AWS IoT セキュリティガイドラインに従う必要があります。
コンテナ権限: デモは昇格された権限で実行され、ホストリソースへの無制限アクセスを許可し、開発ワークフローを簡素化します。本番環境では、コンテナは最小限必要な権限で動作する必要があります。
ネットワークブリッジ設定: デモでは、デバッグとモニタリングを容易にするために内部ネットワークトラフィックを公開するネットワークブリッジ設定を使用します。本番環境

では、適切なネットワーク分離とセグメンテーションを実装して、内部ネットワークトラフィックへの不正アクセスを防止します。
USB デバイスのアクセス許可: 開発周辺機器とテストデバイスの接続を容易にするために、無制限の USB デバイスアクセスが有効になっています。本番環境では、厳格な USB デバイスコントロールと検証を実装して、デバイスのなりすまし攻撃を防止します。
これらの設定により、簡単なテストが可能になり、本番環境で使用してはいけません。本番環境にデプロイする場合は、セキュリティのベストプラクティスに従って、ホストシステムの侵害や認証情報への不正アクセスを防止してください。

前提条件として、Raspberry Pi を設定する前に Sonoff Zigbee USB ドングルを設定する必要があります。

ファームウェアを Sonoff Zigbee USB ドングルにフラッシュする

前提条件

- Sonoff Zigbee USB ドングル
- Windows: CP210x ユニバーサル Windows ドライバーをインストールする

ファームウェアをフラッシュする

1. Zigbee Dongle Firmware Build 7.4.1.0 をダウンロードします。
2. Silabs Firmware Flasher を開きます。
3. Sonoff Zigbee USB ドングルをコンピュータに接続します。
4. ZBDongle-E をスクロールして検索します。
5. [接続] を選択します。
6. デバイスが接続されるまで待ちます。
7. ファームウェアの変更を選択します。
8. 独自のファームウェアをアップロードを選択します。
9. Zigbee Dongle Firmware Build 7.4.1.0 ダウンロードの場所を見つけて選択します。
10. [Install] (インストール) をクリックします。
11. ファームウェアのインストールを待ちます。

12. インストールが完了したら続行を選択します。

これで、ドングルを使用する準備ができました。

Raspberry Pi でマネージド統合 Hub SDK を実行するには、以下のオプションから選択します。両方のアプローチのセットアップと検証のステップを以下に示します。

トピック

Raspberry Pi のマネージド統合 Hub SDK イメージ

> **Note**
>
> Raspberry Pi AWS IoT での Hub SDK のこの実装は、学習とテストのみを目的としており、本番環境での使用を意図していません。このデモでは、開発を容易にするために以下の設定を行います。
> AWS 認証情報ストレージ: デモのみを目的として、認証情報と証明書は、テストと開発を容易にするために、アクセス可能な場所に保存されます。本番環境では AWS Secrets Manager、 や Systems Manager Parameter Store などの安全なストレージソリューションを使用する必要があります。保管時の暗号化を実装し、 AWS IoT セキュリティガイドラインに従う必要があります。
> コンテナ権限: デモは昇格された権限で実行され、ホストリソースへの無制限アクセスを許可し、開発ワークフローを簡素化します。本番環境では、コンテナは最小限必要な権限で動作する必要があります。
> ネットワークブリッジ設定: デモでは、デバッグとモニタリングを容易にするために内部ネットワークトラフィックを公開するネットワークブリッジ設定を使用します。本番環境では、適切なネットワーク分離とセグメンテーションを実装して、内部ネットワークトラフィックへの不正アクセスを防止します。
> USB デバイスのアクセス許可: 開発周辺機器とテストデバイスの接続を容易にするために、無制限の USB デバイスアクセスが有効になっています。本番環境では、厳格な USB デバイスコントロールと検証を実装して、デバイスのなりすまし攻撃を防止します。

これらの設定により、簡単なテストが可能になり、本番環境で使用してはいけません。本番環境にデプロイする場合は、セキュリティのベストプラクティスに従って、ホストシステムの侵害や認証情報への不正アクセスを防止してください。

前提条件

Raspberry Pi イメージをデプロイする前に、以下の要件を満たす必要があります。

- Raspberry Pi イメージャをダウンロードしてインストールします。
- SD カードを取得します。
- 2.4Ghz 64 ビットクアッドコア CPU (8GB RAM) を搭載した Raspberry Pi 5 をセットアップします。
- Sonoff Zigbee USB ドングルを接続します。
- ファームウェアを Sonoff Zigbee USB ドングルにフラッシュする.
- Silicon Labs SLUSB001A ドングルを接続します。
- AWS アカウントにサインアップします。
- AWS CLI マネージド統合コマンドリファレンスから最新バージョンの AWS CLI をインストールします。

新しい SD カードで Raspberry Pi イメージをフラッシュする

以下の手順に従って、マネージド統合イメージを SD カードにフラッシュします。

1. マネージド統合 Raspberry Pi Hub SDK イメージをダウンロードします。
2. デスクトップで Raspberry Pi Imager を起動します。
3. コンピュータの組み込み SD カードリーダーまたは外部 USB カードリーダーに SD カードを挿入します。
4. デバイスの選択 → Raspberry Pi 5 を選択します。
5. Select OS → Use custom → Find the IotMI-HubSDK-RPi-Image-v1.0.0.img.gz file → Open を選択します。
6. Select storage → Select your SD Card Reader を選択します。
7. 設定が次の画面と一致することを確認します。

8. [次へ] をクリックします。
9. OS カスタマイズ設定を構成します。
   - ホスト名: raspberrypi を選択します。
   - ユーザー名とパスワード:
     - Enable ユーザー名とパスワードの設定:
     - ユーザー名: には、 を入力しますhub123456。
     - パスワード: には、 を入力しますsh123456。
   - ワイヤレス LAN:
     - ワイヤレス LAN の設定を有効にします。
     - ルーターの SSID とパスワードを入力します。

       設定の例

       - SSID:iotmi-tplink
       - パスワード:******** (8 文字以上)
     - 国: を に設定しますUS。
   - ロケール設定の設定:
     - タイムゾーン を に設定しますAmerica/Los Angeles。
     - キーボードレイアウト を に設定しますUS。
   - SSH:
     - サービスタブを選択します。
     - SSH を有効にする を確認します。
     - パスワード認証を使用する を選択します。
10. OS のカスタマイズとデータ消去のすべてのポップアップを確認します。
11. 書き込みプロセスが完了するまで待ちます。
12. 次の画面で正常に完了したことを確認します。

13. [次へ] をクリックします。
14. SD カードを取り外し、Raspberry Pi に挿入します。

Raspberry Pi で Hub SDK を実行する

設定した Raspberry Pi で Hub SDK サービスを起動します。

1. 準備した SD カードを Raspberry Pi 5 デバイスに挿入します。
2. Sonoff Zigbee USB ドングルと Silicon Labs SLUSB001A ドングルを Raspberry Pi に接続します。
3. Raspberry Pi の電源を入れます。
4. Raspberry Pi とコンピュータ (SSH を使用) が同じネットワーク上にあることを確認します。
5. イメージのデプロイ時に設定した認証情報を使用して、Raspberry Pi に SSH します。

```
ssh username@hostname
```

6. ハブ SDK ディレクトリに移動します。

```
cd /data/aws/iotmi
```

7. Hub オンボーディング設定を完了して、認証マテリアルと設定マテリアルを追加します。

> **Note**
>
> このステップを実行するには、 YULまたは DUBリージョンにいる必要があります。

8. Hub SDK を実行します。

```
cd /data/aws/iotmi
bash start_hub_sdk.sh
```

Hub SDK が正常に起動すると、次のレスポンスが表示されます。

```
-------Stopping SDK running processes---
-------Starting Hub SDK-------------
-------Creating logs directory----------
Logs directory created.
-------Verifying Middleware paths-------
All middleware libraries exist
-------Verifying Middleware pre reqs---
AIPC and KVstroage directories exist
-------Starting HubOnboarding-----------
-------Starting MQTT Proxy-----------
-------Staring Log Daemon---
-------Starting Event Manager-----------
-------Starting Zigbee Service----------
```

```
--Checking Zigbee network information--
-------Starting Zwave Service----------
/data/aws/iotmi/middleware/AceZwave/bin /data/aws/iotmi
/data/aws/iotmi
-------Starting CDMB--------------------
-------Starting Agent--------------------
-------Starting Provisioner--------------
-------Checking SDK status---------------
hub1234+    1780  0.2  0.1 1093936 16368 pts/1   Sl+  16:34   0:00 ./iotmi_mqtt_proxy -
C /data/aws/iotmi/config/iotmi_config.json
Process 'iotmi_mqtt_proxy' is running.
hub1234+    1884  0.0  0.0 236272  2624 pts/1    Sl+  16:34   0:00 ./middleware/
AceCommon/bin/ace_eventmgr
Process 'ace_eventmgr' is running.
hub1234+    1892  9.1  0.1 393040  8352 pts/1    Sl+  16:34   0:04 ./middleware/
AceZigbee/bin/ace_zigbee_service
Process 'ace_zigbee_service' is running.
hub1234+    1923  0.0  0.1 1570736 12736 pts/1   Sl+  16:34   0:00 ./zwave_svc
Process 'zwave_svc' is running.
hub1234+    1958  0.0  0.0 1067632 5776 pts/1    Sl+  16:34   0:00 ./iotmi_cdmb
Process 'iotmi_cdmb' is running.
hub1234+    2001  0.2  0.2 2017712 21264 pts/1   Sl+  16:35   0:00 ./iotmi_device_agent
Process 'iotmi_device_agent' is running.
hub1234+    2045  0.0  0.1 1457824 12624 pts/1   Sl+  16:35   0:00 ./
iotmi_lpw_provisioner
Process 'iotmi_lpw_provisioner' is running.
hub1234+    1813  0.0  0.0 875152  6848 pts/1    Sl+  16:34   0:00 ./iotmi_log_daemon
Process 'iotmi_log_daemon' is running.
------Successfully Started Hub SDK----
```

次のステップ

Hub SDK を正常に起動したら、 でデバイスのオンボーディングと管理に進みますデバイスをオンボードして運用するためのユーザーガイドのセットアップ。

Raspberry Pi のマネージドインテグレーション Hub SDK Docker コンテナ

> **Note**
>
> Raspberry Pi AWS IoT での Hub SDK のこの実装は、学習とテストのみを目的としており、本番環境での使用を意図していません。このデモでは、開発を容易にするために以下の設定を行います。

AWS 認証情報ストレージ: デモのみを目的として、認証情報と証明書は、テストと開発を容易にするために、アクセス可能な場所に保存されます。本番環境では AWS Secrets Manager、 や Systems Manager Parameter Store などの安全なストレージソリューションを使用する必要があります。保管時の暗号化を実装し、 AWS IoT セキュリティガイドラインに従う必要があります。
コンテナ権限: デモは昇格された権限で実行され、ホストリソースへの無制限アクセスを許可し、開発ワークフローを簡素化します。本番環境では、コンテナは最小限必要な権限で動作する必要があります。
ネットワークブリッジ設定: デモでは、デバッグとモニタリングを容易にするために内部ネットワークトラフィックを公開するネットワークブリッジ設定を使用します。本番環境では、内部ネットワークトラフィックへの不正アクセスを防ぐために、適切なネットワーク分離とセグメンテーションを実装します。
USB デバイスのアクセス許可: 開発周辺機器とテストデバイスの接続を容易にするために、無制限の USB デバイスアクセスが有効になっています。本番環境では、厳格な USB デバイスコントロールと検証を実装して、デバイスのなりすまし攻撃を防止します。
これらの設定により、簡単なテストが可能になり、本番環境で notcbe を使用する必要があります。本番環境にデプロイする場合は、セキュリティのベストプラクティスに従って、ホストシステムの侵害や認証情報への不正アクセスを防止してください。

前提条件

Docker コンテナの には、次の前提条件が必要です。

- Raspberry Pi イメージャをダウンロードしてインストールします。
- SD カードを取得します。
- 2.4Ghz 64 ビットクアッドコア CPU (8GB RAM) を搭載した Raspberry Pi 5 をセットアップします。
- Sonoff Zigbee USB ドングルを接続します。
- ファームウェアを Sonoff Zigbee USB ドングルにフラッシュする.
- Silicon Labs SLUSB001A ドングルを接続します。
- AWS アカウントにサインアップします。
- AWS CLI マネージド統合コマンドリファレンスから最新バージョンの AWS CLI をインストールします。
- IP アドレスまたはホスト名を使用した Raspberry Pi への SSH アクセス。

Raspberry Pi で Managed Integrations Hub SDK Docker コンテナを使用する

1. マネージド統合 Raspberry Pi Hub SDK Docker をダウンロードします。
2. SCP を使用して Raspberry Pi にファイルをコピーします。

```
scp ~/path/to/IotMI-HubSDK-Docker-v1.0.0.tar.gz [username]@raspberrypi.local:~
```

3. SSH 経由で Raspberry Pi に接続します。

```
ssh hub123456@raspberrypi.local
```

4. 存在しない場合は Docker をインストールします。

```
# Install Docker
cd
curl -fsSL https://get.docker.com | sudo sh

# Add your user to docker group
sudo usermod -aG docker $USER
exit # exit ssh

# Log in again
```

5. 存在しない場合は Docker Compose をインストールします。

```
# Install Docker Compose
sudo apt-get update
sudo apt-get install -y docker-compose-plugin
```

6. Hub SDK ファイルを抽出します。

```
# Navigate to the home directory
cd

# Extract the hub-docker.tar.gz file
tar -xzf IotMI-HubSDK-Docker-v1.0.0.tar.gz
```

7. hub-docker ディレクトリに移動します。

```
cd IotMI-HubSDK-Docker
```

8. Hub のオンボーディング設定を完了して、認証と設定を設定します。

> **Note**
>
> このステップを実行するには、 YULまたは DUB リージョンにいる必要があります。

9. Docker コンテナを起動します。

```
# The first time it's called, it will build the container
docker compose up -d
docker compose logs -f
```

予想される出力:

```
[+] Running 1/1
 # Container iotmi-hubsdk-docker-hubsdk-1  Started
hubsdk-1  | -\-\-\-\-\-\-Checking USB dongles-\-\-\-\-\-\-\-\-\-\-\-\-\-
hubsdk-1  | -\-\-\-\-\-\-Stopping SDK running processes-\-\-
hubsdk-1  | iotmi_mqtt_proxy: no process found
hubsdk-1  | ace_eventmgr: no process found
hubsdk-1  | ace_zigbee_service: no process found
hubsdk-1  | zwave_svc: no process found
hubsdk-1  | iotmi_cdmb: no process found
hubsdk-1  | iotmi_device_agent: no process found
hubsdk-1  | iotmi_lpw_provisioner: no process found
hubsdk-1  | iotmi_log_daemon: no process found
hubsdk-1  | -\-\-\-\-\-\-Starting Hub SDK-\-\-\-\-\-\-\-\-\-\-\-\-\-
hubsdk-1  | -\-\-\-\-\-\-Creating logs directory-\-\-\-\-\-\-\-\-\-\-
hubsdk-1  | Logs directory created.
hubsdk-1  | -\-\-\-\-\-\-Verifying Middleware paths-\-\-\-\-\-\-
hubsdk-1  | All middleware libraries exist
hubsdk-1  | -\-\-\-\-\-\-Verifying Middleware pre reqs-\-\-
hubsdk-1  | AIPC and KVstroage directories exist
hubsdk-1  | -\-\-\-\-\-\-Starting HubOnboarding-\-\-\-\-\-\-\-\-\-\-\-
hubsdk-1  | -\-\-\-\-\-\-Starting MQTT Proxy-\-\-\-\-\-\-\-\-\-\-\-\-
hubsdk-1  | -\-\-\-\-\-\-Staring Log Daemon-\-\-
hubsdk-1  | -\-\-\-\-\-\-Starting Event Manager-\-\-\-\-\-\-\-\-\-\-\-
hubsdk-1  | -\-\-\-\-\-\-Starting Zigbee Service-\-\-\-\-\-\-\-\-\-\-\-
hubsdk-1  | -\-\-\-\-\-\-Starting Zwave Service-\-\-\-\-\-\-\-\-\-\-\-
hubsdk-1  | /data/aws/iotmi/middleware/AceZwave/bin /data/aws/iotmi
hubsdk-1  | /data/aws/iotmi
hubsdk-1  | -\-\-\-\-\-\-Starting CDMB-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-
hubsdk-1  | -\-\-\-\-\-\-Starting Agent-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-
```

```
hubsdk-1  | -\-\-\-\-\-\-Starting Provisioner-\-\-\-\-\-\-\-\-\-\-\-\-\-\-
hubsdk-1  | -\-\-\-\-\-\-Checking SDK status-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-
hubsdk-1  | root         105  0.2  0.2 1093488 16608 ?       Sl   20:51   0:00 ./
iotmi_mqtt_proxy -C /data/aws/iotmi/config/iotmi_config.json
hubsdk-1  | Process 'iotmi_mqtt_proxy' is running.
hubsdk-1  | root         183  0.0  0.0 236272  3152 ?        Sl   20:51   0:00 ./
middleware/AceCommon/bin/ace_eventmgr
hubsdk-1  | Process 'ace_eventmgr' is running.
hubsdk-1  | root         190 12.0  0.1 319264  8352 ?        Sl   20:51   0:04 ./
middleware/AceZigbee/bin/ace_zigbee_service
hubsdk-1  | Process 'ace_zigbee_service' is running.
hubsdk-1  | root         200  0.0  0.1 1365792 12480 ?       Sl   20:51   0:00 ./
zwave_svc
hubsdk-1  | Process 'zwave_svc' is running.
hubsdk-1  | root         233  0.0  0.0 1198704 5760 ?        Sl   20:51   0:00 ./
iotmi_cdmb
hubsdk-1  | Process 'iotmi_cdmb' is running.
hubsdk-1  | root         268  0.2  0.2 2017424 21968 ?       Sl   20:51   0:00 ./
iotmi_device_agent
hubsdk-1  | Process 'iotmi_device_agent' is running.
hubsdk-1  | root         311  0.1  0.1 1523072 13008 ?       Sl   20:51   0:00 ./
iotmi_lpw_provisioner
hubsdk-1  | Process 'iotmi_lpw_provisioner' is running.
hubsdk-1  | root         132  0.0  0.0 875024  7232 ?        Sl   20:51   0:00 ./
iotmi_log_daemon
hubsdk-1  | Process 'iotmi_log_daemon' is running.
hubsdk-1  | -\-\-\-\-\-\-Successfully Started Hub SDK-\-\-\-
```

Hub SDK を正常に起動したら、 でデバイスのオンボーディングと管理に進みますデバイスをオンボードして運用するためのユーザーガイドのセットアップ。

Note

- Docker コンテナ bash シェルにアクセスするには、次のコマンドを実行します。

  ```
  docker compose exec hubsdk bash
  ```

- 再起動後にコンテナを再起動するには、次のコマンドを実行します。

  ```
  docker compose up -d
  ```

- Hub SDK を更新するには、次のフォルダのバイナリを置き換えます。

```
hub-docker/iotmi
```

- データの保存中にコンテナを安全に再起動するには、次の手順を実行します。

```
docker compose down
docker compose up -d
docker compose logs -f
```

マネージド統合デモアプリケーション

> **Note**
>
> Raspberry Pi AWS IoT での Hub SDK のこの実装は、学習とテストのみを目的としており、本番環境での使用を意図していません。このデモでは、開発を容易にするために以下の設定を行います。
> AWS 認証情報ストレージ: デモのみを目的として、認証情報と証明書は、テストと開発を容易にするために、アクセス可能な場所に保存されます。本番環境では AWS Secrets Manager、 や Systems Manager Parameter Store などの安全なストレージソリューションを使用する必要があります。保管時の暗号化を実装し、 AWS IoT セキュリティガイドラインに従う必要があります。
> コンテナ権限: デモは昇格された権限で実行され、ホストリソースへの無制限アクセスを許可し、開発ワークフローを簡素化します。本番環境では、コンテナは最小限必要な権限で動作する必要があります。
> ネットワークブリッジ設定: デモでは、デバッグとモニタリングを容易にするために内部ネットワークトラフィックを公開するネットワークブリッジ設定を使用します。本番環境では、内部ネットワークトラフィックへの不正アクセスを防ぐために、適切なネットワーク分離とセグメンテーションを実装します。
> USB デバイスのアクセス許可: 開発周辺機器とテストデバイスの接続を容易にするために、無制限の USB デバイスアクセスが有効になっています。本番環境では、厳格な USB デバイスコントロールと検証を実装して、デバイスのなりすまし攻撃を防止します。
> これらの設定により、簡単なテストが可能になり、本番環境で notcbe を使用する必要があります。本番環境にデプロイする場合は、セキュリティのベストプラクティスに従って、ホストシステムの侵害や認証情報への不正アクセスを防止してください。

デモアプリケーションは、スマートホームデバイス管理用のマネージド統合機能を示す React ベースのデモアプリケーションです。このアプリケーションは、最新のウェブインターフェイスを通じて Z-Wave および Zigbee デバイスのデバイスのオンボーディング、制御、モニタリングを示します。

前提条件

- AWS アカウントにサインアップします。
- 認証情報ロッカーを作成し、認証情報ロッカーをハブ に追加します。
- Hub のオンボーディング設定を完了します。
- Node.js 18+ および npm。
- AWS CLI マネージド統合コマンドリファレンスから最新バージョンの AWS CLI をインストールします。
- 最新のウェブブラウザ (Chrome、Firefox、Safari、Edge)

アプリケーションのインストールと設定

1. マネージド統合デモアプリケーションをダウンロードします。
2. パッケージを抽出します。

```
cd ~/Downloads
tar -xzf IotMI-HubSDK-DemoApp-v1.0.0.tar.gz
cd IotManagedIntegrations-DemoApp
```

3. 依存関係をインストールします。

```
npm install
```

4. ルートディレクトリに .env ファイルを作成します。

```
# AWS Configuration
REACT_APP_AWS_REGION=your_region
REACT_APP_AWS_ACCESS_KEY_ID=your_access_key
REACT_APP_AWS_SECRET_ACCESS_KEY=your_secret_key
REACT_APP_AWS_SESSION_TOKEN=your_session_token

# IoT Managed Integrations Endpoint
REACT_APP_IOT_ENDPOINT=https://your-iot-endpoint.amazonaws.com
```

```
# Hub Configuration
REACT_APP_HUB_MANAGED_THING_ID=your_hub_id
REACT_APP_CREDENTIAL_LOCKER_ID=your_credential_locker_id
```

5. アプリケーションを構築して起動します。

```
npm start
```

6. アプリケーションへのアクセス先:

```
http://localhost:3000
```

料金情報については、AWS IoT 「デバイス管理の料金」ページの「マネージド統合」セクションを参照してください。

オフボードマネージド統合ハブ

Hub SDK オフボードプロセスの概要

ハブオフボーディングプロセスは、 AWS クラウド 管理システムからハブを削除します。クラウドが DeleteManagedThing リクエストを送信すると、プロセスは 2 つの主要な目的を達成します。

デバイス側のアクション：

- ハブの内部状態をリセットする
- ローカルに保存されたすべてのデータを削除する
- 今後の再オンボーディングに備えてデバイスを準備する

クラウド側のアクション：

- ハブに関連付けられているすべてのクラウドリソースを削除する
- 前のアカウントからの完全な切断

お客様は通常、次の場合にハブオフボーディングを開始します。

- ハブの関連アカウントの変更
- 既存のハブを新しいデバイスに置き換える

このプロセスにより、ハブ設定間のクリーンで安全な移行が保証され、シームレスなデバイス管理とアカウントの柔軟性が可能になります。

前提条件

- オンボーディングされているハブが必要です。手順については、「ハブのオンボーディング設定」を参照してください。
- /data/aws/iotmi/config/ にある `iotmi_config.json` ファイルで、 に `iot_provisioning_state`が表示されていることを確認しますPROVISIONED。
- で参照される永続的な証明書とキーが、指定されたパス`iotmi_config.json`に存在することを確認します。
- HubOnboarding、Agent、Provisioner、および MQTT プロキシが正しく設定され、実行されていることを確認します。
- ハブに子デバイスがないことを確認します。 DeleteManagedThing API を使用して、続行する前にすべての子デバイスを削除します。

Hub SDK オフボードプロセス

ハブをオフボードするには、次の手順に従います。

hub_managed_thing ID を取得する

`iotmi_config.json` ファイルは、マネージド統合ハブのマネージドモノ ID を保存するために使用されます。この識別子は、ハブが AWS IoT Managed Integrations サービスと通信できるようにする重要な情報です。マネージド型モノ ID は、JSON ファイルの rw (読み取り/書き込み) セクションの `managed_thing_id`フィールドの下に保存されます。これは、次のサンプル設定で確認できます。

```
{
      "ro": {
          "iot_provisioning_method": "FLEET_PROVISIONING",
          "iot_claim_cert_path": "PATH",
          "iot_claim_pk_path": "PATH",
          "UPC": "UPC",
          "sh_endpoint_url": "ENDPOINT_URL",
          "SN": "SN",
          "fp_template_name": "TEMPLATENAME"
```

```
        },
        "rw": {
            "iot_provisioning_state": "PROVISIONED",
            "client_id": "ID",
            "managed_thing_id": "ID",
            "iot_permanent_cert_path": "CERT_PATH",
            "iot_permanent_pk_path": "KEY",
            "metadata": {
                "last_updated_epoch_time": 1747766125
            }
        }
}
```

オフボードハブにコマンドを送信する

アカウントの認証情報を使用して、前のセクションでmanaged_thing_id取得した で コマンドを実行します。

```
aws iot-managed-integrations delete-managed-thing \
    --identifier HUB_MANAGED_THING_ID
```

ハブがオフボードされたことを確認する

アカウントの認証情報を使用して、前のセクションでmanaged_thing_id取得した で コマンドを実行します。

```
aws iot-managed-integrations get-managed-thing \
    --identifier HUB_MANAGED_THING_ID
```

成功と失敗のシナリオ

成功シナリオ

ハブをオフボードするコマンドが成功した場合、次のサンプルレスポンスが期待されます。

```
{
      "Message" : "Managed Thing resource not found."
}
```

さらに、ハブオフボーディングコマンドが成功した場合、次のサンプルが観察iotmi_config.jsonされます。rw セクションに iot_provisioning_stateとオプション

でメタデータのみが含まれていることを確認します。メタデータがないことは許容されます。 は NOT_PROVISIONED iot_provisioning_stateである必要があります。

```
{
      "ro": {
          "iot_provisioning_method": "FLEET_PROVISIONING",
          "iot_claim_cert_path": "PATH",
          "iot_claim_pk_path": "PATH",
          "UPC": "1234567890101",
          "sh_endpoint_url": "ENDPOINT_URL",
          "SN": "1234567890101",
          "fp_template_name": "test-template"
      },
      "rw": {
          "iot_provisioning_state": "NOT_PROVISIONED",
          "metadata": {
              "last_updated_epoch_time": 1747766125
          }
      }
}
```

失敗シナリオ

ハブをオフボードするコマンドが失敗した場合、次のサンプルレスポンスが期待されます。

```
{
      "Arn" : "ARN",
      "CreatedAt" : 1.748968266655E9,
      "Id" : "ID",
      "ProvisioningStatus" : "DELETE_IN_PROGRESS",
      "Role" : "CONTROLLER",
      "SerialNumber" : "SERIAL_NO",
      "Tags" : { },
      "UniversalProductCode" : "UPC",
      "UpdatedAt" : 1.748968272107E9
}
```

- ProvisioningStatus が の場合はDELETE_IN_PROGRESS、「Hub recovery」の手順に従います。
- ProvisioningStatus が でない場合DELETE_IN_PROGRESS、ハブをオフボードするコマンドが Managed Integrations クラウドで失敗したか、 Managed Integrations クラウドで受信されませんでした。「Hub recovery」の手順に従ってください。

- オフボードが失敗した場合、iotmi_config.jsonファイルは以下のサンプルファイルのようになります。

```
{
      "ro": {
          "iot_provisioning_method": "FLEET_PROVISIONING",
          "iot_claim_cert_path": "PATH",
          "iot_claim_pk_path": "PATH",
          "UPC": "123456789101",
          "sh_endpoint_url": "ENDPOINT_URL",
          "SN": "123456789101",
          "fp_template_name": "test-template"
      },
      "rw": {
          "iot_provisioning_state": "PROVISIONED",
          "client_id": "ID",
          "managed_thing_id": "ID",
          "iot_permanent_cert_path": "PATH",
          "iot_permanent_pk_path": "PATH",
          "metadata": {
              "last_updated_epoch_time": 1747766125
          }
      }
}
```

(オプション) Hub SDK をオフボーディングした後

> **Important**
>
> 以下のシナリオでは、Hub SDK のオフボーディング後に実行するオプションのアクションが失敗した場合、またはオフボーディング後にハブを再オンボーディングする場合に実行するアクションを一覧表示します。

再オンボード

オフボーディングが成功した場合は、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) と、残りのオンボードプロセスに従って Hub SDK をオンボードします。

ハブリカバリ

デバイスハブのオフボーディングが成功し、クラウドオフボーディングが失敗する

GetManagedThing API コールがManaged Thing resource not foundメッセージを返さないが、ファイルがiotmi_config.jsonオフボードされている場合。サンプル JSON ファイルの「成功シナリオ」を参照してください。

このシナリオから復旧するには、「強制削除」を参照してください。

デバイスハブのオフボーディングが失敗する

このシナリオでは、ファイルが正しくオフボードiotmi_config.jsonされません。サンプル JSON ファイルの「失敗シナリオ」を参照してください。

このシナリオから復旧するには、「強制削除」を参照してください。がまだオフボードされていない場合iotmi_config.jsonは、ハブを工場出荷時リセットする必要があります。

デバイスハブのオフボーディングとクラウドオフボーディングが失敗する

このシナリオでは、 iotmi_config.jsonはオフボードされず、ハブステータスは ACTIVATED、、または のいずれかですDISCOVERED。

このシナリオから復旧するには、「強制削除」を参照してください。強制削除iotmi_config.jsonが失敗した場合、またはオフボードされていない場合は、ハブを工場出荷時リセットする必要があります。

ハブがオフラインで、ハブのステータスが DELETE_IN_PROGRESS である

このシナリオでは、ハブはオフラインで、クラウドはオフボーディングコマンドを受け取ります。

このシナリオから復旧するには、「強制削除」を参照してください。

強制削除

デバイスハブのオフボーディングが成功せずにクラウドリソースを削除するには、次の手順に従います。このオペレーションでは、クラウドとデバイスの状態の間に不整合が発生し、将来のオペレーションで問題が発生する可能性があります。

ハブの managed_thing_idおよび force パラメータを使用して DeleteManagedThing API を呼び出します。

```
aws iot-managed-integrations delete-managed-thing \
```

```
  --identifier HUB_MANAGED_THING_ID \
  --force
```

次に、GetManagedThing API を呼び出し、 が返されることを確認しますManaged Thing resource not found。これにより、クラウドリソースが削除されたことを確認します。

> **Note**
>
> クラウドとデバイスの状態の間に不整合が生じる可能性があるため、このアプローチはお勧めしません。通常、クラウドリソースを削除する前に、デバイスハブのオフボーディングを成功させることをお勧めします。

プロトコル固有のミドルウェア

> **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。マネージド統合 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。

- 共通の 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。

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、次の例に示すディレクトリ内にあります。

```
./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

シリコンラボ 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、 フォルダにリストされている IoTmanagedintegrationsMiddlewares/*example*iot-ace-zwave-mw/フォルダ内にあります。この場所の 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
```

ミドルウェアを SDK と統合する

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

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開始点として、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。

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。から End Device SDK の最新バージョンをダウンロードする AWS マネジメントコンソール

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、統合手順を確認して、実装の構築を開始します。

トピック

End Device SDK とは

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備えたデバイスをサポートします。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、マネージド統合用のデータモデルハンドラーを組み合わせます。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テクニカルリファレンス。

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の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、ファイル配信のover-the-air経由 (OTA) 更新を処理します。この組み込みサービスは、登録されたすべての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 関数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

- プロビジョニングワークフロー
- 環境変数を設定する
- カスタムエンドポイントを登録する
- 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
- マネージドモノの機能

プロビジョニングワークフロー

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登録する

クラウド環境で RegisterCustomEndpoint 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 iot-managed-integrations register-custom-endpoint
```

レスポンスの例

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

> **Note**
>
> プロビジョニングパラメータを設定するためのエンドポイントアドレスを保存します。GetCustomEndpoint API を使用してエンドポイント情報を返します。詳細については、「 マネージド統合 API リファレンスガイド」のGetCustomEndpoint API」およびRegisterCustomEndpoint API」を参照してください。

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 iot-managed-integrations create-provisioning-profile \
--provisioning-type "FLEET_PROVISIONING" \
--name "PROVISIONING-PROFILE-NAME"
```

レスポンスの例

```
{ "Arn":"arn:aws:iot-managed-integrations: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 iot-managed-integrations 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:iot-managed-integrations: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ソリューションプロバイダーには、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の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>

マネージドモノの機能

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。

Over-the-Air更新

OTA アーキテクチャの概要

Over-the-Air (OTA) 更新プロセスでは、複数のコンポーネントが連携してファームウェア更新をデバイスに配信します。次の図は、エンドデバイス SDK、Hub SDK、 機能間のやり取りを通じて OTA 更新リクエストを処理する方法を示しています。

OTA 更新アーキテクチャは、次のコンポーネントで構成されます。

- 顧客: ジョブドキュメントを S3 バケットにアップロードし、API 経由で更新を開始する
- OTA サービス: ジョブの作成、検証、管理を処理します
- AWS IoT ジョブ: ジョブの実行とデバイスへの配信を管理します
- デバイス: Harmony SDK を使用して更新を受信して適用する

前提条件

OTA タスクを作成する前に、次の前提条件を設定する必要があります。

Amazon S3 アクセスを設定する

OTA 更新を有効にするには、ジョブドキュメントを Amazon S3 バケットにアップロードし、適切なアクセス許可を設定する必要があります。

1. OTA ジョブドキュメントを S3 バケットにアップロードする
2. 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。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "PolicyForS3JobDocument",
      "Effect": "Allow",
      "Principal": {
        "Service": "iotmanagedintegrations.amazonaws.com"
      },
      "Action": "s3:GetObject",
      "Resource": [
        "arn:aws:s3:::YOUR_BUCKET/*",
        "arn:aws:s3:::YOUR_BUCKET/ota_job_document.json",
        "arn:aws:s3:::YOUR_BUCKET"
```

```
      ]
    }
  ]
}
```

Over-the-Air (OTA) タスクの実装

更新要件とデバイスターゲティング戦略に応じて、2 つの方法で OTA タスクを作成できます。

OTA タスクの 1 回限りの更新

ワンタイム OTA タスクには、OTA 更新を実行するターゲット (ManagedThings) の静的リストが含まれています。一度に最大 100 個のターゲットを追加できます。ワークフローは、マネージド統合抽象化レイヤーを維持しながら、フリートインデックス作成で AWS IoT ジョブを使用します。

次の例を使用して、ワンタイム OTA タスクを作成します。

```
aws iotmanagedintegrations create-ota-task \
  --description "One-time OTA update" \
  --s3-url "s3://test-job-document-bucket/ota-job-document.json" \
  --protocol HTTP \
  --target ["arn:aws:iotmanagedintegrations:region:account id:managed-thing/managed
 thing id"] \
  --ota-mechanism PUSH \
  --ota-type ONE_TIME \
  --client-token "foo" \
  --tags '{"key1":"foo","key2":"foo"}'
```

継続的な OTA タスクの更新

OTA (Over-the-Air) グループ化ワークフローを使用すると、マネージド統合抽象化レイヤーを維持しながら、フリートインデックス作成で AWS IoT ジョブを使用して、特定の属性に基づいてデバイスのグループにファームウェア更新をデプロイできます。継続的な OTA タスクは、特定のターゲットではなくクエリ文字列を使用します。クエリ条件に一致するすべてのデバイスは OTA 更新を受け、クエリ条件は継続的に再評価されます。一致するターゲットにはジョブのデプロイが含まれます。

前提条件を設定する

継続的な OTA タスクを作成する前に、以下の前提条件を完了してください。

1. CreateManagedThing API を呼び出してマネージド型モノを作成し、フリートプロビジョニングを実行します。
2. クエリターゲティングのためにマネージドモノにメタデータ属性を追加します。

UpdateManagedThing API ManagedThingを使用して属性とメタデータを に追加します。

```
aws iotmanagedintegrations update-managed-thing \
  --managed-thing-id "YOUR_MANAGED_THING_ID" \
  --meta-data '{"owner":"managedintegrations","version":"1.0"}'
```

連続 OTA タスクを作成するには、次の例を使用します。

```
aws iotmanagedintegrations create-ota-task \
  --description "Continuous OTA update" \
  --s3-url "s3://test-job-document-bucket/ota-job-document.json" \
  --protocol HTTP \
  --ota-mechanism PUSH \
  --ota-type CONTINUOUS \
  --client-token "foo" \
  --ota-target-query-string "attributes.owner=managedintegrations" \
  --tags '{"key1":"foo","key2":"foo"}'
```

継続的な OTA ワークフローを理解する

継続的な OTA 更新ワークフローは、以下のステップに従います。

1. UpdateManagedThing API を使用して、マネージド型モノを属性で更新します。
2. 特定のデバイス属性をターゲットとするクエリ文字列を使用して OTA ジョブを作成します。
3. OTA サービスは、クエリ属性 AWS IoT Core に基づいて に動的なモノのグループを作成します。
4. IoT ジョブは、一致するデバイスで更新を実行します
5. ListOtaTaskExecutions API または Kinesis ストリーム (有効になっている場合) による OTA 通知を使用して進行状況をモニタリングします。

マネージド統合 OTA ジョブと IoT ジョブの違い

マネージド統合 OTA と IoT ジョブの根本的な違いは、サービスオーケストレーションと自動化にあります。マネージド統合 OTA は、マルチサービスの調整の複雑さを抽象化する単一サービスソリューションを提供します。

マネージド統合 OTA が自動で実行するもの:

- モノの動的グループの作成: クエリ条件に基づいてモノのグループを自動的に生成 AWS IoT Core します。
- ターゲット解決: クエリ文字列 (例: `attributes.owner=managedintegrations`) を実際のデバイスターゲットに変換します。
- サービス統合: AWS IoT Core、IoT ジョブ、フリートインデックス作成サービス間でシームレスに調整します。
- ライフサイクル管理: 作成から実行モニタリングまでの OTA ワークフロー全体を処理します。

MI OTA が排除するもの:

- でモノのグループを作成します AWS IoT Core。
- グループへのモノの追加。
- IoT ジョブの作成。

マネージド統合 OTA は、クエリ文字列に基づいて 3 つのオペレーションをすべて内部で処理し、条件に一致するデバイスを自動的に検出し、内部で IoT ジョブを作成し、複数の AWS サービスを直接操作することなく完全な OTA ワークフローをオーケストレーションします。

OTA タスク設定のセットアップ

OTA 更新の設定を作成して、更新をデバイスにロールアウトする方法を制御したり、中止条件を設定したり、タイムアウトを設定したりできます。

例: CreateOtaTaskConfiguration

OTA タスク設定を作成するには、次の例を使用します。

```
aws iotmanagedintegrations create-ota-task-configuration \
  --description "OTA configuration" \
  --name "MyOtaConfig" \
  --push-config '{
    "AbortConfig": {
      "AbortConfigCriteriaList": [
        {
          "Action": "CANCEL",
          "FailureType": "FAILED",
```

```
          "MinNumberOfExecutedThings": 1,
          "ThresholdPercentage": 90.0
        }
      ]
    },
    "RolloutConfig": {
      "ExponentialRolloutRate": {
        "BaseRatePerMinute": 1,
        "IncrementFactor": 3.0,
        "RateIncreaseCriteria": {
          "numberOfNotifiedThings": 1
        }
      },
      "MaximumPerMinute": 1
    },
    "TimeoutConfig": {
      "InProgressTimeoutInMinutes": 100
    }
  }' \
  --client-token "foo"
```

OTA タスクに設定を適用する

設定が作成されると、CreateOtaTaskリクエストに追加された taskConfigurationIdと追加の設定が届きます。

```
aws iotmanagedintegrations create-ota-task \
  --description "OTA with configuration" \
  --s3-url "s3://test-job-document-bucket/ota-job-document.json" \
  --protocol HTTP \
  --target ["arn:aws:iotmanagedintegrations:region:account id:managed-thing/managed
 thing id"] \
  --ota-mechanism PUSH \
  --ota-type ONE_TIME \
  --client-token "foo" \
  --task-configuration-id "ae4f49352c5443369f43ad6c3a7f1580" \
  --ota-scheduling-config '{
    "EndBehavior": "STOP_ROLLOUT",
    "EndTime": "2024-10-23T17:00",
    "StartTime": "2024-10-20T17:00"
  }' \
  --ota-task-execution-retry-config '{
    "RetryConfigCriteria": [
```

```
      {
        "FailureType": "FAILED",
        "MinNumberOfRetries": 1
      }
    ]
  }' \
  --tags '{"key1":"foo","key2":"foo"}'
```

OTA 通知のモニタリング

OTA 更新は、次の 2 つの異なる方法でモニタリングできます。

Kinesis Data Streams を介したプッシュ通知

OTA 通知が有効になっている場合、更新ステータスイベントは自動的に Kinesis ストリームにプッシュされます。これにより、デバイス間のファームウェア更新の進行状況をリアルタイムで可視化できます。

ListOtaTaskExecutions API によるモニタリング

ListOtaTaskExecutions API を使用して、マネージドモノの OTA 更新のステータスを手動で確認できます。

```
aws iotmanagedintegrations list-ota-task-executions \
  --task-id "task-123456789" \
  --max-results 25
```

レスポンスは、各マネージドモノの詳細な実行ステータスを提供します。

```
{
  "taskExecutionSummaries": [
    {
      "taskExecutionSummary": {
        "executionNumber": 1,
        "lastUpdatedAt": 1634567890,
        "queuedAt": 1634567800,
        "startedAt": 1634567830,
        "status": "SUCCEEDED",
        "retryAttempt": 0
      },
      "managedThingId": "device-001"
```

```
    },
    {
      "taskExecutionSummary": {
        "executionNumber": 1,
        "lastUpdatedAt": 1634567920,
        "queuedAt": 1634567800,
        "startedAt": 1634567840,
        "status": "IN_PROGRESS",
        "retryAttempt": 0
      },
      "managedThingId": "device-002"
    }
  ],
  "nextToken": "NEXT_TOKEN"
}
```

この API を使用すると、タイムスタンプや現在のステータスなど、特定の OTA タスクの対象となる各マネージドモノの詳細な実行ステータスを取得できます。

ジョブドキュメントの処理

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. 使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場合、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する必要があります。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します。
3. デジタル署名を検証します。
4. 検証済みの更新をインストールします。
5. SUCCESS または FAILEDステータスiotmi_JobsHandler_updateJobStatusで を呼び出します。

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、ジョブの成功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
- 環境設定
- 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 データモデル (.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、C ソースファイルを作成するコードジェネレーターを示しています。

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

- (オプション) カスタムスキーマ

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

テンプレート	生成されたソース	解説
		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。

(オプション) カスタムスキーマ

エンドデバイス SDK は、標準化されたコード生成プロセスをカスタムスキーマと組み合わせます。これにより、デバイスとデバイスソフトウェアの Matter Data Model を拡張できます。カスタムスキーマは、デバイスdevice-to-cloud通信のデバイス機能を記述するのに役立ちます。

形式、構造、要件など、マネージド統合データモデルの詳細については、「」を参照してくださいマネージド統合データモデル。

codegen.py ツールを使用して、次のようにカスタムスキーマの C ソースファイルを生成します。

> **Note**
>
> 各カスタムクラスターには、次の 3 つのファイルに対して同じクラスター ID が必要です。

- クラウドに新しいカスタムクラスターを作成する機能レポート用のクラスターの表現を提供する JSON形式のカスタムスキーマを作成します。サンプルファイルは にありますcodegen/custom_schemas/custom.SimpleLighting@1.0。
- カスタムスキーマと同じ情報を含むXML形式で ZCL (Zigbee クラスターライブラリ) 定義ファイルを作成します。ZAP ツールを使用して、ZCL XML から Matter IDL ファイルを生成します。サンプルファイルは にありますcodegen/zcl/custom.SimpleLighting.xml。

- ZAP ツールからの出力は Matter IDL File (.matter)であり、カスタムスキーマに対応する Matter クラスターを定義します。これは、エンドデバイス SDK の C ソースファイルを生成するためのcodegen.pyツールの入力です。サンプルファイルは にありますcodegen/matter_files/custom-light.matter。

カスタムマネージド統合データモデルをコード生成ワークフローに統合する方法の詳細については、「」を参照してくださいデバイスのコードを生成する。

環境設定

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. から End Device SDK の最新バージョンをダウンロードします AWS マネジメントコンソール。
2. 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
    ```

3. 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 -b v1.4.0.0 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
```

(オプション) カスタムスキーマでコードを生成するには

1. カスタムスキーマを JSON形式で準備する
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
 --custom-schemas-dir path-to-custom-schema-directory
```

上記のコマンドでは、複数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。

低レベルの C 機能 APIs

が提供する低レベルの C 機能 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。以下のアクションでは、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。たとえば、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、プロパティの変更やイベントが発生すると、C-Function 実装は未承諾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変換します

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。

End Device SDK の使用を開始する

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、マネージド型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. Amazon EC2 インスタンスのセットアップ

   にサインイン AWS マネジメントコンソール し、Amazon Linux AMI を使用して Amazon EC2 インスタンスを起動します。Amazon Elastic Container Registry ユーザーガイドのAmazon EC2 の開始方法」を参照してください。
2. 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
   ```
3. (オプション) ネットワークをセットアップする

   エンドデバイス SDK は、物理ハードウェアに最適です。Amazon EC2 を使用している場合は、このステップを実行しないでください。

   サンプルアプリケーションを使用する前に Amazon EC2 を使用していない場合は、ネットワークを初期化し、デバイスを利用可能な 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();
   ```

4. プロビジョニングパラメータを設定する

> **Note**
>
> 先に進む前に、プロビジョンドに従ってクレーム証明書とプライベートキーを取得します。

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。

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。手順については、「カスタムエンドポイントを登録する」を参照してください。

マクロパラメータ	説明	この情報を取得する方法
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。

5. 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 C 関数を実装します。実行プロビジョニング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: XXXXXXXXXXX
```

```
[2406832728][MAIN][INFO] rootCertificatePath: XXXXXXXXX
[2406832728][MAIN][INFO] pClaimCertificatePath: XXXXXXXX
[2406832728][MAIN][INFO] pClaimKeyPath: XXXXXXXXXXXXXXXX
[2406832728][MAIN][INFO] deviceInfo.serialNumber XXXXXXXXXXX
[2406832728][MAIN][INFO] deviceInfo.universalProductCode XXXXXXXXXXXXXX
[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: XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
[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 :
 XXXXXXXXXXXXXXXXXXX_XXXXXXXXXXXXXXXXXXXXXXX
[2406840540][MAIN][INFO] Managed thing ID : XXXXXXXXXXXXXXXXXXXXXX
[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

Endpoint	クラスター
	ファンコントロール
	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
	フォームの勾配の測定
	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。

6. 次のステップ:

マネージド統合 エンドデバイス SDK とデモアプリケーションが Amazon EC2 インスタンスで実行されました。これにより、独自の物理ハードウェアでアプリケーションを開発およびテストできます。この設定では、 マネージド統合サービスを活用して AWS IoT デバイスを制御できます。

a. 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の C 機能 APIs。

```
struct DeviceState
{
  struct iotmiDev_Agent *agent;
  struct iotmiDev_Endpoint *endpointLight;
```

```
  /* This simulates the HW state of OnOff */
  bool hwState;
};

/* This implementation for OnOff getter just reads
   the state from the DeviceState */
iotmiDev_DMStatus exampleGetOnOff(bool *value, void *user)
{
   struct DeviceState *state = (struct DeviceState *)(user);
  *value = state->hwState;
  return iotmiDev_DMStatusOk;
}
```

b. 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。

i. デバイスエージェントを作成します。

A. 他の SDK 関数を呼び出すiotmiDev_Agent_new()前に、 を使用してデバイスエージェントを作成します。

B. 少なくとも、設定には thingId パラメータと clientId パラメータが含まれている必要があります。

C. iotmiDev_Agent_initDefaultConfig() 関数を使用して、キューサイズや最大エンドポイントなどのパラメータに妥当なデフォルト値を設定します。

D. リソースの使用が終了したら、 iotmiDev_Agent_free()関数でリソースを解放します。これにより、メモリリークが防止され、アプリケーションで適切なリソース管理が保証されます。

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タを入力します。

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します。

を使用してエンドポイントを作成します。これにはiotmiDev_Agent_addEndpoint()以下が必要です。

A. 一意のエンドポイント ID。

B. わかりやすいエンドポイント名

C. AWS データモデル定義に準拠する 1 つ以上のデバイスタイプ。

D. エンドポイントを作成したら、適切なクラスター固有の登録関数を使用してクラスターを登録します。

E. 各クラスター登録には、属性とコマンドのコールバック関数が必要です。システムはユーザーコンテキストポインタをコールバックに渡して、呼び出し間の状態を維持します。

```
struct DeviceState
{
    struct iotmiDev_Agent * agent;
    struct iotmiDev_Endpoint *endpoint1;

    /* OnOff cluster states*/
    bool hwState;
};


/* This implementation for OnOff getter just reads
   the state from the DeviceState */
iotmiDev_DMStatus exampleGetOnOff( bool * value, void * user )
{
    struct DeviceState * state = ( struct DeviceState * ) ( user );
    *value = state->hwState;
    printf( "%s(): state->hwState: %d\n", __func__, state->hwState );
    return iotmiDev_DMStatusOk;
}

iotmiDev_DMStatus exampleGetOnTime( uint16_t * value, void * user )
{
    *value = 0;
    printf( "%s(): OnTime is %u\n", __func__, *value );
    return iotmiDev_DMStatusOk;
}

iotmiDev_DMStatus exampleGetStartUpOnOff( iotmiDev_OnOff_StartUpOnOffEnum *
 value, void * user )
{
    *value = iotmiDev_OnOff_StartUpOnOffEnum_Off;
    printf( "%s(): StartUpOnOff is %d\n", __func__, *value );
    return iotmiDev_DMStatusOk;
}
```

```
void setupOnOff( struct DeviceState *state )
{
    struct iotmiDev_clusterOnOff clusterOnOff = {
        .getOnOff = exampleGetOnOff,
        .getOnTime = exampleGetOnTime,
        .getStartUpOnOff = exampleGetStartUpOnOff,
    };
    iotmiDev_OnOffRegisterCluster( state->endpoint1,
                                   &clusterOnOff,
                                   ( void * ) state);
}


/* Here is the sample setting up an endpoint 1 with OnOff
   cluster. Note all error handling code is omitted. */
void setupAgent(struct DeviceState *state)
{
    struct iotmiDev_Agent_Config config = {
        .thingId = IOTMI_DEVICE_MANAGED_THING_ID,
        .clientId = IOTMI_DEVICE_CLIENT_ID,
    };
    iotmiDev_Agent_InitDefaultConfig(&config);

    /* Create a device agent before calling other SDK APIs */
    state->agent = iotmiDev_Agent_new(&config);

    /* Create endpoint#1 */
    state->endpoint1 = iotmiDev_Agent_addEndpoint( state->agent,
                                                   1,
                                                   "Data Model Handler Test
 Device",
                                                   (const char*[])
{ "Camera" },
                                                   1 );
    setupOnOff(state);
}
```

c. 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

i. 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;
}
```

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

iii. 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。

iv. 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." ) );

}
```

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. 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リファレンス: 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非ハードウェア固有の 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**
>
> 最も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。

テクニカルリファレンス

トピック

- リファレンス: サポートされているプラットフォーム
- リファレンス: 技術要件
- リファレンス: 共通 API

リファレンス: 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	Linux
アンバレラ	Armv8 (AArch64)	Linux
AmebaD	Armv8-M 32 ビット	FreeRTOS
ESP32S3	Xtensa LX7 32 ビット	FreeRTOS

リファレンス: 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 (使用バイト)
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)

リファレンス: 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);
```

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 、セキュリティを最も重視する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 お客様とお客様の間の責任共有です。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。 は、お客様が安全に使用できるサービス AWS 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に責任共有モデルを適用する方法を理解するのに役立ちます。以下のトピックでは、セキュリティとコンプライアンスの目的を達成するようにマネージド統合を設定する方法について説明します。また、マネージド統合リソースのモニタリングや保護に役立つ他の 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 アイデンティティセンターまたは 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、 内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

のマネージド統合は、デフォルトで AWS IoT Device Management は暗号化キーを使用して保管中の機密データを暗号化します。

マネージド型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、または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 AWS KMS リクエストGenerateDataKeyWithoutPlaintext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。
- 物理デバイスを表すマネージドモノと、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を設定する。
- 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 マネジメントコンソール 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 用に 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 マネジメントコンソール

で暗号化設定を更新するには AWS マネジメントコンソール、 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 Identity and Access Management

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

オーディエンス

AWS Identity and Access Management (IAM) の使用方法は、ロールによって異なります。

- サービスユーザー - 機能にアクセスできない場合は、管理者にアクセス許可をリクエストします (「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」を参照)。
- サービス管理者 - ユーザーアクセスを決定し、アクセス許可リクエストを送信する (「マネージド統合と IAM の連携方法」を参照)
- IAM 管理者 - アクセスを管理するためのポリシーを作成します (「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」を参照)

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。、IAM ユーザー 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で認証される必要があります。

AWS IAM アイデンティティセンター (IAM Identity Center)、シングルサインオン認証、Google/Facebook 認証情報などの ID ソースからの認証情報を使用して、フェデレーティッド ID としてサイ

ンインできます。サインインの詳細については、「AWS サインイン ユーザーガイド」の「AWS アカウントへのサインイン方法」を参照してください。

プログラムによるアクセスの場合、 は SDK と CLI AWS を提供してリクエストを暗号化して署名します。詳細については、「IAM ユーザーガイド」の「API リクエストに対するAWS 署名バージョン 4」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、まず、すべての AWS のサービス および リソースへの完全なアクセス権を持つ AWS アカウント root ユーザーと呼ばれる 1 つのサインインアイデンティティから始めます。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。ルートユーザー認証情報を必要とするタスクについては、「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。

フェデレーテッドアイデンティティ

ベストプラクティスとして、人間のユーザーが一時的な認証情報 AWS のサービス を使用して にアクセスするには、ID プロバイダーとのフェデレーションを使用する必要があります。

フェデレーティッド ID は、エンタープライズディレクトリ、ウェブ ID プロバイダー、または ID ソースの認証情報 AWS のサービス を使用して Directory Service にアクセスするユーザーです。フェデレーテッドアイデンティティは、一時的な認証情報を提供するロールを引き受けます。

アクセスを一元管理する場合は、 AWS IAM アイデンティティセンターをお勧めします。詳細については、「AWS IAM アイデンティティセンター ユーザーガイド」の「IAM アイデンティティセンターとは」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、1 人のユーザーまたは 1 つのアプリケーションに対して特定のアクセス許可を持つアイデンティティです。長期的な認証情報を持つ IAM ユーザーの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。詳細については、IAM ユーザーガイドの「ID プロバイダーとのフェデレーションを使用して にアクセスすることを人間のユーザーに要求する AWS」を参照してください。

IAM グループは IAM ユーザーのコレクションを指定し、大量のユーザーのアクセス許可の管理を容易にします。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、一時的な認証情報を提供する特定のアクセス許可を持つ ID です。ユーザーから IAM ロール (コンソール) に切り替えるか、 または API オペレーションを呼び出すことで、ロールを引き受けることができます。 AWS CLI AWS 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 ロールは、フェデレーションユーザーアクセス、一時的な IAM ユーザーアクセス許可、クロスアカウントアクセス、クロスサービスアクセス、および Amazon EC2 で実行されているアプリケーションに役立ちます。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。

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、アイデンティティまたはリソースに関連付けられたときにアクセス許可を定義します。は、プリンシパル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。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。

ポリシーを使用して、管理者は、どのプリンシパルがどのリソースに対してどの条件でアクションを実行できるかを定義することで、誰が何にアクセスできるかを指定します。

デフォルトでは、ユーザーやロールにアクセス許可はありません。IAM 管理者は IAM ポリシーを作成し、ユーザーが引き受けることができるロールに追加します。IAM ポリシーは、オペレーションの実行方法を問わずアクセス許可を定義します。

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のポリシーは、アイデンティティ (ユーザー、グループ、ロール) にアタッチする JSON アクセス許可ポリシードキュメントです。これらのポリシーは、アイデンティティが実行できるアクション、リソース、および条件を制御します。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、インラインポリシー (単一の ID に直接埋め込む) または管理ポリシー (複数の ID にアタッチされたスタンドアロンポリシー) にすることができます。管理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。例としては、IAM ロールの信頼ポリシーや Amazon S3 バケットポリシーなどがあります。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。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。

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。

その他のポリシータイプ

AWS は、より一般的なポリシータイプによって付与されるアクセス許可の上限を設定できる追加のポリシータイプをサポートしています。

- アクセス許可の境界 – アイデンティティベースのポリシーで IAM エンティティに付与できるアクセス許可の上限を設定します。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 (SCP) – AWS Organizationsにおいて組織または組織単位のアクセス許可の上限を指定します。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 を参照してください。
- リソースコントロールポリシー (RCP) – は、アカウント内のリソースで利用できる最大数のアクセス許可を定義します。詳細については、AWS Organizations ユーザーガイド の リソースコントロールポリシー (RCP) を参照してください。
- セッションポリシー – ロールまたはフェデレーションユーザーの一時セッションを作成する際にパラメータとして渡される高度なポリシーです。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アクセス許可を理解するのがさらに難しくなります。が複数のポリシータイプが関与す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損なわれ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 をサポートしています。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 マネジメントコンソール、「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。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iotmanagedintegrations:*",
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": "iam:CreateServiceLinkedRole",
      "Resource": "arn:aws:iam::*:role/aws-service-role/
iotmanagedintegrations.amazonaws.com/AWSServiceRoleForIoTManagedIntegrations",
      "Condition": {
        "StringEquals": {
          "iam:AWSServiceName": "iotmanagedintegrations.amazonaws.com"
        }
      }
    }
  ]
}
```

AWS 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 マネジメントコンソール、「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。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "CloudWatchLogs",
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup"
      ],
      "Resource": [
        "arn:aws:logs:*:*:log-group:/aws/iotmanagedintegrations/*"
      ],
      "Condition": {
        "StringEquals": {
          "aws:PrincipalAccount": "${aws:ResourceAccount}"
        }
      }
    },
    {
      "Sid": "CloudWatchStreams",
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogStream",
        "logs:PutLogEvents"
      ],
      "Resource": [
        "arn:aws:logs:*:*:log-group:/aws/iotmanagedintegrations/*:log-stream:*"
      ],
      "Condition": {
```

```
        "StringEquals": {
          "aws:PrincipalAccount": "${aws:ResourceAccount}"
        }
      }
    },
    {
      "Sid": "CloudWatchMetrics",
      "Effect": "Allow",
      "Action": [
        "cloudwatch:PutMetricData"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "cloudwatch:namespace": [
            "AWS/IoTManagedIntegrations",
            "AWS/Usage"
          ]
        }
      }
    }
  ]
}
```

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の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しました	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 機能を確認してください。

マネージド統合で使用できる IAM 機能

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

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、ユーザーグループ、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。

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。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コントロールできます。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、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、全体のアカウント、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。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。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。ベストプラクティスとして、Amazon リソースネーム (ARN) を使用してリソースを指定します。リソースレベルのアクセス許可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 要素は、定義された基準に基づいてステートメントが実行される時期を指定します。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。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アクセス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、タグと呼ばれる属性に基づいてアクセス許可を定義する認可方法です。IAM エンティティと AWS リソースにタグをアタッチし、プリンシパルのタグがリソースのタグと一致するときにオペレーションを許可するように ABAC ポリシーを設計できます。

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

一時的な認証情報は、 AWS リソースへの短期的なアクセスを提供し、フェデレーションまたは切り替えロールの使用時に自動的に作成されます。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 ユーザーガイド」の「IAM の一時的な認証情報」および「AWS のサービス と IAM との連携」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

転送アクセスセッション (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。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。

これらのサンプルの 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、アカウント内の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では、 AWS アカウントに費用が発生する場合があります。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 管理ポリシーを開始し、最小特権のアクセス許可に移行 – 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 ポリシーでアクセス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。たとえば、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 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 アクセスアナライザー 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 アクセスアナライザー 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、MFA をオンにしてセキュリティを強化します。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アクセス許可が必要です。

次の例のエラーは、 という IAM 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`marymajor`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アクセス許可が必要です。Mary には、ロールをサービスに渡すアクセス許可がありません。

```
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 している 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 している別の 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、リソースにアクセスするため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.

JSON

```
{
  "Version":"2012-10-17",
  "Statement" : [
    {
      "Sid" : "CloudWatchLogs",
      "Effect" : "Allow",
      "Action" : [
        "logs:CreateLogGroup",
        "logs:DescribeLogGroups"
      ],
      "Resource" : [
        "arn:aws:logs:*:*:log-group:/aws/iotmanagedintegrations/*"
      ]
    },
    {
      "Sid" : "CloudWatchStreams",
      "Effect" : "Allow",
      "Action" : [
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams"
      ],
      "Resource" : [
        "arn:aws:logs:*:*:log-group:/aws/iotmanagedintegrations/*:log-stream:*"
      ]
    },
```

```
    {
      "Sid" : "CloudWatchMetrics",
      "Effect" : "Allow",
      "Action" : [
        "cloudwatch:PutMetricData"
      ],
      "Resource" : "*",
      "Condition" : {
        "StringEquals" : {
          "cloudwatch:namespace" : [
            "AWS/IoTManagedIntegrations",
            "AWS/Usage"
          ]
        }
      }
    }
  ]
}
```

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 マネジメントコンソール、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。

C2C ワークフローのデータ保護 AWS Secrets Manager に を使用する

AWS Secrets Manager は、データベース認証情報、API キー、およびその他のシークレット情報を保護するために使用できるシークレットストレージサービスです。次に、コード内で、ハードコードされた認証情報を Secrets Manager への API コールに置き換えることができます。シークレットはそこに存在しないため、お客様のコードを調べている誰かがシークレットを漏らさないようにするのに役立ちます。この概要については、「AWS Secrets Manager ユーザーガイド」を参照してください

Secrets Manager は、 AWS Key Management Service キーを使用してシークレットを暗号化します。詳細については、「AWS Key Management Serviceのシークレットの暗号化と復号」を参照してください。

のマネージド統合は と AWS IoT Device Management 統合 AWS Secrets Manager されるため、データを Secrets Manager に保存し、設定でシークレット ID を使用できます。

マネージド統合でシークレットを使用する方法

Open Authorization (OAuth) は、委任されたアクセス認可のオープンスタンダードであり、ユーザーはパスワードを共有せずにウェブサイトやアプリケーションに他のウェブサイトの情報へのアクセスを許可できます。これは、サードパーティーアプリケーションがユーザーに代わってユーザーデータにアクセスする安全な方法であり、パスワードを共有するよりも安全な方法を提供します。

OAuth では、クライアント ID とクライアントシークレットは、アクセストークンをリクエストするときにクライアントアプリケーションを識別して認証する認証情報です。

のマネージド統合 AWS IoT Device Management は、OAuth を使用して C2C ワークフローを使用する顧客と通信します。お客様は、通信するためにクライアント ID とクライアントシークレットを指定する必要があります。マネージド型統合のお客様はクライアント ID とクライアントシークレットを AWS アカウントに保存し、マネージド型統合はお客様のアカウントのクライアント ID とクライアントシークレットを読み取ります。

シークレットを作成する方法

シークレットを作成するには、 AWS Secrets Manager 「 ユーザーガイド」の「 AWS Secrets Manager シークレットを作成する」の手順に従います。

シークレット値を読み取るには、マネージド統合用のカスタマーマネージド AWS KMS キーを使用してシークレットを作成する必要があります。詳細については、「 AWS Secrets Manager ユーザーガイド」の「 AWS KMS キーのアクセス許可」を参照してください。

また、次のセクションで IAM ポリシーを使用する必要があります。

シークレットを取得 AWS IoT Device Management するための のマネージド統合へのアクセスを許可する

マネージド統合が Secrets Manager からシークレット値を取得できるようにするには、シークレットの作成時にシークレットのリソースポリシーに次のアクセス許可を含めます。

JSON

```
{
  "Version":"2012-10-17",
  "Statement" : [ {
    "Effect" : "Allow",
    "Principal" : {
      "Service" : "iotmanagedintegrations.amazonaws.com"
    },
    "Action" : [ "secretsmanager:GetSecretValue" ],
    "Resource" : "*" ,
    "Condition": {
        "StringEquals": {
          "aws:SourceArn": "arn:aws:iotmanagedintegrations:AWS Region:account-
id:account-association:account-association-id"

        }
      }
  } ]
}
```

カスタマーマネージド AWS KMS キーのポリシーに次のステートメントを追加します。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [{
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt",
                "kms:DescribeKey"
            ],
            "Principal": {
                "Service": [
                    "iotmanagedintegrations.amazonaws.com"
                ]
            },
            "Resource": [
            "arn:aws:kms:us-east-1:123456789012:key/*"
            ]
        }

    ]
}
```

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、「コンプライアンスAWS のサービス プログラムによるスコープ」の「コンプライアンス」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。を使用する際のコンプライアンス責任の詳細については AWS のサービス、AWS 「 セキュリティドキュメント」を参照してください。

インターフェイス VPC エンドポイントとのマネージド統合を使用する

インターフェイス Amazon VPC エンドポイントを作成することで、Amazon VPC と AWS IoT マネージド統合間のプライベート接続を確立できます。インターフェイスエンドポイントは AWS PrivateLink、プライベート IP アドレスを使用して サービスにプライベートにアクセスできるテクノロジーである を利用しています。AWS PrivateLink は、VPC と IoT マネージド統合間のすべてのネットワークトラフィックを Amazon ネットワークに制限します。インターネットゲートウェイ、NAT デバイス、VPN 接続は必要ありません。

を使用する必要はありませんが AWS PrivateLink、推奨されます。 AWS PrivateLink および VPC エンドポイントの詳細については、「 AWS PrivateLink ガイド」の「 による AWS サービスへのアクセス AWS PrivateLink」を参照してください。

トピック

- AWS IoT マネージド統合 VPC エンドポイントに関する考慮事項
- AWS IoT マネージド統合用のインターフェイス VPC エンドポイントの作成
- VPC エンドポイントのテスト
- VPC エンドポイントを介した サービスへのアクセスの制御
- 料金
- 制限

AWS IoT マネージド統合 VPC エンドポイントに関する考慮事項

AWS IoT マネージド統合用のインターフェイス VPC エンドポイントを設定する前に、「 AWS PrivateLink ガイド」の「インターフェイスエンドポイントのプロパティと制限」を参照してください。

AWS IoT マネージド統合では、インターフェイス VPC エンドポイントを介して VPC からすべての API アクションを呼び出すことができます。

サポートされているエンドポイント

AWS IoT マネージド統合は、次のサービスインターフェイスの VPC エンドポイントをサポートします。

- コントロールプレーン API: `com.amazonaws.region.iotmanagedintegrations.api`

サポートされていないエンドポイント

以下の AWS IoT マネージド統合エンドポイントは VPC エンドポイントをサポートしていません。

- MQTT エンドポイント: MQTT デバイスは、通常、 AWS VPCs 内ではなくエンドユーザー環境にデプロイされるため、 AWS PrivateLink 統合は不要です。
- OAuth コールバックエンドポイント: 多くのサードパーティープラットフォームは AWS インフラストラクチャ内で動作しないため、OAuth フロー AWS PrivateLink のサポートのメリットが減ります。

利用可能な状況

AWS IoT マネージド統合 VPC エンドポイントは、次の AWS リージョンで使用できます。

- カナダ (中部) ca-central-1
- ヨーロッパ (アイルランド) eu-west-1

AWS IoT マネージド統合の可用性が拡大すると、追加のリージョンがサポートされます。

デュアルスタックのサポート

AWS IoT マネージド統合 VPC エンドポイントは、IPv4 トラフィックと IPv6 トラフィックの両方をサポートします。次の IP アドレスタイプを使用して VPC エンドポイントを作成できます。

- IPv4: エンドポイントネットワークインターフェイスに IPv4 アドレスを割り当てる
- IPv6: エンドポイントネットワークインターフェイスに IPv6 アドレスを割り当てる (IPv6-onlyサブネットが必要)
- デュアルスタック: IPv4 アドレスと IPv6 アドレスの両方をエンドポイントネットワークインターフェイスに割り当てます

AWS IoT マネージド統合用のインターフェイス VPC エンドポイントの作成

Amazon VPC コンソールまたは AWS CLI (AWS CLI) を使用して、 AWS IoT マネージド統合サービスの VPC エンドポイントを作成できます。

AWS IoT マネージド統合用のインターフェイス VPC エンドポイントを作成するには (コンソール)

1. Amazon VPC コンソールで Amazon VPC コンソールを開きます。
2. ナビゲーションペインで、[エンドポイント] を選択します。
3. [エンドポイントの作成] を選択します。
4. [Service category] (サービスカテゴリ) で、[AWS services] (のサービス) を選択します。
5. サービス名で、 AWS リージョンに対応するサービス名を選択します。例:
   - `com.amazonaws.ca-central-1.iotmanagedintegrations.api`
   - `com.amazonaws.eu-west-1.iotmanagedintegrations.api`
6. VPC の場合は、 AWS IoT マネージド統合にアクセスする VPC を選択します。
7. 追加設定では、デフォルトで Enable DNS name が選択されています。この設定はそのままにしておくことをお勧めします。これにより、 AWS IoT マネージド統合パブリックサービスエンドポイントへのリクエストが Amazon VPC エンドポイントに解決されます。
8. [サブネット] には、エンドポイントネットワークインターフェイスの作成先となるサブネットを選択します。アベイラビリティーゾーンごとにサブネットを 1 つ選択できます。
9. [IP address type] (IP アドレスのタイプ) で、次のオプションから選択します。
   - IPv4: エンドポイントネットワークインターフェイスに IPv4 アドレスを割り当てる
   - IPv6: エンドポイントネットワークインターフェイスに IPv6 アドレスを割り当てる (選択したすべてのサブネットが IPv6-onlyである場合にのみサポート)
   - デュアルスタック: IPv4 アドレスと IPv6 アドレスの両方をエンドポイントネットワークインターフェイスに割り当てる
10. [Security groups] (セキュリティグループ) で、エンドポイントネットワークインターフェイスに関連付けるセキュリティグループを選択します。セキュリティグループルールでは、エンドポイントネットワークインターフェイスと、 サービスと通信する VPC 内のリソース間の通信を許可する必要があります。
11. ポリシー でフルアクセスを選択し、インターフェイスエンドポイント上のすべてのリソースに対するすべてのプリンシパルによるすべてのオペレーションを許可します。アクセスを制限するには、カスタム を選択し、ポリシーを指定します。
12. (オプション) タグを追加するには、[Add new tag] (新しいタグを追加) を選択し、そのタグのキーと値を入力します。
13. [エンドポイントの作成] を選択します。

IoT マネージド統合のインターフェイス VPC エンドポイントを作成するには (AWS CLI)

create-vpc-endpoint コマンドを使用して、VPC ID、VPC エンドポイントタイプ (インターフェイス) 、サービス名、エンドポイントを使用するサブネット、およびエンドポイントネットワークインターフェイスに関連付けるセキュリティグループを指定します。

```
aws ec2 create-vpc-endpoint \
  --vpc-id vpc-12345678 \
  --route-table-ids rtb-12345678 \
  --service-name com.amazonaws.ca-central-1.iotmanagedintegrations.api \
  --vpc-endpoint-type Interface \
  --subnet-ids subnet-12345678 subnet-87654321 \
  --security-group-ids sg-12345678
```

VPC エンドポイントのテスト

VPC エンドポイントを作成したら、VPC 内の EC2 インスタンスから AWS IoT マネージド統合に API コールを行うことで、接続をテストできます。

前提条件

- VPC 内のプライベートサブネット内の EC2 インスタンス
- AWS IoT マネージド統合オペレーションの適切な IAM アクセス許可
- VPC エンドポイントへの HTTPS トラフィック (ポート 443) を許可するセキュリティグループルール

接続をテストする

1. プライベートサブネットの Amazon EC2 インスタンスに接続します。
2. プライベート DNS 名の DNS 解決を確認します。

```
dig api.iotmanagedintegrations.region.api.aws
```

3. HTTPS 接続をテストする:

```
curl -v https://api.iotmanagedintegrations.region.api.aws
```

4. AWS IoT マネージド統合 API コールを実行します。

```
aws iot-managed-integrations list-destinations \
  --region region \
  --endpoint-url https://api.iotmanagedintegrations.region.api.aws
```

を AWS リージョン (などca-central-1) regionに置き換えます。

VPC エンドポイントを介した サービスへのアクセスの制御

VPC エンドポイントポリシーは、エンドポイントを作成または変更するときにインターフェイス VPC エンドポイントにアタッチする IAM リソースポリシーです。エンドポイントの作成時にポリシーをアタッチしない場合、サービスへのフルアクセスを許可するデフォルトのポリシーがアタッチされます。エンドポイントポリシーは、IAM ユーザーポリシーやサービス固有のポリシーを上書き、または置き換えません。これは、評価項目から指定されたサービスへのアクセスを制御するための別のポリシーです。

評価項目のポリシーは、JSON形式で記載する必要があります。詳細については、Amazon VPC ユーザーガイドのVPC エンドポイントによるサービスのアクセスコントロールを参照してください。

例: AWS IoT マネージド統合アクションの VPC エンドポイントポリシー

AWS IoT マネージド統合のエンドポイントポリシーの例を次に示します。このポリシーは、VPC エンドポイントを介して AWS IoT マネージド統合に接続するユーザーに送信先へのアクセスを許可しますが、認証情報ロッカーへのアクセスを拒否します。

```
{
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": [
                "iotmanagedintegrations:ListDestinations",
                "iotmanagedintegrations:GetDestination",
                "iotmanagedintegrations:CreateDestination",
                "iotmanagedintegrations:UpdateDestination",
                "iotmanagedintegrations:DeleteDestination"
            ],
            "Resource": "*"
        },
```

```
        {
            "Effect": "Deny",
            "Principal": "*",
            "Action": [
                "iotmanagedintegrations:ListCredentialLockers",
                "iotmanagedintegrations:GetCredentialLocker",
                "iotmanagedintegrations:CreateCredentialLocker",
                "iotmanagedintegrations:UpdateCredentialLocker",
                "iotmanagedintegrations:DeleteCredentialLocker"
            ],
            "Resource": "*"
        }
    ]
}
```

例: 特定の IAM ロールへのアクセスを制限する VPC エンドポイントポリシー

次の VPC エンドポイントポリシーは、信頼チェーンに指定された IAM ロールを持つ IAM プリンシパルに対してのみ、 AWS IoT マネージド統合へのアクセスを許可します。他のすべての IAM プリンシパルはアクセスを拒否されます。

```
{
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "*",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:PrincipalArn": "arn:aws:iam::123456789012:role/
IoTManagedIntegrationsVPCRole"
                }
            }
        }
    ]
}
```

料金

AWS IoT マネージド統合でインターフェイス VPC エンドポイントを作成して使用すると、標準料金が課金されます。詳細については、AWS PrivateLink 料金表を参照してください。

制限

- CreateAccountAssociation API は、Amazon ネットワークを離れるリクエストを必要とするサードパーティーのクラウドサービスで OAuth を実行するように設計されています。これは、 を使用して VPC 内にトラフィックを格納 AWS PrivateLink するお客様にとって重要です。これは、 がこの API コールにend-to-endの封じ込めを提供 AWS PrivateLink できないためです。
- AWS IoT マネージド統合の VPC エンドポイントは では使用できません AWS GovCloud (US) Regions。

一般的な VPC エンドポイントの制限については、「Amazon VPC ユーザーガイド」の「インターフェイスエンドポイントのプロパティと制限」を参照してください。

FIPS AWS IoT Device Management エンドポイントのマネージド統合に接続する

AWS IoT は、連邦情報処理規格 (FIPS) 140-2 をサポートするコントロールプレーンエンドポイントを提供します。FIPS 準拠のエンドポイントは、標準 AWS エンドポイントとは異なります。FIPS 準拠の方法で の AWS IoT Device Management マネージド統合を操作するには、FIPS 準拠のクライアントで以下で説明するエンドポイントを使用する必要があります。 AWS IoT コンソールは FIPS に準拠していません。

以下のセクションでは、REST API、 SDK、または を使用して FIPS 準拠の AWS IoT エンドポイントにアクセスする方法について説明します AWS CLI。

コントロールプレーンエンドポイント

マネージド統合オペレーションをサポートする FIPS 準拠のコントロールプレーンエンドポイントとその関連 AWS CLI コマンドは、FIPS Endpoints by Service に記載されています。FIPS Endpoints by Service で、 AWS IoT Device Management - Managed Integrations サービスを検索し、 のエンドポイントを検索します AWS リージョン。

マネージド統合オペレーションにアクセスするときに FIPS 準拠のエンドポイントを使用するには、に適したエンドポイントで AWS SDK または REST API を使用します AWS リージョン。

マネージド統合 CLI コマンドを実行するときに FIPS 準拠のエンドポイントを使用するには、 に適切なエンドポイントを持つ --endpointパラメータ AWS リージョン を コマンドに追加します。

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。

- 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。呼び出し元のユーザーとアカウント AWS、呼び出し元の送信元 IP アドレス、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。

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、アカウントを作成する AWS アカウント 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すべての証跡 AWS マネジメントコンソール はマルチリージョンです。 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、以下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 CloudTrail に記録します。

- `CreateCloudConnector`
- `UpdateCloudConnector`
- `GetCloudConnector`
- `DeleteCloudConnector`
- `ListCloudConnectors`
- `CreateConnectorDestination`
- `UpdateConnectorDestination`
- `GetConnectorDestination`
- `DeleteConnectorDestination`
- `ListConnectorDestinations`
- `CreateAccountAssociation`
- `UpdateAccountAssociation`
- `GetAccountAssociation`
- `DeleteAccountAssociation`
- `ListAccountAssociations`
- `StartAccountAssociationRefresh`
- `ListManagedThingAccountAssociations`
- `RegisterAccountAssociation`
- `DeregisterAccountAssociation`
- `SendConnectorEvent`
- `ListDeviceDiscoveries`
- `ListDiscoveredDevices`

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、正常な `CreateCloudConnector` API オペレーションを示す CloudTrail イベントを示しています。

`CreateCloudConnector` 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": "EXAMPLE",
        "arn": "arn:aws:sts::111122223333:assumed-role/Admin/EXAMPLE",
        "accountId": "111122223333",
        "accessKeyId": "EXAMPLEKYSBQSCGRIC",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROAZOZQFKYSFZVB2J2GN",
                "arn": "arn:aws:iam::111122223333:role/Admin",
                "accountId": "111122223333",
                "userName": "Admin"
            },
            "attributes": {
                "creationDate": "2025-06-05T18:26:16Z",
                "mfaAuthenticated": "false"
            }
        }
    },
    "eventTime": "2025-06-05T18:30:40Z",
    "eventSource": "iotmanagedintegrations.amazonaws.com",
    "eventName": "CreateCloudConnector",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "PostmanRuntime/7.44.0",
    "requestParameters": {
        "EndpointType": "LAMBDA",
        "Description": "Manual testing for C2C CT Validation",
```

```
        "ClientToken": "abc7460",
        "EndpointConfig": {
            "lambda": {
                "arn": "arn:aws:lambda:us-
east-1:111122223333:function:LightweightMockConnector7460"
            }
        },
        "Name": "EdenManualTestCloudConnector"
    },
    "responseElements": {
        "X-Frame-Options": "DENY",
        "Access-Control-Expose-Headers": "Content-Length,Content-Type,X-Amzn-
Errortype,X-Amzn-Requestid",
        "Strict-Transport-Security": "max-age:47304000; includeSubDomains",
        "Cache-Control": "no-store, no-cache",
        "X-Content-Type-Options": "nosniff",
        "Content-Security-Policy": "upgrade-insecure-requests; default-src 'none';
 object-src 'none'; frame-ancestors 'none'; base-uri 'none'",
        "Pragma": "no-cache",
        "Id": "f7e633e719404c4a933596b4d0cc276e",
        "Arn": "arn:aws:iotmanagedintegrations:us-east-1:111122223333:cloud-connector/
EXAMPLE404c4a933596b4d0cc276e"
    },
    "requestID": "c0071fd1-b8e0-400a-bcc0-EXAMPLE9e4",
    "eventID": "95b318ea-2f63-4183-9c22-EXAMPLE3e",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```


次の例は、正常な ListDiscoveredDevices API オペレーションを示す CloudTrail イベントを示しています。

ListDiscoveredDevices 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": "EZAMPLE",
        "arn": "arn:aws:sts::444455556666:assumed-role/Admin/EXAMPLE",
```

```
        "accountId": "444455556666",
        "accessKeyId": "EXAMPLERJ26PYMH",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "EXAMPLE",
                "arn": "arn:aws:iam::444455556666:role/Admin",
                "accountId": "444455556666",
                "userName": "Admin"
            },
            "attributes": {
                "creationDate": "2025-06-10T23:37:31Z",
                "mfaAuthenticated": "false"
            }
        }
    },
    "eventTime": "2025-06-10T23:38:07Z",
    "eventSource": "iotmanagedintegrations.amazonaws.com",
    "eventName": "ListDiscoveredDevices",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "EXAMPLE-runtime/2.4.0",
    "requestParameters": {
        "Identifier": "EXAMPLE4f268483a17d8060f014"
    },
    "responseElements": null,
    "requestID": "27ae1f61-e2e6-43e4-bf17-EXAMPLEa568",
    "eventID": "34734e81-76a8-49a4-9641-EXAMPLE28ed",
    "readOnly": true,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "444455556666",
    "eventCategory": "Management"
}
```


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。

マネージドインテグレーション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
一般提供リリース	マネージドインテグレーションデベロッパーガイドの一般提供リリース	2025 年 6 月 25 日
初期プレビューリリース	マネージドインテグレーションデベロッパーガイドの初回プレビューリリース	2025 年 3 月 3 日

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。