aws

开发人员指南

的托管集成 AWS IoT Device Management

的托管集成 AWS IoT Device Management: 开发人员指南

Table of Contents

托管集成有什么用 AWS IoT Device Management？

的托管集成可 AWS IoT Device Management 帮助物联网解决方案提供商统一对来自数百家制造商的物联网设备的控制和管理。您可以使用托管集成来自动执行设备设置工作流程，并支持许多设备之间的互操作性，无论设备供应商或连接协议如何。这允许解决方案提供商使用单一用户界面和一组用户界面 APIs 来控制、管理和操作一系列设备。

主题

您是首次使用托管集成的用户吗？

如果您是首次使用托管集成的用户，我们建议您先阅读以下章节：

- 设置托管集成
- 开始使用托管集成 AWS IoT Device Management

托管集成概述

下图提供了托管集成的高级概述

托管集成术语

在托管集成中，有许多概念和术语对于管理自己的设备实现至关重要。以下各节概述了这些关键概念和术语，以便更好地理解托管集成。

通用托管集成术语

与事物相比，托管集成需要理解的一个重要概念是托管 AWS IoT Core 事物。

- AWS IoT Core 事物： AWS IoT Core 事物是一种提供数字表示的 AWS IoT Core 结构。开发人员需要管理策略、数据存储、规则、操作、MQTT 主题以及向数据存储传输设备状态。有关什么是 AWS IoT Core 事物的更多信息，请参阅使用管理设备 AWS IoT。

- 托管集成托管事物：通过托管事物，我们提供了一个抽象来简化设备交互，并且不需要开发人员创建规则、操作、MQTT 主题和策略等项目。

Cloud-to-cloud 术语

与托管集成集成的物理设备可能来自第三方云提供商。为了将这些设备加入托管集成并与第三方云提供商通信，以下术语涵盖了支持这些工作流程的一些关键概念：

- Cloud-to-cloud (C2C) 连接器：C2C 连接器在托管集成和第三方云提供商之间建立连接。
- 第三方云提供商：对于在托管集成之外制造和管理的设备，第三方云提供商允许最终用户控制这些设备，而托管集成则与第三方云提供商就各种工作流程（例如设备命令）进行通信。

数据模型术语

托管集成使用数据模型来组织数据和设备之间的 end-to-end通信。以下术语涵盖了理解这两种数据模型的一些关键概念：

- 设备：代表物理设备（例如可视门铃）的实体，该实体具有多个节点协同工作以提供完整的功能集。
- 端点：端点封装了一项独立功能（铃声、运动检测、可视门铃中的照明）。
- 功能：一种实体，代表在端点中提供功能所需的组件（按钮或可视门铃的灯光和铃声功能）。
- 动作：代表与设备功能的交互的实体（按铃或查看谁在门口）。
- 事件：代表来自设备功能的事件的实体。设备可以发送事件来报告门incident/alarm, an activity from a sensor etc. (e.g. there is knock/ring上的）。
- 属性：表示设备状态下特定属性的实体（铃响了，门廊灯亮了，摄像机正在录制）。
- 数据模型：数据层对应于有助于支持应用程序功能的数据和动词元素。当有意与设备交互时，应用程序会对这些数据结构进行操作。欲了解更多信息，请参阅网站上的 c onnectedhomeip。GitHub
- 架构：架构是以 JSON 格式表示数据模型。

设置托管集成

以下各节将指导您完成使用托管集成的初始设置。 AWS IoT Device Management

主题

注册获取 AWS 账户

如果您没有 AWS 账户，请完成以下步骤来创建一个。

要注册 AWS 账户

1. 打开https://portal.aws.amazon.com/billing/注册。
2. 按照屏幕上的说明操作。

   在注册时，将接到电话或收到短信，要求使用电话键盘输入一个验证码。

   当您注册时 AWS 账户，就会创建AWS 账户根用户一个。根用户有权访问该账户中的所有 AWS 服务 和资源。作为最佳安全实践，请为用户分配管理访问权限，并且只使用根用户来执行需要根用户访问权限的任务。

AWS 注册过程完成后会向您发送一封确认电子邮件。您可以随时前往 https://aws.amazon.com/并选择“我的账户”，查看您当前的账户活动并管理您的账户。

创建具有管理访问权限的用户

注册后，请保护您的安全 AWS 账户 AWS 账户根用户 AWS IAM Identity Center，启用并创建管理用户，这样您就不会使用 root 用户执行日常任务。

保护你的 AWS 账户根用户

1. 选择 Root 用户并输入您的 AWS 账户 电子邮件地址，以账户所有者的身份登录。AWS Management Console在下一页上，输入您的密码。

要获取使用根用户登录方面的帮助，请参阅《AWS 登录 用户指南》中的 Signing in as the root user。

2. 为您的根用户启用多重身份验证（MFA）。

   有关说明，请参阅 I A M 用户指南中的为 AWS 账户 根用户启用虚拟 MFA 设备（控制台）。

创建具有管理访问权限的用户

1. 启用 IAM Identity Center。

   有关说明，请参阅《AWS IAM Identity Center 用户指南》中的 Enabling AWS IAM Identity Center。

2. 在 IAM Identity Center 中，为用户授予管理访问权限。

   有关使用 IAM Identity Center 目录 作为身份源的教程，请参阅《用户指南》 IAM Identity Center 目录中的使用默认设置配置AWS IAM Identity Center 用户访问权限。

以具有管理访问权限的用户身份登录

- 要使用您的 IAM Identity Center 用户身份登录，请使用您在创建 IAM Identity Center 用户时发送到您的电子邮件地址的登录网址。

  有关使用 IAM Identity Center 用户登录的帮助，请参阅AWS 登录 用户指南中的登录 AWS 访问门户。

将访问权限分配给其他用户

1. 在 IAM Identity Center 中，创建一个权限集，该权限集遵循应用最低权限的最佳做法。

   有关说明，请参阅《AWS IAM Identity Center 用户指南》中的 Create a permission set。

2. 将用户分配到一个组，然后为该组分配单点登录访问权限。

   有关说明，请参阅《AWS IAM Identity Center 用户指南》中的 Add groups。

开始使用托管集成 AWS IoT Device Management

以下各节概述了开始使用托管集成需要采取的步骤。

主题

设备类型

托管集成可以管理多种类型的设备。每台设备属于以下三个类别之一：

- 直接连接的设备：此类设备直接连接到托管集成端点。通常，这些设备由设备制造商构建和管理，其中包括用于直接连接的托管集成终端设备 SDK。
- 连接集线器的设备：这些设备通过运行托管集成 Hub SDK 的集线器连接到托管集成，该集线器管理设备发现、入门和控制功能。最终用户可以通过按下按钮启动或扫描条形码来加载这些设备。

  支持以下两个工作流程来加载与集线器连接的设备：

  - 按下最终用户启动的按钮即可开始设备发现
  - 基于条形码的扫描以执行设备关联
- Cloud-to-cloud (C2C) 设备：这些设备由供应商设计和管理，这些供应商维护自己的云基础架构和用于设备控制的品牌移动应用程序。托管集成客户可以访问预先构建的 C2C 连接器目录或创建自己的连接器，以开发通过统一接口与多个第三方供应商云配合使用的物联网解决方案。

  当最终用户首次开启 C2C 设备时，必须向其相应的第三方云提供商进行托管集成，以获取其设备功能和元数据。完成配置工作流程后，托管集成可以代表最终用户与云设备和第三方云提供商进行通信。

> **Note**
>
> 集线器不是上面列出的特定设备类型。其目的是充当智能家居设备的控制器，并促进托管集成与第三方云提供商之间的连接。它既可以作为上面列出的设备类型，也可以用作集线器。

配置 加密密钥

对于在最终用户、托管集成和第三方云之间路由的数据，安全性至关重要。我们支持保护您的设备数据的方法之一是使用安全的 end-to-end加密密钥进行加密，用于路由您的数据。

作为托管集成的客户，您可以使用以下两种方式使用加密密钥：

- 使用默认的托管集成托管加密密钥。
- 提供 AWS KMS key 您创建的。

有关 AWS KMS 服务的更多信息，请参阅密钥管理服务 (KMS)

通过调用《PutDefaultEncryptionConfiguration托管集成 API 参考指南》中的 API，您可以更新要使用的加密密钥选项。默认情况下，托管集成使用默认的托管集成托管加密密钥。您可以随时使用PutDefaultEncryptionConfigurationAPI 更新您的加密密钥配置。

此外，调用 GetDefaultEncryptionConfigurationAPI 命令会返回有关默认或指定区域中 AWS 账户的加密配置的信息。

入职技巧

下面列出了入职的类型：

直接连接的设备上线

预备人有关载入直连设备的步骤，请参阅。

Hub 入职

将您的集线器加入托管集成有关加载集线器的步骤，请参阅。

连接集线器的设备上线

加载设备并在集线器中对其进行操作有关载入连接集线器的设备的步骤，请参阅。

Cloud-to-cloud 设备上线

使用 C2C（云到云）连接器有关将云设备从第三方云供应商加载到托管集成的步骤，请参阅。

设备预调配

设备配置简化了设备上线流程，监督了整个设备生命周期，并为托管集成的其他方面可以访问的设备信息建立集中存储库。托管集成提供了一个用于管理各种设备类型的统一接口，可容纳通过设备软件开发套件 (SDK) 或通过集线器设备间接链接的 commercial-off-the-shelf (COTS) 设备直接连接的第一方客户设备。

托管集成中的每台设备，无论设备类型如何，都有一个名为 a `managedThingId` 的全球唯一标识符。在整个设备生命周期中，该标识符用于设备的入门和管理。它完全由托管集成管理，并且在所有托管集成中都是该特定设备所独有的。 AWS 区域当设备最初添加到托管集成时，系统会创建此标识符并将其附加到托管集成中的托管事物。托管事物是托管集成中物理设备的数字表示形式，用于镜像物理设备的所有设备元数据。对于第三方设备，除了`deviceId`存储在代表物理设备的托管集成中的标识符外，它们可能还有自己的、针对其第三方云的独立唯一标识符。

以下入职流程用于为中心配置托管集成：

将您的集线器加入托管集成：设置核心配置器和协议特定的插件，它们可以协同工作以处理设备身份验证、通信和设置。

提供了以下入门流程，用于为集线器连接的设备配置托管集成：

- 简单设置 (SS)：最终用户打开物联网设备的电源，并使用设备制造商的应用程序扫描其二维码。然后，设备将注册到托管集成云并连接到物联网中心。
- 零触摸设置 (ZTS)：该设备已预先关联到供应链的上游。例如，最终用户无需扫描设备二维码，而是提前完成此步骤，以便将设备预关联到客户账户。
- 用户指导设置 (UGS)：最终用户开启设备并按照交互式步骤将其加入托管集成。这可能包括按下 IoT 中心上的按钮、使用设备制造商的应用程序或同时按下集线器和设备上的按钮。如果简单设置失败，则可以使用此方法。

> **Note**
>
> 托管集成中的设备配置工作流程与设备的入门要求无关。无论设备类型或设备协议如何，托管集成都提供了简化的用户界面，便于用户登录和管理设备。

设备和设备配置文件生命周期

管理设备的生命周期和设备配置文件可确保您的设备群安全且高效运行。

主题

设备

在最初的入职过程中，系统会创建名为 “托管事物” 的物理设备的数字表示形式。Managed Th managedThingID ing 具有一个全球唯一标识符，用于在所有区域的托管集成中识别设备。设备在配置期间与本地集线器配对，以便与托管集成进行实时通信，或者为第三方设备配置第三方云。设备还与所有者相关联，该所有者由托管事物的公开owner APIs 参数标识，例如GetManagedThing。根据设备类型，设备会链接到相应的设备配置文件。

> **Note**
>
> 如果在不同的客户下多次配置物理设备，则该设备可能有多条记录。

设备生命周期从使用 API 在托管集成中创建托管事物开始，到客户使用 CreateManagedThing API 删除托管事物时结束。DeleteManagedThing设备的生命周期由以下公众管理 APIs：

- CreateManagedThing
- ListManagedThings
- GetManagedThing
- UpdateManagedThing
- DeleteManagedThing

设备配置文件

设备配置文件代表一种特定类型的设备，例如灯泡或门铃。它与制造商相关联，包含设备的功能。设备配置文件存储了托管集成的设备连接设置请求所需的身份验证材料。使用的身份验证材料是设备条形码。

在设备制造过程中，制造商可以使用托管集成注册其设备配置文件。这使制造商能够在入门和配置工作流程中从托管集成中获取设备所需的材料。设备配置文件中的元数据存储在物理设备上或打印在设备标签上。当制造商在托管集成中删除设备配置文件时，设备配置文件的生命周期即告结束。

数据模型

数据模型表示系统中数据的组织层次结构。此外，它还支持在整个设备实现之间进行 end-to-end通信。对于托管集成，使用了两种数据模型。托管集成数据模型和 AWS“物质数据模型” 的实现。它们都有相似之处，但也有细微的差异，将在以下主题中概述。

对于第三方设备，这两种数据模型都用于最终用户、托管集成和第三方云提供商之间的通信。要转换来自两个数据模型的设备命令和设备事件等消息，可以利用 Conn Cloud-to-Cloud ector 功能

主题

托管集成数据模型

托管集成数据模型管理最终用户与托管集成之间的所有通信。

设备层次结构

endpoint和capability数据元素用于描述托管集成数据模型中的设备。

endpoint

endpoint表示该功能提供的逻辑接口或服务。

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

Capability

capability代表设备功能。

```
{
    "$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[]
}
```

对于capability数据元素，有三个项目构成该项目：propertyaction、和event。它们可用于与设备交互和监控。

- 属性：设备保持的状态，例如可调光灯的当前亮度等级属性。
  - ```
    {
        "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

    }
    ```
- 操作：可以执行的任务，例如在门锁上锁门。操作可能会产生响应和结果。
  - ```
    {
        "name": { "$ref": "/schema-versions/definition/aws.name@1.0" }, //required
        "parameters": Map<String name, JSONNode value>,
        "responseCode": HTTPResponseCode,
        "errors": {
            "code": "string",
            "message": "string"
        }
    }
    ```
- 事件：本质上是过去状态转换的记录。虽然事件property代表当前的状态，但却是过去的日记，包括单调递增的计数器、时间戳和优先级。它们支持捕获状态转换，以及无法轻易实现的数据建模property。
  - ```
    {
        "name": { "$ref": "/schema-versions/definition/aws.name@1.0" },        //
    required
        "parameters": Map<String name, JSONNode value>
    }
    ```
    ```

AWS 案件数据模型的实现

AWS Matter 数据模型的实施管理托管集成与第三方云提供商之间的所有通信。

有关更多信息，请参阅 M atter 数据模型：开发者资源。

设备层次结构

有两个数据元素用于描述设备：endpoint、和cluster。

endpoint

endpoint表示该功能提供的逻辑接口或服务。

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

cluster

cluster代表设备功能。

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

对于cluster数据元素，有三个项目构成该项目：attributecommand、和event。它们可用于与设备交互和监控。

- 属性：设备保持的状态，例如可调光灯的当前亮度等级属性。
  - ```
    {
        "id" (hexadecimalString): (JsonNode) value
    }
    ```
- 命令：可以执行的任务，例如在门锁上锁门。命令可能会生成响应和结果。
  - ```
    "id": {
    ```

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

- 事件：本质上是过去状态转换的记录。虽然事件attributes代表当前的状态，但却是过去的日记，包括单调递增的计数器、时间戳和优先级。它们支持捕获状态转换，以及无法轻易实现的数据建模attributes。
  - ```
    "id": {
        "fieldId": "fieldValue",
        ...
    }
    ```
    ```

管理 IoT 设备命令和事件

设备命令提供了远程管理物理设备的功能，除了执行关键的安全、软件和硬件更新外，还可确保对设备的完全控制。对于庞大的设备群，知道设备何时执行命令可以对整个设备实现进行监督。设备命令或自动更新将触发设备状态更改，这反过来又会创建新的设备事件。此设备事件将触发自动发送到客户管理的目的地的通知。

主题

设备命令

命令请求是向设备发送的命令。命令请求包含一个有效负载，用于指定要执行的操作，例如打开灯泡。要发送设备命令，托管集成代表最终用户调用 SendManagedThingCommand API，然后将命令请求发送到设备。

有关 SendManagedThingCommand API 操作的更多信息，请参阅SendManagedThingCommand。

`UpdateState` 操作

要更新设备的状态，例如灯光亮起的时间，请在调用 SendManagedThingCommand API 时使用UpdateState操作。提供要更新的数据模型属性和新值parameters。以下示例说明了将灯泡更新OnTime为 SendManagedThingCommand API 请求5。

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

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

ReadState 操作

要获取设备的最新状态，包括所有数据模型属性的当前值，请在调用 SendManagedThingCommand API 时使用ReadState操作。在中propertiesToRead，您可以使用以下选项：

- 提供特定的数据模型属性以获取最新值，例如OnOff确定灯是开还是关。
- 使用通配符运算符 (*) 读取某项功能的所有设备状态属性。

以下示例说明了使用ReadState操作进行 SendManagedThingCommand API 请求的两种场景：

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

设备事件

设备事件包括设备的当前状态。这可能意味着设备已更改状态，或者即使状态未更改，也正在报告其状态。它包括在数据模型中定义的属性报告和事件。事件可能是洗衣机循环已完成，或者恒温器已达到最终用户设定的目标温度。

设备事件通知

最终用户可以订阅他们为更新特定设备事件而创建的特定客户管理的目的地。要创建客户管理的目的地，请调用 CreateDestination API。当设备向托管集成报告设备事件时，如果存在设备事件，则会通知客户管理的目的地。

托管集成通知

托管集成通知管理向客户发送的所有通知，便于在他们的设备上提供更新和见解的实时沟通。无论是通知客户设备事件、设备生命周期还是设备状态，托管集成通知在增强整体客户体验方面都起着至关重要的作用。通过提供可操作的信息，客户可以做出明智的决策并优化资源利用率。

设置托管集成通知

要设置托管集成通知，请按照以下步骤操作：

1. 创建 Amazon Kinesis 数据流

   要创建 Kinesis 数据流，请按照创建和管理 Kinesis 数据流中概述的步骤进行操作。

   目前，仅支持 Amazon Kinesis 数据流作为客户管理的托管集成通知目的地。
2. 创建 Amazon Kinesis 直播访问角色

   创建 AWS Identity and Access Management 有权访问您刚刚创建的 Kinesis 直播的访问角色

   有关更多信息，请参阅《AWS Identity and Access Management用户指南》中的 IAM 角色创建。
3. 向用户授予调用 **`CreateDestination`** API 的权限

   以下策略定义了用户调用 CreateDestinationAPI 的要求。如果未设置，则对 `CreateDestination` API 的调用将失败。

   要获取托管集成的passrole权限，请参阅AWS Identity and Access Management用户指南中的授予用户将角色传递给 AWS 服务的权限。

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

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

4. 调用 **CreateDestination** API

   创建 Amazon Kinesis 数据流和流访问角色后，调用 CreateDestinationAPI 创建您的客户管理的目标，托管集成通知将发送到该目的地。对于deliveryDestinationArn参数，请使用您的新 Amazon Kinesis 数据流中的。arn

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

5. 调用 **CreateNotificationConfiguration** API

   最后，您将创建通知配置，通过将通知路由到由您的 Amazon Kinesis 数据流表示的客户管理的目的地，通知您所选的事件类型。调用 CreateNotificationConfigurationAPI 来创建通知配置。在destinationName参数中，使用与最初使用 CreateDestination API 创建客户管理的目的地时创建的目标名称相同的目标名称。

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

使用托管集成监控的事件类型

以下是使用托管集成通知监控的事件类型：

- DEVICE_COMMAND
  - SendManagedThingAPI 命令的状态。有效值为 succeeded 或 failed。

```
{
      "version":"0",
      "messageId":"6a7e8feb-b491-4cf7-a9f1-bf3703467718",
      "messageType":"DEVICE_EVENT",
      "source":"aws.iotmanagedintegrations",
      "customerAccountId":"123456789012",
      "timestamp":"2017-12-22T18:43:48Z",
      "region":"ca-central-1",
      "resources":[
        "arn:aws:iotmanagedintegrations:ca-central-1:123456789012: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
  - 来自网络实时通信 (WebRTC) 的命令请求。

    WebRTC标准允许两个对等方之间进行通信。这些对等体可以传输实时视频、音频和任意数据。托管集成支持WebRTC，以便在客户的移动应用程序和最终用户的设备之间实现这些类型的流式传输。有关 WebRTC 标准的更多信息，请参阅 WebRTC。

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

```
      "payload":{
        "endpoints":[{
          "endpointId":"1",
          "capabilities":[{
            "id":"aws.DoorLock",
            "name":"Door Lock",
            "version":"1.0"
          }]
        }]
      }
}
```

- DEVICE_DISCOVERY_STATUS
  - 设备的发现状态。

```
{
      "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
  - 设备事件发生的通知。

```
{
      "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
  - 设备生命周期的状态。

```
{
      "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
  - 设备的 OTA 通知。
- DEVICE_STATE
  - 设备状态更新时的通知。

```
{
      "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) 连接器

cloud-to-cloud连接器允许您创建和促进第三方设备与之间的双向通信。 AWS

主题

什么是 Cloud-to-Cloud (C2C) 连接器？

cloud-to-cloud连接器是一种预先构建的软件包，可将安全地链接 AWS 云 到第三方云提供商的端点。使用 C2C 连接器，解决方案提供商可以利用 AWS IoT Device Management 的托管集成来控制连接到第三方云的设备。

托管集成包括连接器目录， AWS 客户可以在其中查看和选择要集成的连接器。有关更多信息，请参阅什么是 C2C 连接器目录？。

托管集成要求将每个连接器作为一个 AWS Lambda 功能来实现。

什么是 C2C 连接器目录？

AWS IoT Device Management 的托管集成连接器目录是一系列 C2C 连接器，可促进 AWS IoT Device Management 的托管集成与第三方云提供商之间的双向通信。您可以在 AWS Management Console 或中查看连接器 AWS CLI。

使用控制台查看托管集成连接器目录

1. 打开托管集成控制台
2. 在左侧导航窗格中，选择托管集成

3. 在托管集成控制台的左侧导航窗格中，选择目录。

AWS Lambda 用作 C2C 连接器

每个 C2C 连接器 Lambda 函数在托管集成和第三方平台上的相应操作之间转换和传输命令和事件。有关 Lambda 的更多信息，请参阅什么是。 AWS Lambda

例如，假设最终用户拥有由第三方 OEM 制造的智能灯泡。借助 C2C 连接器，最终用户可以通过托管集成平台发出开启或关闭此灯的命令。然后，该命令将被转发到连接器中托管的 Lambda 函数，该函数会将请求转换为针对第三方平台的 API 调用，以打开或关闭设备。

当您调用 API 时，需要使用 Lambda 函数。`CreateCloudConnector`部署到 Lambda 函数中的代码必须实现中提到的所有接口和功能。构建 C2C（云到云）连接器

托管集成连接器工作流程

开发人员必须使用托管集成注册 C2C 连接器。 AWS IoT Device Management此注册过程会创建逻辑连接器资源，客户可以访问该资源以使用该连接器。

> **Note**
>
> C2C 连接器是在托管集成中为 AWS IoT Device Management 创建的一组元数据，用于描述连接器。

下图描述了 C2C 连接器在将命令从移动应用程序发送到连接云的设备时所扮演的角色。C2C 连接器充当 AWS IoT Device Management 托管集成和第三方云平台之间的转换层。

使用 C2C（云到云）连接器的指南

您创建的任何 C2C 连接器都是您的内容，而您访问的由其他客户创建的任何 C2C 连接器都是第三方内容。 AWS 不创建或管理任何 C2C 连接器作为托管集成的一部分。

您可以与其他托管集成客户共享您的 C2C 连接器。如果您这样做，则您授权 AWS 作为您的服务提供商在 AWS 控制台上列出这些 C2C 连接器和相关联系信息，并且您知道其他 AWS 客户可能会与您联系。您全权负责授予客户访问您的 C2C 连接器的权限，以及管理其他 AWS 客户访问您的 C2C 连接器的任何条款。

构建 C2C（云到云）连接器

以下各节介绍为 AWS IoT Device Management 的托管集成构建 C2C（云到云）连接器的步骤。

主题

先决条件

在创建 C2C（云到云）连接器之前，您需要满足以下条件：

- AWS 账户 用于托管您的 C2C 连接器并通过托管集成进行注册。有关更多信息，请参阅创建 AWS 账户。
- 确保连接器所针对的第三方云提供商支持 OAuth 2.0 授权。有关更多信息，请参阅 OAuth 2.0 账户关联要求。

  此外，要测试连接器，连接器的开发人员必须具备以下条件：

  - 来自第三方云的客户端 ID，用于与 C2C 连接器关联
  - 来自第三方云的客户端密钥，用于与您的 C2C 连接器关联
  - OAuth 2.0 授权网址
  - OAuth 2.0 代币网址
- 您的第三方 API 所需的任何 API 密钥

C2C 连接器要求

您开发的 C2C 连接器促进了 AWS IoT Device Management 的托管集成与第三方供应商云之间的双向通信。连接器必须实现托管集成的接口，AWS IoT Device Management 才能代表最终用户执行操作。

这些接口提供了发现最终用户设备、启动从 AWS IoT Device Management 托管集成发送的设备命令以及基于访问令牌识别用户的功能。为了支持设备操作，连接器必须管理 AWS IoT Device Management 托管集成与相关第三方平台之间的请求和响应消息的转换。

以下是 C2C 连接器的要求：

- 第三方授权服务器必须符合 OAuth2 .0 标准以及中OAuth 配置要求列出的配置。
- 需要使用 C2C 连接器来解释物质数据模型 AWS 实现中的标识符，并且必须发出符合物质数据模型 AWS 实现的响应和事件。有关更多信息，请参阅 AWS 案件数据模型的实现。
- C2C 连接器必须能够通过身份验证调用 AWS IoT Device Manag APIs ement 的托管集成。SigV4对于 SendConnectorEvent 通过 API 发送的异步事件，必须使用用于注册连接器的相同 AWS 账户 凭据来签署相关 SendConnectorEvent 请求。
- 连接器必须实现AWS.ActivateUser、AWS.DiscoverDevicesAWS.SendCommand、和AWS.DeactivateUser操作。
- 当您的 C2C 连接器收到与设备命令响应或设备发现相关的第三方事件时，它必须将其转发到与 API 的托管集成。SendConnectorEvent有关这些事件和 SendConnectorEvent API 的更多信息，请参阅SendConnectorEvent。

> **Note**
>
> SendConnectorEventAPI 是托管集成 SDK 的一部分，用于代替手动构建和签署请求。

OAuth 2.0 账户关联要求

每个 C2C 连接器都依赖 OAuth 2.0 授权服务器对最终用户进行身份验证。通过此服务器，最终用户将其第三方账户与客户的设备平台关联起来。账户关联是最终用户使用 C2C 连接器支持的设备所需的第一步。有关账户关联和 OAuth 2.0 中不同角色的更多信息，请参阅账户关联角色。

虽然您的 C2C 连接器不需要实现特定的业务逻辑来支持授权流程，但与 C2C 连接器关联的 OAuth2 .0 授权服务器必须满足。OAuth 配置要求

> **Note**
>
> AWS IoT Device Management 仅适用于的托管集成支持带有授权代码流的 OAuth 2.0。有关更多信息，请参阅 RFC 6749。

账户关联是一个允许托管集成和连接器使用访问令牌访问最终用户设备的过程。此令牌为最终用户许可的 AWS IoT Device Management 提供托管集成，以便连接器可以通过 API 调用与最终用户的数据进行交互。有关更多信息，请参阅 账户关联工作流程。

的托管集成 AWS IoT Device Management 不会直接获得访问令牌；而是通过授权码授权类型获得访问令牌。首先，AWS IoT Device Management 的托管集成必须获得授权码。然后，它将代码交换为访问令牌和刷新令牌。刷新令牌用于在旧访问令牌过期时请求新的访问令牌。如果访问令牌和刷新令牌都已过期，则必须再次执行账户关联流程。你可以通过 `StartAccountAssociationRefresh` API 操作来做到这一点。

> **Important**
>
> 发放的访问令牌的作用域必须按用户划分，但不能按 OAuth 客户机划分。该令牌不应提供对客户端下所有用户的所有设备的访问权限。
> 授权服务器必须执行以下操作之一：
>
> - 发行包含可提取的最终用户（资源所有者）ID 的访问令牌，例如 JWT-Token。
> - 返回每个已颁发的访问令牌的最终用户 ID。

OAuth 配置要求

下表说明了 OAuth 授权服务器中用于托管集成 AWS IoT Device Management 以执行账户关联所需的参数：

OAuth 服务器参数

字段	必填	评论
`clientId`	是	您的应用程序的公共标识符。它用于启动身份验证流程，并且可以公开共享。
`clientSecret`	是	用于通过授权服务器对应用程序进行身份验证的密钥，尤其是在使用授权码交换访问令牌时。应将其保密，不得公开共享。

authorizationType	是	此授权配置支持的授权类型。当前，“OAuth 2.0” 是唯一支持的值。
authUrl	是	第三方云提供商的授权 URL。
tokenUrl	是	第三方云提供商的令牌 URL。
tokenEndpointAuthenticationScheme	是	“HTTP_BASIC” 或 “REQUEST_BODY_CREDENTIALS” 的身份验证方案。HTTP_BASIC 表示客户端凭证包含在授权标头中，而阶梯表示它们包含在请求正文中。

必须对您使用的 OAuth 服务器进行配置，使访问令牌字符串值必须使用 UTF-8 字符集进行 Base64 编码。

账户关联角色

要创建 C2C 连接器，你需要一个 OAuth 2.0 授权服务器和账户关联。有关更多信息，请参阅 账户关联工作流程。

OAuth 2.0 在实现账户关联时定义了以下四个角色：

1. 授权服务器
2. 资源所有者（最终用户）
3. 资源服务器
4. 客户端

以下内容定义了这些 OAuth 角色中的每一个：

授权服务器

授权服务器是识别和验证第三方云中最终用户身份的服务器。此服务器提供的访问令牌可以将 AWS 最终用户的客户平台帐户与其第三方平台帐户关联起来。此过程称为账户关联。

授权服务器通过提供以下内容来支持账户关联：

- 显示登录页面，供最终用户登录您的系统。这通常被称为授权端点。
- 对系统中的最终用户进行身份验证。
- 生成识别最终用户的授权码。
- 将授权码传递给 AWS IoT Device Management 的托管集成。
- 接受 AWS IoT Device Management 托管集成的授权码，并返回访问令牌，AWS IoT Device Management 的托管集成可用于访问系统中最终用户的数据。这通常通过单独的 URI（称为令牌 URI 或端点）完成。

> **Important**
>
> 授权服务器必须支持 OAuth 2.0 授权码流程，才能与 AWS IoT Device Management Connector 的托管集成一起使用。AWS IoT Device Management 的托管集成还支持带有代码交换证明密钥 (PKCE) 的授权代码流。
> 授权服务器必须：
>
> - 发布包含可提取的最终用户或资源所有者 ID 的访问令牌，例如 jwt-Tokens
> - 能够返回每个已发放的访问令牌的最终用户 ID
>
> 否则，您的连接器将无法支持所需的`AWS.ActivateUser`操作。这将防止在托管集成中使用连接器。

如果连接器开发者或所有者没有维护自己的授权服务器，则使用的授权服务器必须为连接器开发者第三方平台管理的资源提供授权。这意味着托管集成从授权服务器接收的任何令牌都必须在设备（资源）上提供有意义的安全边界。例如，最终用户令牌不允许在另一台最终用户设备上执行命令；该令牌提供的权限会映射到平台内的资源。以 L ights 公司为例。当最终用户启动与其连接器的账户关联流程时，他们将被重定向到授权服务器正面的 L ights In corporated 登录页面。一旦他们登录并向客户端授予权限，他们就会提供一个令牌，允许连接器访问其 L ights In corporated 账户中的资源。

资源所有者（最终用户）

作为资源所有者，您可以通过执行账户关联来允许 AWS IoT Device Management 客户通过托管集成访问与您的账户关联的资源。例如，以最终用户已登录 Lights Incorporated 移动应用程序的智能灯泡为例。资源所有者是指购买并登录设备的最终用户账户。在我们的示例中，资源所有者被建模为 L ights In OAuth2 corporated .0 账户。作为资源所有者，此账户提供发出命令和管理设备的权限。

资源服务器

这是托管需要授权才能访问的受保护资源（设备数据）的服务器。 AWS 客户需要代表最终用户访问受保护的资源，他们可以通过账户关联后的 AWS IoT Device Management 连接器的托管集成来访问受保护的资源。以之前的智能灯泡为例，资源服务器是L ights In corporated拥有的一项基于云的服务，用于在灯泡上线后对其进行管理。通过资源服务器，资源所有者可以向智能灯泡发出命令，例如将其打开和关闭。受保护的资源仅向最终用户的账户以及 accounts/entities 他们可能已提供权限的其他账户提供权限。

客户端

在这种情况下，客户端就是您的 C2C 连接器。客户机被定义为代表最终用户授予对资源服务器内资源的访问权限的应用程序。账户关联过程代表连接器（客户端），请求访问第三方云中最终用户的资源。

尽管连接器是 OAuth 客户端，但 AWS IoT Device Management 的托管集成代表连接器执行操作。例如，AWS IoT Device Management 的托管集成向授权服务器发出获取访问令牌的请求。连接器仍被视为客户端，因为它是访问资源服务器中受保护资源（设备数据）的唯一组件。

以最终用户安装的智能灯泡为例。在客户平台和L ights In corporated授权服务器之间完成账户关联后，连接器本身将与资源服务器通信，以检索有关最终用户智能灯泡的信息。然后，连接器可以接收来自最终用户的命令。这包括通过 Lights Incorporated 资源服务器代表他们打开或关闭灯。因此，我们将连接器指定为客户端。

账户关联工作流程

对于客户通过 C2C 连接器与 AWS IoT Device Management 平台上的终端用户设备进行交互的托管集成，它会通过以下工作流程获取访问令牌：

1. 当用户通过客户应用程序启动第三方设备的启动时，AWS IoT Device Management 的托管集成会返回授权 URI 以及. AssociationId
2. 应用程序前端存储 AssociationId 并将最终用户重定向到第三方平台的登录页面。
   - 最终用户登录。最终用户授予客户端访问其设备数据的权限。
3. 第三方平台创建授权码。最终用户将被重定向到 AWS IoT Device Management 平台回调 URI 的托管集成，包括重定向请求所附的代码。
4. 托管集成会将此代码与第三方平台令牌 URI 交换。
5. 令牌 URI 验证授权码并返回与最终用户关联的 OAuth2 .0 访问令牌和刷新令牌。

6. 托管集成调用 C2C 连接器并进行`AWS.ActivateUser`操作，以完成账户关联流程并获取。UserId
7. 托管集成 OAuthRedirectUrl （从连接器策略配置）将成功的身份验证页面返回到客户应用程序。

> **Note**
>
> 如果出现故障，AWS IoT Device Management 的托管集成会将错误和 error_ description 查询参数附加到向客户应用程序提供错误详细信息的网址。

8. 客户应用程序将最终用户重定向到。 OAuth RedirectUrl此时，应用程序前端从第一步就知道 AssociationId 了关联。

   通过 C2C 连接器向 AWS IoT Device Management 托管集成向第三方云平台发出的所有后续请求，例如发现设备和发送命令的命令，都将包含 OAuth2 .0 访问令牌。

下图显示了账户关联的关键组成部分之间的关系：

实现 C2C 连接器接口操作

的托管集成 AWS IoT Device Management 定义了您 AWS Lambda 必须处理的四个操作才有资格成为连接器。您的 C2C 连接器必须实现以下每项操作：

1. `AWS.ActivateUser`- AWS IoT Device Management 服务的托管集成调用此 API 来检索与提供的 OAuth2 .0 令牌关联的全球唯一用户标识符。可以选择使用此操作来执行账户关联过程的任何其他要求。
2. `AWS.DiscoverDevices`-AWS IoT Device Management 服务的托管集成调用此 API 到您的连接器以发现用户的设备
3. `AWS.SendCommand`-AWS IoT Device Management 服务的托管集成会将此 API 调用到您的连接器，以便为用户设备发送命令
4. `AWS.DeactivateUser`-AWS IoT Device Management 服务的托管集成将此 API 调用您的连接器，以停用用户的访问令牌，以便在您的授权服务器中取消链接。

的托管集成 AWS IoT Device Management 始终通过操作调用带有 JSON 字符串负载的 Lambda 函数。 AWS Lambda `invokeFunction`请求操作必须在每个请求负载中包含一个`operationName`字段。有关更多信息，请参阅 AWS Lambda API 参考中的调用。

目前，每次调用超时设置为 15 秒，如果调用失败，则将重试五次。

您为连接器实现的 Lambda 将operationName从请求有效负载中解析一个，并实现相应的功能以映射到第三方云：

```
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**
>
> 连接器的开发者必须实现前面示例中列出的activateUserManager.activateUser(request)deviceDiscoveryManager.listDevice和deactivateUser.deactivateUser操作。

以下示例详细介绍了来自托管集成的通用连接器请求，其中包含每个必需接口的公共字段。从示例中，您可以看到既有请求标头，又有请求负载。请求标头在每个操作接口中都很常见。

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

```
  },
  "payload":{
   "operationName": "AWS.SendCommand",
   "operationVersion": "1.0",
   "connectorId": “exampleId”,
  …
  }
}
```

默认请求标头

默认标题字段如下所示。

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

连接器托管的任何 API 都必须处理以下标头参数：

默认标题和字段

字段	必填/可选	描述
header:auth	是	C2C 连接器生成器在连接器注册期间提供的授权信息。
header:auth:token	是	由第三方云提供商生成并链接到的用户的授权令牌connectorAssociationID 。
header:auth:type	是	所需的授权类型。

> **Note**
>
> 对您的连接器的所有请求都将附加最终用户的访问令牌。您可以假设最终用户与托管集成客户之间已经存在账户关联。

请求负载

除了常用标头外，每个请求都将有一个有效负载。虽然此有效载荷对每种操作类型都有唯一的字段，但每个有效载荷都有一组将始终存在的默认字段。

请求有效载荷字段：

- `operationName`：给定请求的操作，等于以下值之一：`AWS.ActivateUser`、`AWS.SendCommand`、`AWS.DiscoverDevices`、`AWS.DeactivateUser`。
- `operationVersion`：每个操作都有版本控制，以允许其随着时间的推移而演变，并为第三方连接器提供稳定的接口定义。托管集成在所有请求的有效载荷中传递一个版本字段。
- `connectorId`：已向其发送请求的连接器的 ID。

默认响应标头

每项ACK操作都将通过 AWS IoT Device Management 的托管集成进行响应，确认您的 C2C 连接器已收到请求并开始处理请求。以下是上述回复的通用示例：

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

每个操作响应都必须具有以下通用标头：

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

```
    }
}
```

下表列出了默认的响应标头：

默认响应标头和字段

字段	必填/可选	评论
header:responseCode	是	表示请求执行状态的值的枚举。

在本文档中描述的各种连接器接口和 API 架构中，都有一个responseMessage或Message字段。这是一个可选字段，用于 C2C 连接器 Lambda 来响应有关请求及其执行的任何上下文。最好是，任何导致状态码之外的错误都200应包含描述错误的消息值。

使用 API 响应 C2C 连接器操作请求 SendConnectorEvent

的托管集成要求 AWS IoT Device Management 您的连接器在每个 and 操作中都以异步方式运行AWS.SendCommand。AWS.DiscoverDevices这意味着对这些操作的初始响应只是 “确认” 您的 C2C 连接器已收到请求。

使用 SendConnectorEvent API，您的连接器应将以下列表中的事件类型发送到 for and operations，以及主动设备事件（例如手动开启和关闭灯光）。AWS.DiscoverDevices AWS.SendCommand要阅读有关这些事件类型及其用例的详细说明，请参阅实施 AWS。DiscoverDevices 操作实施 AWS。 SendCommand 操作、和使用 SendConnectorEvent API 发送设备事件。

例如，如果您的 C2C 连接器收到DiscoverDevices请求，则 AWS IoT Device Management 的托管集成希望它与上面定义的响应格式同步响应。然后，您必须使用中实施 AWS。 DiscoverDevices 操作定义的请求结构为 DEVICE_DISCOVERY 事件调用 SendConnectorEvent API。API 调用可以在任何您有权访问 C2C 连接器 AWS 账户 Lambda 凭证的地方进行。SendConnectorEvent在 AWS IoT Device Management 的托管集成收到此事件之前，设备发现流程才会成功。

> **Note**
>
> 或者，如有必要，SendConnectorEventAPI 调用可以在 C2C 连接器 Lambda 调用响应之前进行。但是，这种流程与软件开发的异步模型相矛盾。

- SendConnectorEvent-您的连接器调用 AWS IoT Device Management API 的托管集成，将设备事件发送到 AWS IoT Device Management 的托管集成。托管集成仅接受三种类型的事件：
  - “DEV ICE_DISCOVERY” — 此事件操作应用于发送第三方云中发现的设备列表以获取特定的访问令牌。
  - “DEVICE_COMMAND_RESPONSE” — 此事件操作应用于发送作为命令执行结果的特定设备事件。
  - “DEVICE_EV ENT”-此事件操作应用于源自设备且不是基于用户的命令的直接结果的任何事件。这可以作为一种常规事件类型，用于主动报告设备状态变化或通知。

实施 AWS。 ActivateUser 操作

AWS IoT Device Management 的托管集成需要执行该`AWS.ActivateUser`操作，才能从最终用户的 OAuth2 .0 令牌中检索用户标识符。的托管集成 AWS IoT Device Management 将在请求标头中传递 OAuth令牌，并期望您的连接器在响应负载中包含全局唯一的用户标识符。此操作发生在账户关联流程成功之后。

以下列表概述了连接器的要求，以促进成功的`AWS.Activate`用户流。

- 您的 C2C 连接器 Lambda 可以处理来自 AWS `AWS.ActivateUser` IoT Device Management 托管集成的操作请求消息。
- 您的 C2C 连接器 Lambda 可以根据 OAuth2提供的.0 令牌确定唯一的用户标识符。通常，如果是 JWT 令牌，则可以从令牌本身中提取它，也可以通过令牌从授权服务器请求它。

`AWS.ActivateUser` 工作流

1. 的托管集成使用以下有效负 AWS IoT Device Management 载调用您的 C2C 连接器 Lambda：

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

```
}
}
```

2. C2C 连接器通过令牌或通过查询您的第三方资源服务器来确定要包含在AWS.ActivateUser响应中的用户 ID。
3. C2C 连接器会响应 Lambda AWS.ActivateUser 操作调用，包括默认负载以及字段内相应的用户标识符。userId

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

实施 AWS。 DiscoverDevices 操作

设备发现会将最终用户拥有的物理设备列表与 AWS IoT Device Management 托管集成中维护的最终用户设备的数字表示形式保持一致。只有在用户与 AWS IoT Device Management 的托管集成之间完成账户关联后， AWS 客户才会在最终用户拥有的设备上执行此操作。设备发现是一个异步过程，其中 AWS IoT Device Management 的托管集成会调用连接器来启动设备发现请求。C2C 连接器异步返回已发现的最终用户设备列表，其中包含托管集成生成的参考标识符（称为deviceDiscoveryId）。

下图说明了 AWS IoT Device Management 的最终用户和托管集成之间的设备发现工作流程：

AWS。 DiscoverDevices 工作流程

1. 客户代表最终用户启动设备发现流程。
2. 的托管集成 AWS IoT Device Management 会生成一个参考标识符，该标识符调deviceDiscoveryId用 AWS 客户生成的设备发现请求。
3. 的托管集成使用AWS.DiscoverDevices操作界面向 C2C 连接器 AWS IoT Device Management 发送设备发现请求，包括有效 OAuthaccessToken的最终用户以及. deviceDiscoveryId

4. 您的连接器存储deviceDiscoveryId以包含在DEVICE_DISCOVERY活动中。此事件还将包含已发现的最终用户设备的列表，并且必须将其发送到 AWS IoT Device Management 的托管集成，并将 SendConnectorEvent API 作为DEVICE_DISCOVERY事件发送。
5. 您的 C2C 连接器应调用资源服务器来获取最终用户拥有的所有设备。
6. 您的 C2C 连接器 Lambda 会响应 Lambda 调用 invokeFunction ()，并向 AWS IoT Device Management 的托管集成发出 ACK 响应，作为操作的初始响应。AWS.DiscoverDevices托管集成通过 ACK 通知客户已启动的设备发现流程。
7. 您的资源服务器会向您发送最终用户拥有和操作的设备列表。
8. 您的连接器将每台最终用户设备转换为 AWS IoT Device Management 所需设备格式的托管集成 ConnectorDeviceIdConnectorDeviceName，包括每台设备的能力报告。
9. C2C 连接器还提供UserId已发现设备所有者的信息。它可以作为设备列表的一部分从您的资源服务器中检索，也可以单独调用，具体取决于您的资源服务器实现。
10. 接下来，您的 C2C 连接器将使用 AWS 账户 凭证并将操作参数设置为 “DEVICE_DISCOVERY”SendConnectorEvent，通过 sigv4 调用 AWS IoT 设备管理 API 的托管集成。发送到 AWS IoT Device Management 托管集成的设备列表中的每台设备都将由设备特定的参数表示connectorDeviceId，例如connectorDeviceName、和 a。capabilityReport

    - 根据您的资源服务器响应，您需要相应地通知 AWS IoT Device Management 的托管集成。

      例如，如果您的资源服务器对最终用户发现的设备列表进行了分页响应，那么对于每次轮询，您都可以发送一个statusCode参数为的3xx单个DEVICE_DISCOVERY操作事件。如果您的设备发现仍在进行中，请重复步骤 5、6 和 7。
11. 的托管集成会向客户 AWS IoT Device Management 发送有关已发现最终用户设备的通知。
12. 如果您的 C2C 连接器发送的DEVICE_DISCOVERY操作事件statusCode参数更新为 200，则托管集成将通知客户设备发现工作流程已完成。

> **Important**
>
> 如果需要，步骤 7 到 11 可以在步骤 6 之前进行。例如，如果您的第三方平台有一个用于列出最终用户设备的 API，则可以在 C2C 连接器 Lambda 使用典型的 ACK 响应SendConnectorEvent之前发送 DEVICE_DISCOVERY 事件。

设备发现的 C2C 连接器要求

以下列表概述了 C2C 连接器的要求，以便于成功发现设备。

- C2C 连接器 Lambda a 可以处理来自 AWS IoT Device Management 托管集成的设备发现请求消息并处理操作。`AWS.DiscoverDevices`
- 您的 C2C 连接器可以使用用于注册连接器的凭证 APIs 通过 Sigv4 调用 AWS IoT Device Management 的 AWS 账户 托管集成。

设备发现过程

以下步骤概述了使用您的 C2C 连接器和托管集成 AWS IoT Device Management 的设备发现过程。

设备发现过程

1. 托管集成会触发设备发现：
   - 使用以下 JSON 负载`DiscoverDevices`向发送 POST 请求：

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

2. 连接器确认发现：
   - 连接器发送带有以下 JSON 响应的确认：

```
{
     "header": {
          "responseCode":200
```

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

3. 连接器发送设备发现事件：

   - 使用以下 JSON 负载/connector-event/*{your_connector_id}*向发送 POST 请求：

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

```
}
```

为 DISCOVER_D CapabilityReport EVICES 事件构造一个

如上面定义的事件结构所示，在 DISCOVER_DEVICES 事件中报告的每台设备作为对`AWS.DiscoverDevices`操作的响应，都需要 CapbilityReport 来描述相应设备的功能。`CapabilityReport`以符合 Matter 的格式表示 AWS IoT Device Management 设备功能的托管集成。` CapabilityReport`中必须提供以下字段：

- `nodeId`，字符串：包含以下内容的设备节点的标识符 `endpoints`
- `version`，String：此设备节点的版本，由连接器开发者设置
- `endpoints`，<Cluster>列表：此设备端点支持的案件数据模型 AWS 实现列表。
  - `id`，字符串：连接器开发者设置的端点标识符
  - `deviceTypes`，<String>列表：此端点捕获的设备类型列表，即 “摄像头”。
  - `clusters`，Lis <Cluster>t：此端点支持的案件数据模型的 AWS 实现列表。
    - `id`，字符串：Matter 标准定义的集群标识符。
    - `revision`，整数：Matter 标准定义的聚类修订号。
    - `attributes`，<String, Object> 地图：属性标识符及其对应的当前设备状态值的映射，标识符和有效值由问题标准定义。
      - `id`，字符串：案件数据模型的 AWS 实现所定义的属性 ID。
      - `value`，对象：由属性 ID 定义的属性的当前值。“值” 的类型可以根据属性而变化。该`value`字段对于每个属性都是可选的，只有当您的连接器 lambda 可以在发现期间确定当前状态时，才应包含该字段。
    - `commands`，<String>列表：按照 Matter 标准的定义，此集群 IDs 支持的命令列表。
    - `events`，<String>列表：根据 Matter 标准的定义，此集群 IDs 支持的事件列表。

有关物质数据模型支持的功能及其相应AWS 实现的当前列表，请参阅最新版本的数据模型文档。

实施 AWS。 SendCommand 操作

该`AWS.SendCommand`操作允许 AWS IoT Device Management 的托管集成通过 AWS 客户将最终用户启动的命令发送到您的资源服务器。您的资源服务器可能支持多种类型的设备，其中每种类型都有自己的响应模型。命令执行是一个异步过程，其中 AWS IoT Device Management 的托管集成发送带有 “TraceID” 的命令执行请求，您的连接器将包含在通过 “API” SendConnectorEvent 发送回托管集成的

命令响应中。AWS IoT Device Management 的托管集成期望资源服务器返回一个确认已收到命令的响应，但不一定表示命令已执行。

下图以最终用户尝试打开房屋灯光的示例说明了命令执行流程：

设备命令执行工作流程

1. 最终用户使用 AWS 客户的应用程序发送开灯的命令。
2. 客户将命令信息与最终用户的设备信息传递给 AWS IoT Device Management 的托管集成。
3. 托管集成会生成 “TraceID”，您的连接器将在将命令响应发送回服务时使用该标识。
4. AWS IoT Device Management 的托管集成使用`AWS.SendCommand`操作界面向您的连接器发送命令请求。
    - 此接口定义的有效载荷包括设备标识符、以 Matter 形式制定的设备命令endpoints/clusters/commands、最终用户的访问令牌以及其他必需的参数。
5. 您的连接器存储`traceId`要包含在命令响应中的内容。
    - 您的连接器会将托管集成命令请求转换为资源服务器的相应格式。
6. 您的连接器`UserId`从提供的最终用户的访问令牌中获取，并将其与命令相关联。
    a. 要`UserId`么使用单独的调用从您的资源服务器中检索，要么从 JWT 和类似令牌中提取访问令牌。
    b. 实现取决于您的资源服务器和访问令牌的详细信息。
7. 您的连接器调用资源服务器以 “打开” 最终用户的灯。
8. 资源服务器与设备交互。
    a. 连接器中继到资源服务器已下发命令的 AWS IoT IoT Device Management 托管集成，并以 ACK 作为初始同步命令响应进行响应。
    b. 然后，托管集成将其中继回客户应用程序。
9. 设备开灯后，您的资源服务器会捕获该设备事件。
10. 您的资源服务器将设备事件发送到连接器。
11. 您的连接器将资源服务器生成的设备事件转换为托管集成 DEVICE_COMMAND_RESPONSE 事件操作类型。
12. 您的连接器调用 `SendConnectorEvent` API，操作为 “DEVICE_COMMAND_RESPONSE”。
    - 它会在初始请求中附上由 AWS IoT Device Management 托管集成`traceId`提供的内容。

13. 托管集成会通知客户有关最终用户的设备状态变化。
14. 客户通知最终用户设备灯已亮起。

> **Note**
>
> 您的资源服务器配置决定了处理失败的设备命令请求和响应消息的逻辑。这包括对命令使用相同的 referenceId 进行消息重试尝试。

执行设备命令的 C2C 连接器要求

以下列表概述了 C2C 连接器的要求，以促进设备命令的成功执行。

- C2C 连接器 Lambda 可以`AWS.SendCommand`处理来自 AWS IoT Device Management 托管集成的操作请求消息。
- 您的 C2C 连接器必须跟踪发送到您的资源服务器的命令，并将其映射到相应的 “TraceID”。
- 您可以使用用于注册 C2C 连接器的 AWS 凭证通过 Sigv4 调用 AWS IoT Device Management 服务 API 的 AWS 账户 托管集成。

1. 托管集成向连接器发送命令（请参阅上图中的步骤 4）。

   - ```
     /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 连接器 ACK 命令（请参阅上图中的步骤 7，其中连接器向 AWS IoT 设备管理服务的托管集成发送 ACK）。

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

3. 连接器发送设备命令响应事件（请参阅上图中的步骤 11）。

   ```
   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**
>
> 在通过 API 收到相应的 DEVICE_COMMAND_RESPONSE 事件之前，由于命令执行而导致的设备状态变化不会反映在 AWS IoT Device Management 的托管集成中。SendConnectorEvent 这意味着，在托管集成收到前面步骤 3 的事件之前，无论您的连接器调用响应是否表示成功，设备状态都不会更新。

解释 AWS 中包含的 “终端节点”。 SendCommand 请求

托管集成将使用设备发现期间报告的设备功能来确定设备可以接受哪些命令。每个设备功能都是通过 AWS 实现物质数据模型进行建模的；因此，所有传入的命令都将从给定集群中的 “commands” 字段派生。您的连接器负责解析 “端点” 字段，确定相应的 Matter 命令，然后对其进行翻译，以便正确的命令到达设备。通常，这意味着将 Matter 数据模型转换为相关的 API 请求。

执行命令后，您的连接器将确定由物质数据模型的 AWS 实现定义的哪些 “属性” 因此发生了变化。然后，这些更改将通过 API 发送的 API DEVICE_COMMAND_RESPONSE 事件报告给 AWS IoT Device Management 的托管集成。SendConnectorEvent

考虑以下示例负载中包含的 “端点” 字段：AWS.SendCommand

```
    "endpoints": [{
        "id": "1",
        "clusters": [{
            "id": "0x0202",
            "commands": [{
                "0xff01":
                    {
```

```
                        "0x0000": "3"
                    }
                }]
            }]
        }]
```

通过此对象，连接器可以确定以下内容：

1. 设置终端节点和集群信息：
   a. 将端点设置id为 “1”。

      > **Note**
      >
      > 如果设备定义了多个端点，例如单个集群（例如On/Off) can control multiple capabilities (i.e. turn a light on/off as well as turning a strobe on/off），则使用此 ID 将命令路由到正确的功能。

   b. 将集群设置id为 “0x0202”（风扇控制集群）。
2. 设置命令信息：
   a. 将命令标识符设置为 “0xff01”（更新状态命令由定义）。 AWS
   b. 使用请求中提供的值更新包含的属性标识符。
3. 更新属性：
   a. 将属性标识符设置为 “0x0000”（风扇控制集FanMode 群的属性）。
   b. 将属性值设置为 “3”（高风扇速度）。

托管集成定义了两种 “自定义” 命令类型，这些类型不是由物质数据模型的 AWS 实现严格定义的：ReadState 和 UpdateState 命令。要获取和设置 Matter 定义的集群属性，托管集成将向您的连接器发送一个AWS.SendCommand请求，其中包含与 UpdateState (id: 0xff01) 或 ReadState (id: 0xff02) IDs 相关的命令，以及必须更新或读取的相应属性参数。对于设置为可变（可更新）或可检索（可读取）的属性，可以为任何设备类型调用这些命令，这些属性可以从相应的 Matter 数据模型 AWS 实现中调用。

使用 SendConnectorEvent API 发送设备事件

设备启动的事件概述

虽然 SendConnectorEvent API 用于异步响应AWS.SendCommand和AWS.DiscoverDevices操作，但它也用于将任何设备启动的事件通知托管集成。设备启动的事件可以定义为设备在没有用户启动命令的情况下生成的任何事件。这些设备事件可能包括但不限于设备状态变化、运动检测、电池电量等。您可以使用带操作 DEV IC E_EVENT 的 SendConnectorEvent API 将这些事件发送回托管集成。

以下部分以安装在家中的智能摄像头为例，进一步说明这些事件的工作流程：

设备事件工作流程

1. 您的摄像机会检测到动作，从而生成一个发送到您的资源服务器的事件。
2. 您的资源服务器处理该事件并将其发送到您的 C2C 连接器。
3. 您的连接器会将此事件转换为 AWS IoT Device Manag DEVICE_EVENT ement 接口的托管集成。
4. 您的 C2C 连接器使用操作设置为 “DEVICE_EVENT” SendConnectorEvent 的 API 将此设备事件发送到托管集成。
5. 托管集成可识别相关客户，并将此事件转发给客户。
6. 客户收到此事件并通过用户标识符将其显示给用户。

有关 SendConnectorEvent API 操作的更多信息，请参阅 SendConnectorEvent AWS IoT Device Management 托管集成 API 参考指南。

设备启动的事件要求

以下是设备启动的事件的一些要求。

- 您的 C2C 连接器资源应该能够从您的资源服务器接收异步设备事件
- 您的 C2C 连接器资源应该能够使用用于注册 C2C 连接器的 AWS 凭证通过 Sigv4 调用 AWS IoT 设备管理服务 API 的 AWS 账户 托管集成。

以下示例演示了连接器通过 API 发送源于设备的事件： SendConnectorEvent

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

从以下示例中，我们可以看到以下内容：

- 这来自 ID 等于 1 的设备端点。
- 与该事件相关的设备功能的集群 ID 为 0x0202，与风扇控制问题集群有关。
- 已更改的属性的 ID 为 0x000，与集群中的风扇模式枚举有关。它已更新为值 3，与 High 的值有关。
- 由于`connectorId`是云服务在创建时返回的参数，因此 Connectors 必须使用查询 GetCloudConnector 和筛选依据`lambdaARN`。使用 `Lambda.get_function_url_config` API 查询 lambda 自己的ARN值。这`CloudConnectorId`允许在 lambda 中动态访问，而不是像之前那样进行静态配置。

实施 AWS。 DeactivateUser 操作

用户停用概述

当客户删除其客户账户，或者最终用户想要取消其在系统中的账户与 AWS 客户系统的关联时，需要停用提供的用户访问令牌。 AWS 在这两种用例中，托管集成都需要使用 C2C 连接器来简化此工作流程。

下图说明了如何取消最终用户帐户与系统的关联

用户停用工作流程

1. 用户启动 AWS 客户账户与与 C2C 连接器关联的第三方授权服务器之间的解除关联过程。
2. 客户通过 AWS IoT Device Management 的托管集成启动删除用户关联。
3. 托管集成通过使用操作界面向连接器发出请求来启动停用过程。`AWS.DeactivateUser`
   - /user 的访问令牌包含在请求的标头中。
4. 您的 C2C 连接器接受请求并调用您的授权服务器来撤消令牌及其提供的任何访问权限。
   - 例如，来自未关联用户账户的事件在执行后不应再发送到托管集成。`AWS.DeactivateUser`
5. 您的授权服务器撤消访问权限并将响应发送回您的 C2C 连接器。
6. 您的 C2C 连接器会向 AWS IoT Device Management 的托管集成发送用户访问令牌已被撤销的 ACK。
7. 托管集成会删除最终用户拥有的、与您的资源服务器关联的所有资源。
8. 托管集成会向客户发送 ACK，说明与您的系统相关的所有关联都已删除。
9. 客户通知最终用户其账户已与您的平台取消关联。

AWS。 DeactivateUser 要求

- C2C 连接器 Lambda 函数接收来自托管集成的请求消息以处理该操作。`AWS.DeactivateUser`
- C2C 连接器必须撤销授权服务器中用户提供的 OAuth2.0 令牌和相应的刷新令牌。

以下是您的连接器将收到的示例`AWS.DeactivateUser`请求：

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

         }
     },
     "payload":{
         "operationName": "AWS.DeactivateUser"
         "operationVersion": "1.0"
```

```
        "connectorId": "Your-connector-Id"
    }
}
```

调用你的 C2C 连接器

AWS Lambda 允许基于资源的策略授权谁可以调用 Lambda。由于 AWS IoT Device Management 的托管集成是 AWS 服务，因此您必须允许托管集成通过资源策略调用 C2C 连接器 Lambda。

将至少具有以下最低权限的资源策略附加到您的 C2C 连接器 Lambda。这提供了与 Lambda 函数调用权限的托管集成：

```
{
  "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:connector-region:your-aws-account-
id:function:connector-lambda-name"
    }
  ]
}
```

为您的 IAM 角色添加权限

所有托管集成都 APIs 需要 AWS Sigv4 身份验证才能调用。Sigv4 正在签署协议，使用您的 AWS 账户凭据对 AWS API 请求进行身份验证。您用于调用托管集成的 IAM 角色 APIs 必须具有以下权限才能成功调用： APIs

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

```
    ],
    "Resource": [
     "Your-Resource"
    ]
   }]
}
```

有关添加这些权限的更多信息，请联系 支持。

其他资源

要注册您的 C2C 连接器，您需要满足以下条件：

- 表示您要注册的连接器的 Lambda ARN。

手动测试您的 C2C 连接器

要手动测试 C2C 连接器 end-to-end，必须同时模拟客户和最终用户。

您将需要以下资源：

- 表示您要 AWS Lambda 测试的连接器的 ARN。
- 来自您的云平台的测试 OAuth 2.0 用户帐户。
- 在 AWS IoT Device Management 托管集成中注册的连接器。有关更多信息，请参阅 使用 C2C（云到云）连接器。

使用 C2C（云到云）连接器

C2C 连接器管理请求和响应消息的翻译，并支持托管集成与第三方供应商云之间的通信。它促进了对不同设备类型、平台和协议的统一控制，从而可以加载和管理第三方设备。

以下过程列出了使用 C2C 连接器的步骤。

使用 C2C 连接器的步骤：

1. CreateCloudConnector

   配置连接器以启用托管集成与第三方供应商云之间的双向通信。

   设置连接器时，请提供以下详细信息：

- 名称：为连接器选择一个描述性名称。
- 描述：简要概述连接器的用途和功能。
- AWS Lambda ARN：指定为连接器供电的 AWS Lambda 函数的亚马逊资源名称 (ARN)。

构建和部署与第三方供应商通信的 AWS Lambda 函数 APIs 以创建连接器。接下来，在托管集成中调用 CreateCloudConnectorAPI，并提供 AWS Lambda 函数 ARN 进行注册。确保该 AWS Lambda 函数部署在托管集成中创建连接器的 AWS 账户中。系统将为您分配一个唯一的连接器 ID 来识别集成。

CreateCloudConnector API 请求和响应示例：

```
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"
}
```

创建流程：

> **Note**
>
> 根据需要使用GetCloudConnectorUpdateCloudConnector、DeleteCloudConnector、和 ListCloudConnectors APIs ，执行此过程。

2. CreateConnectorDestination

配置目标以提供连接器与第三方供应商云建立安全连接所需的设置和身份验证凭据。使用 Destinations 将您的第三方身份验证凭据注册到托管集成，例如 OAuth 2.0 授权详细信息，包括授权 URL、身份验证方案以及其中凭据的位置 AWS Secrets Manager。

先决条件

在创建之前 ConnectorDestination，您必须：

- 调用 CreateCloudConnectorAPI 创建连接器。在 CreateConnectorDestinationAPI API 调用中使用该函数返回的 ID。
- 检索连接`tokenUrl`器的 3P 平台的。（你可以用 authCod e 兑换 AccessT oken）。
- 检索连接器的 3P 平台的 authURL。（最终用户可以使用其用户名和密码进行身份验证）。
- 在账户的密钥管理器中使用`clientId`和`clientSecret`（来自 3P 平台）。

CreateConnectorDestination API 请求和响应示例：

```
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,
                "timeBeforeRenewal": 30
            }
        }
    },
    "CloudConnectorId": "<connectorId>", // The connectorID instance from response
 of Step 1.
    "SecretsManager": {
        "arn": "arn:aws:secretsmanager:*****:secret:*******",
```

```
        "versionId": "********"
    },
    "ClientToken": "***"
}

Response:

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

云目标创建流程：

> **Note**
>
> 根据需要使用GetCloudConnectorUpdateCloudConnector、DeleteCloudConnector、和 ListCloudConnectors APIs ，执行此过程。

3. CreateAccountAssociation

关联表示最终用户的第三方云账户与连接器目标之间的关系。在创建关联并将最终用户与托管集成关联后，他们的设备可通过唯一的关联 ID 进行访问。这种集成支持三个关键功能：发现设备、发送命令和接收事件。

先决条件

在创建之前，AccountAssociation您必须完成以下操作：

- 调用 CreateConnectorDestinationAPI 创建目的地。该函数返回的 ID 将用于 CreateAccountAssociationAPI 调用。
- 调用 CreateAccountAssociation API。

CreateAccountAssociation API 请求和响应示例：

```
Request:

{
    "Name": "CreateAccountAssociation",
    "Description": "CreateAccountAssociation",
```

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

Response:

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

> **Note**
>
> 根据需要使用GetCloudConnectorUpdateCloudConnector、DeleteCloudConnector、和 ListCloudConnectors APIs ，执行此过程。

AccountAssociation的状态是从GetAccountAssociation和ListAccountAssociations APIs中查询的。这些 APIs 显示了协会的状况。StartAccountAssociationRefreshAPI 允许在刷新令牌到期时刷新AccountAssociation状态。

4. 设备发现

每个托管事物都与设备特定的详细信息相关联，例如其序列号和数据模型。数据模型描述了设备的功能，表明它是灯泡、开关、恒温器还是其他类型的设备。要发现 3P 设备并为该 3P 设备创建 ManagedThing，您必须按顺序执行以下步骤。

a. 调用 StartDeviceDiscoveryAPI 开始设备发现过程。

StartDeviceDiscovery API 请求和响应示例：

```
Request:

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

Response:

{
```

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

b. 调用 GetDeviceDiscoveryAPI 来检查发现过程的状态。

c. 调用 ListDiscoveredDevicesAPI 列出发现的设备。

ListDiscoveredDevices API 请求和响应示例：

```
Request:

//Empty body

Response:

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

d. 调用 CreateManagedThingAPI 从发现列表中选择要导入到托管集成的设备。

CreateManagedThing API 请求和响应示例：

```
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. 调用 GetManagedThingAPI 来查看这个新创建的managedThing。状态将是UNASSOCIATED。

f. 调用 RegisterAccountAssociationAPI managedThing 将其与特定关联accountAssociation。成功的 RegisterAccountAssociationAPI 结束后，ASSOCIATED状态会managedThing发生变化。

RegisterAccountAssociation API 请求和响应示例：

```
Request:

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

Response:

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

5. 向 3P 设备发送命令

要控制新上线的设备，请使用 SendManagedThingCommandAPI，使用先前创建的关联 ID 和基于设备支持的功能的控制操作。连接器使用账户关联过程中存储的凭据向第三方云进行身份验证并调用操作的相关 API 调用。

SendManagedThingCommand API 请求和响应示例：

```
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"
}
```

向 3P 设备流程发送命令：

6. 连接器向托管集成发送事件

   SendConnectorEventAPI 捕获从连接器到托管集成的四种类型的事件，由 O peration Type 参数的以下枚举值表示：

   - D@@ EVICE_COMMAND_RESPONSE：连接器为响应命令而发送的异步响应。

- DEV@@ ICE_DISCO VERY：为了响应设备发现过程，连接器使用 API 将发现的设备列表发送给托管集成。SendConnectorEvent
- 设备事件：发送接收到的设备事件。
- D@@ EVICE_COMMAND_REQUEST：从设备发起的命令请求。例如，WebRTC工作流程。

连接器还可以使用带有可选`userId`参数的 SendConnectorEventAPI 转发设备事件。

- 对于带有以下内容的设备事件`userId`：

  SendConnectorEvent API 请求和响应示例：

```
Request:

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

Response:

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

- 对于没有`userId`：的设备事件

  SendConnectorEvent API 请求和响应示例：

```
Request:
```

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

Response:

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

要删除特定账户关联managedThing和账户关联之间的关联，请使用注销机制：

DeregisterAccountAssociation API 请求和响应示例：

```
Request:

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

Response:

HTTP/1.1 200 // Empty body
```

发送事件流：

7. 将连接器状态更新为 “已上市”，使其对其他托管集成客户可见

默认情况下，连接器是私有的，只有创建连接器的 AWS 账户才能看见。您可以选择让其他托管集成客户看到连接器。

要与其他用户共享您的连接器，请使用连接器详细信息页面 AWS Management Console 上的 “设为可见” 选项，将您的连接器 ID 提交给以 AWS 供审核。一旦获得批准，连接器便可供所有托管集成用户使用。 AWS 区域此外，您可以 IDs 通过修改连接器关联 AWS Lambda 功能的访问策略来限制对特定 AWS 账户的访问权限。为确保您的连接器可供其他客户使用，请管理您的 Lambda 函数从其他 AWS 账户到可见连接器的 IAM 访问权限。

在将连接器设置为其他托管集成客户可见之前，请查看管理连接器共享和访问权限的 AWS 服务条款和组织政策。

托管集成 Hub SDK

使用本节中的主题来学习如何使用托管集成 Hub SDK 加载和控制 IoT 中心设备。有关托管集成终端设备 SDK 的更多信息，请参阅。托管集成终端设备 SDK

中心 SDK 架构

设备上线

在开始使用托管集成之前，请先查看 Hub SDK 组件如何支持设备上线。本节介绍设备入门所需的基本架构组件，包括核心配置器和协议专用插件如何协同工作以处理设备身份验证、通信和设置。

用于设备加载的 Hub SDK 组件

SDK 组件

- 核心供应器
- 特定于协议的供应器插件
- 特定于协议的中间件

核心供应器

核心配置器是在 IoT 中心部署中协调设备启动的核心组件。它协调托管集成与您的协议特定配置器插件之间的所有通信，确保设备启动安全可靠。当您加载设备时，核心配置器会通过以下功能处理身份验证流程、管理 MQTT 消息并处理设备请求：

MQTT 连接

与 MQTT 代理建立连接，以便发布和订阅云主题。

消息队列和处理程序

按顺序处理传入的添加和删除设备请求。

协议插件接口

通过管理身份验证和无线电加入模式，与协议特定的配置器插件配合使用，用于设备入门。

中心 SDK 客户端 APIs

接收来自特定协议的 CDMB 插件的设备功能报告，并将其转发到托管集成。

特定于协议的供应器插件

特定于协议的配置器插件是用于管理不同通信协议的设备加载的库。每个插件都会将来自核心配置程序的命令转换为物联网设备的特定于协议的操作。这些插件执行以下操作：

- 特定于协议的中间件初始化
- 基于核心供应器请求的无线电加入模式配置
- 通过中间件 API 调用移除设备

特定于协议的中间件

特定于协议的中间件充当设备协议和托管集成之间的转换层。该组件处理双向通信——接收来自供应器插件的命令并将其发送到协议堆栈，同时还收集来自设备的响应并通过系统将其路由回去。

设备上线流程

查看使用 Hub SDK 加载设备时发生的操作顺序。本节显示了组件在入门过程中的交互方式，并概述了支持的入门方法。

入职流程

- 简单设置 (SS)
- 零触摸设置 (ZTS)
- 用户指导设置 (UGS)

简单设置 (SS)

最终用户打开物联网设备的电源，并使用设备制造商的应用程序扫描其二维码。然后，设备将注册到托管集成云并连接到物联网中心。

零触摸设置 (ZTS)

零触摸设置 (ZTS) 通过在供应链上游预关联设备来简化设备上游。例如，最终用户无需扫描设备二维码，而是提前完成此步骤，以便将设备预先关联到客户账户。例如，可以在运营中心完成此步骤。

当最终用户收到设备并开机时，它会自动注册到托管集成云中并连接到物联网中心，无需任何其他设置操作。

用户指导设置 (UGS)

最终用户开启设备并按照交互式步骤将其加入托管集成。这可能包括按下 IoT 中心上的按钮、使用设备制造商的应用程序或同时按下集线器和设备上的按钮。如果简单设置失败，则可以使用此方法。

设备控制

托管集成可处理设备注册、命令执行和控制。您可以使用与供应商和协议无关的设备管理功能，在不了解设备特定协议的情况下打造最终用户体验。

通过设备控制，您可以查看和修改设备状态，例如灯泡亮度或门位置。该功能会针对状态变化发出事件，您可以将其用于分析、规则和监控。

主要特征

修改或读取设备状态

根据设备类型查看和更改设备属性。您可以访问：

- 设备状态：当前设备属性值
- 连接状态：设备可接通性状态
- He@@ alth status：系统值，例如电池电量和信号强度 (RSSI)

状态变更通知

当设备属性或连接状态发生变化时接收事件，例如灯泡亮度调整或门锁状态变化。

离线模式

即使没有互联网连接，设备也能与同一物联网中心上的其他设备通信。恢复连接后，设备状态会与云同步。

状态同步

跟踪来自多个来源、设备制造商应用程序和手动设备调整的状态变化。

查看通过托管集成控制设备所需的 Hub SDK 组件和流程。本主题介绍 Edge Agent、Common Data Model Bridge (CDMB) 和特定于协议的插件如何协同工作，以处理设备命令、管理设备状态和处理不同协议的响应。

设备控制流程

下图通过描述最终用户如何打开 ZigBee 智能插头来演示 end-to-end设备控制流程。

用于设备控制的 Hub SDK 组件

Hub SDK 架构使用以下组件来处理和路由物联网实现中的设备控制命令。在将云命令转换为设备操作、管理设备状态和处理响应方面，每个组件都起着特定的作用。以下各节详细介绍了这些组件在您的部署中如何协同工作：

Hub SDK 由以下组件组成，便于在物联网中心上启动和控制设备。

主要组件：

边缘代理

充当物联网中心和托管集成之间的网关。

通用数据模型桥 (CDMB)

在 AWS 数据模型和本地协议数据模型（如 Z-Wave 和 Zigbee）之间进行转换。它包括一个核心 CDMB 和特定于协议的 CDMB 插件。

置备者

处理设备发现和上线。它包括用于特定于协议的入门任务的核心配置器和特定于协议的配置器插件。

次要组件

Hub 新手入门

为集线器配置客户端证书和密钥，以实现安全的云通信。

MQTT 代理

提供与托管集成云的 MQTT 连接。

日志记录程序

将日志写入本地或托管集成云。

安装并验证托管集成 Hub SDK

在以下部署方法之间进行选择，在您的设备上安装托管集成 Hub SDK，AWS IoT Greengrass 用于自动部署或手动安装脚本。本节介绍两种方法的设置和验证步骤。

部署方法

- 使用以下命令安装 Hub SDK AWS IoT Greengrass
- 使用脚本部署 Hub SDK
- 使用系统部署 Hub SDK

使用以下命令安装 Hub SDK AWS IoT Greengrass

使用 AWS IoT Greengrass （ Java 版本 ）为您的设备部署托管集成 Hub SDK 组件。

> **Note**
>
> 您必须已经设置并已了解 AWS IoT Greengrass。有关更多信息，请参阅AWS IoT Greengrass 开发者指南文档 AWS IoT Greengrass中的内容。

AWS IoT Greengrass 用户必须具有修改以下目录的权限：

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

主题

- 在本地部署组件
- 云部署
- 验证集线器配置
- 验证 CDMB 的运行情况

- 验证 LPW 配置器的运行情况

在本地部署组件

使用设备上CreateDeployment AWS IoT Greengrass 的 API 部署 Hub SDK 组件。版本号不是静态的，可能因您当时使用的版本而异。使用以下格式表示**version**：com.amazon.io。TManaged IntegrationsDevice AceCommon= `0.2.0`。

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

云部署

按照AWS IoT Greengrass 开发者指南中的说明执行以下步骤：

1. 将项目上传到亚马逊 S3。
2. 更新配方以包含 Amazon S3 工件的位置。
3. 在设备上为新组件创建云部署。

验证集线器配置

通过检查您的配置文件来确认配置成功。打开`/data/aws/iotmi/config/iotmi_config.json`文件并验证状态是否设置为`PROVISIONED`。

验证 CDMB 的运行情况

检查日志文件中是否有 CDMB 启动消息和成功初始化。*`logs file`*位置可能因安装位置 AWS IoT Greengrass 而异。

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

示例

```
[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
```

验证 LPW 配置器的运行情况

检查日志文件中是否有 LPW-Provisioner 启动消息和成功初始化。*logs file*位置可能因安装位置 AWS IoT Greengrass 而异。

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

示例

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

使用脚本部署 Hub SDK

使用安装脚本手动部署托管集成 Hub SDK 组件，然后验证部署。本节介绍脚本执行步骤和验证过程。

主题

准备好您的环境

在运行 SDK 安装脚本之前，请完成以下步骤：

1. 在文件夹`middleware`内创建一个名为`artifacts`的文件夹。
2. 将您的集线器中间件文件复制到该文件`middleware`夹。
3. 在启动 SDK 之前运行初始化命令。

> **Important**
>
> 每次集线器重新启动后重复初始化命令。

```
#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
```

运行 Hub SDK 脚本

导航到构件目录并运行`start_iotmi_sdk.sh`脚本。此脚本按正确的顺序启动 Hub SDK 组件。查看以下示例日志以验证是否成功启动：

> **Note**
>
> 所有正在运行的组件的日志都可以在该`artifacts/logs`文件夹中找到。

```
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----
```

验证集线器配置

检查中的iot_provisioning_state字段/data/aws/iotmi/config/iotmi_config.json是否设置为PROVISIONED。

验证代理操作

检查日志文件中是否有代理启动消息和成功初始化。

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

示例

```
[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**
>
> 检查您的`artifacts`目录中是否存在该`iotmi.db`数据库。

验证 LPW 配置器的运行情况

检查日志文件中是否有`LPW-Provisioner`启动消息和成功初始化。

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

下面的代码显示了一个示例。

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

使用系统部署 Hub SDK

> **Important**
>
> 请按`readme.md`照 release.tgz 文件`hubSystemdSetup`目录中的内容获取最新更新。

本节介绍在基于 Linux 的中心设备上部署和配置服务的脚本和过程。

概览

部署过程由两个主要脚本组成：

- `copy_to_hub.sh`: 在主机上运行以将必要文件复制到集线器
- `setup_hub.sh`: 在集线器上运行以配置环境和部署服务

此外，还systemd/deploy_iotshd_services_on_hub.sh处理进程引导顺序和进程权限管理，并由setup_hub.sh自动触发。

先决条件

成功部署需要满足列出的先决条件。

- 集线器上有 systemd 服务可用
- 通过 SSH 访问中心设备
- 集线器设备上的 Sudo 权限
- scp安装在主机上的实用程序
- sed安装在主机上的实用程序
- 主机上安装了 unzip 实用程序

文件结构

文件结构旨在便于组织和管理其各个组成部分，从而实现内容的高效访问和导航。

```
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
```

在 SDK 发行版 tgz 文件中，总体文件结构为：

```
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
```

初始 设置

解压软件开发工具包包

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

导航到解压缩的目录并准备软件包：

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

添加设备配置文件

按照列出的两个步骤创建设备配置文件并将其复制到集线器。

1. 添加设备配置文件以创建所需的设备配置文件。SDK 使用此文件来实现其功能。
2. 复制配置文件以将创建的配置文件复制到集线器。

将文件复制到集线器

从您的主机运行部署脚本：

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

Example 示例

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

这复制了：

- 软件包文件（在集线器上重命名为 package.zip）
- 配置文件
- 证书
- 系统服务文件

设置集线器

复制文件后，通过 SSH 连接到集线器并运行安装脚本：

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

用户和群组配置

默认情况下，我们将用户中心和群组中心用于 SDK 组件。有多种方法可以对其进行配置：

- 使用自定义用户/群组：

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

- 在运行安装脚本之前手动创建它们：

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

- 在中添加命令`setup_hub.sh`。

管理 服务

要重新启动所有服务，请从 hub 运行以下脚本：

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

安装脚本将创建必要的目录、设置适当的权限并自动部署服务。如果您不使用 SSH/SCP，则必须`copy_to_hub.sh`针对您的特定部署方法进行修改。在部署之前，请确保所有证书文件和配置均已正确设置。

将您的集线器加入托管集成

通过配置所需的目录结构、证书和设备配置文件，将您的中心设备设置为与托管集成进行通信。本节介绍集线器载入子系统组件如何协同工作、证书和配置文件的存储位置、如何创建和修改设备配置文件以及完成集线器配置过程的步骤。

Hub 入门子系统

集线器载入子系统使用以下核心组件来管理设备配置和配置：

Hub 入门组件

通过协调中心状态、配置方法和身份验证材料来管理中心入职流程。

设备配置文件

在设备上存储重要的集线器配置数据，包括：

- 设备配置状态（已配置或未配置）
- 证书和密钥位置
- 身份验证信息其他 SDK 进程（例如 MQTT 代理）将引用此文件来确定集线器状态和连接设置。

证书处理程序接口

提供用于读取和写入设备证书和密钥的实用程序接口。你可以实现这个接口来使用：

- 文件系统存储
- 硬件安全模块 (HSM)
- 可信平台模块 (TPM)
- 定制安全存储解决方案

MQTT 代理组件

使用以下方法管理 device-to-cloud通信：

- 预配置的客户端证书和密钥
- 配置文件中的设备状态信息
- 与托管集成的 MQTT 连接

下图描述了集线器载入子系统架构及其组件。如果您不使用 AWS IoT Greengrass，则可以忽略图中的该组件。

Hub 入职设置

在开始队列配置入门流程之前，请完成每台中心设备的这些设置步骤。本节介绍如何创建托管事物、设置目录结构和配置所需的证书。

设置步骤

- 步骤 1：注册自定义终端节点
- 步骤 2：创建配置文件
- 步骤 3：创建托管事物（队列配置）
- 步骤 4：创建目录结构
- 第 5 步：向中心设备添加身份验证材料
- 步骤 6：创建设备配置文件
- 第 7 步：将配置文件复制到集线器

步骤 1：注册自定义终端节点

创建专用的通信端点，您的设备使用该端点与托管集成交换数据。此端点为所有 device-to-cloud消息（包括设备命令、状态更新和通知）建立安全的连接点。

要注册终端节点

- 使用 RegisterCustomEndpointAPI 创建用于 device-to-managed集成通信的端点。

  RegisterCustomEndpoint请求示例

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

  响应：

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

  > **Note**
  >
  > 存储端点地址。你将需要它来进行 future 设备通信。

要返回端点信息，请使用 `GetCustomEndpoint` API。

有关更多信息，请参阅《RegisterCustomEndpoint托管集成 GetCustomEndpointAPI 参考指南》中的 API 和 API。

步骤 2：创建配置文件

配置文件包含您的设备连接到托管集成所需的安全凭证和配置设置。

创建队列配置文件

- 调用 CreateProvisioningProfileAPI 生成以下内容：
  - 用于定义设备连接设置的配置模板
  - 用于设备身份验证的索赔证书和私钥

> **Important**
>
> 安全地存储索赔证书、私钥和模板 ID。您需要这些凭据才能将设备加入托管集成。如果您丢失了这些凭据，则必须创建新的配置文件。

`CreateProvisioningProfile`请求示例

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

响应：

```
{
"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"
}
```

步骤 3：创建托管事物（队列配置）

使用 CreateManagedThing API 为您的中心设备创建托管事物。每个中心都需要自己的托管东西和独特的身份验证材料。有关更多信息，请参阅《托管集成 CreateManagedThingAPI 参考》中的 API。

创建托管事物时，请指定以下参数：

- Role：将此值设置为CONTROLLER。
- AuthenticationMaterial：包括以下字段。
  - SN: 此设备的唯一序列号
  - UPC: 此设备的通用产品代码
- Owner: 此托管事物的所有者标识符。

> **Important**
>
> 每台设备的身份验证材料中必须有一个唯一的序列号 (SN)。

CreateManagedThing请求示例：

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

有关更多信息，请参阅托管集成 API 参考CreateManagedThing中的。

（可选）获取托管事物

在ProvisioningStatus继续操作UNCLAIMED之前，必须先完成托管事务。使用 GetManagedThing API 验证您的托管事物是否存在且已准备好进行配置。有关更多信息，请参阅托管集成 API 参考GetManagedThing中的。

步骤 4：创建目录结构

为您的配置文件和证书创建目录。默认情况下，Hub 入职流程使用。/data/aws/iotmi/config/iotmi_config.json

您可以在配置文件中为证书和私钥指定自定义路径。本指南使用默认路径/data/aws/iotmi/certs。

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

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

第 5 步：向中心设备添加身份验证材料

将证书和密钥复制到您的中心设备，然后创建设备特定的配置文件。在配置过程中，这些文件可在您的集线器和托管集成之间建立安全的通信。

复制索赔证书和密钥

- 将这些身份验证文件从 CreateProvisioningProfile API 响应复制到您的中心设备：
  - claim_cert.pem: 索赔证书（所有设备通用）
  - claim_pk.key: 索赔证书的私钥

  将两个文件都放在/data/aws/iotmi/certs目录中。

> **Important**
>
> 以 PEM 格式存储证书和私钥时，请通过正确处理换行符来确保格式正确。对于 PEM 编码的文件，(\n)必须将换行符替换为实际的行分隔符，因为仅存储转义的换行符以后将无法正确检索。

> **Note**
>
> 如果您使用安全存储，请将这些凭据存储在您的安全存储位置而不是文件系统中。有关更多信息，请参阅 创建用于安全存储的自定义证书处理程序。

步骤 6：创建设备配置文件

创建包含唯一设备标识符、证书位置和配置设置的配置文件。SDK 在集线器启动期间使用此文件对您的设备进行身份验证、管理配置状态和存储连接设置。

> **Note**
>
> 每台集线器设备都需要自己的配置文件，其中包含唯一的设备特定值。

使用以下过程创建或修改您的配置文件，然后将其复制到集线器。

- 创建或修改配置文件（队列配置）。

  在设备配置文件中配置以下必填字段：

  - 证书路径

    1. `iot_claim_cert_path`: 您的索赔证明的位置 (`claim_cert.pem`)
    2. `iot_claim_pk_path`: 您的私钥的位置 (`claim_pk.key`)
    3. 在实现安全存储证书处理程序时，这两个字段都使用 `SECURE_STORAGE`

  - 连接设置

    1. `fp_template_name`: 之前的`ProvisioningProfile`名字。

2. endpoint_url：来自 RegisterCustomEndpoint API 响应的托管集成终端节点 URL（一个区域内的所有设备都相同）。

- 设备标识符

1. SN: 与您的 CreateManagedThing API 调用匹配的设备序列号（每台设备都是唯一的）
2. UPC来自您的 CreateManagedThing API 调用的通用产品代码（此产品的所有设备都相同）

```
{
    "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"
    }
}
```

配置文件的内容

查看iotmi_config.json文件内容。

内容

键	值	由客户添加？	备注
iot_provisioning_method	FLEET_PROVISIONING	是	指定要使用的配置方法。
iot_claim_cert_path	您指定的文件路径或 SECURE_STORAGE 。例如，/data/aws/	是	指定要使用的文件路径或SECURE_STORAGE 。

键	值	由客户添加？	备注
	iotmi/certs/claim_cert.pem		
iot_claim_pk_path	您指定的文件路径或 SECURE_STORAGE 。例如，/data/aws/iotmi/certs/claim_pk.pem	是	指定要使用的文件路径或SECURE_STORAGE 。
fp_template_name	队列配置模板名称应等于之前使用的名称。ProvisioningProfile	是	等同于之前使用的名称 ProvisioningProfile
endpoint_url	托管集成的终端节点 URL。	是	您的设备使用此 URL 连接到托管集成云。要获取此信息，请使用 RegisterCustomEndpointAPI。
SN	设备序列号。例如 AIDACKCEVSQ6C2EXAMPLE 。	是	您必须为每台设备提供此唯一信息。
UPC	设备通用产品代码。例如 841667145075 。	是	您必须为设备提供此信息。
managed_thing_id	托管事物的 ID。	否	此信息稍后在集线器配置后由入职流程添加。
iot_provisioning_state	供应状态。	是	必须将置备状态设置为NOT_PROVISIONED 。
iot_permanent_cert_path	物联网证书路径。例如 /data/aws/iotmi/iot_cert.pem 。	否	此信息稍后在集线器配置后由入职流程添加。

键	值	由客户添加？	备注
iot_permanent_pk_path	物联网私钥文件路径。例如 /data/aws/iotmi/iot_pk.pem 。	否	此信息稍后在集线器配置后由入职流程添加。
client_id	将用于 MQTT 连接的客户端 ID。	否	此信息稍后由集线器配置后的入门流程添加，以供其他组件使用。
event_manager_upper_bound	默认值为 500	否	此信息稍后由集线器配置后的入门流程添加，以供其他组件使用。

第 7 步：将配置文件复制到集线器

将您的配置文件复制到/data/aws/iotmi/config或您的自定义目录路径。您将在入门过程中提供此HubOnboarding二进制文件路径。

用于舰队配置

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

加载设备并在集线器中对其进行操作

通过创建托管设备并将其连接到集线器，将设备设置为加载到托管集成中心。可以通过简单的设置或用户指导的设置将设备加载到集线器。

主题

- 使用简单的设置来加载和操作设备

- 使用用户指导设置来载入和操作设备

使用简单的设置来加载和操作设备

通过创建托管设备并将其连接到集线器，将设备设置为加载到托管集成中心。本节介绍使用简单设置完成设备上线流程的步骤。

先决条件

在尝试加载设备之前，请完成以下步骤：

- 将集线器设备载入托管集成中心。
- 安装《托管集成 AWS CLI 命令 AWS CLI参考》中的最新版本的
- 订阅 DEVICE_L IFE_CYCLE 事件通知。

设置步骤

- 步骤 1：创建凭证柜
- 第 2 步：将凭证柜添加到您的中心
- 步骤 3：使用凭据创建托管事物。
- 第 4 步：插入设备并检查其状态。
- 步骤 5：获取设备功能
- 步骤 6：向托管事物发送命令
- 第 7 步：从集线器中移除托管内容

步骤 1：创建凭证柜

为您的设备创建一个凭证柜。

创建凭证柜

- 使用 `create-credential-locker` 命令。执行此命令将触发所有制造资源的创建，包括 Wi-Fi 设置 key pair 和设备证书。

  create-credential-locker 示例

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

响应：

```
{
  "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"
}
```

有关更多信息，请参阅《托管集成create-credential-locker命令参考》中的 AWS CLI 命令。

第 2 步：将凭证柜添加到您的中心

将凭证柜添加到您的中心。

向您的中心添加凭证柜

- 使用以下命令将凭证储物柜添加到您的集线器。

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

步骤 3：使用凭据创建托管事物。

使用设备凭据创建托管事物。每台设备都需要自己的托管设备。

创建托管事物

- 使用create-managed-thing命令为您的设备创建托管事物。

  create-managed-thing 示例

```
#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-Wave 和 Zigbee 设备有不同的命令。

响应：

```
{
  "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"
}
```

有关更多信息，请参阅《托管集成create-managed-thing命令参考》中的 AWS CLI 命令。

第 4 步：插入设备并检查其状态。

.

插入设备并检查其状态。

- 使用get-managed-thing命令检查设备的状态。

  get-managed-thing 示例

```
#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"
```

响应：

```
{
  "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"
}
```

有关更多信息，请参阅《托管集成get-managed-thing命令参考》中的 AWS CLI 命令。

步骤 5：获取设备功能

使用get-managed-thing-capabilities命令获取您的终端节点 ID 并查看设备可能执行的操作列表。

获取设备的功能

- 使用get-managed-thing-capabilities命令并记下端点 ID。

  get-managed-thing-capabilties 示例

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

响应：

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

有关更多信息，请参阅《托管集成get-managed-thing-capabilities命令参考》中的 AWS CLI 命令。

步骤 6：向托管事物发送命令

使用该send-managed-thing-command命令向您的托管事物发送切换操作命令。

向你的托管事物发送命令

- 使用send-managed-thing-command命令向您的托管事物发送命令。

send-managed-thing-command 示例

```
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**
>
> 这个例子使用了 jq cli，但你也可以传递整个字符串 endpointId

响应：

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

有关更多信息，请参阅《托管集成send-managed-thing-command命令参考》中的 AWS CLI 命令。

第 7 步：从集线器中移除托管内容

通过移除托管的东西来清理集线器。

删除托管事物

- 使用delete-managed-thing命令从设备中心移除托管内容。

  delete-managed-thing 示例

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

  有关更多信息，请参阅《托管集成delete-managed-thing命令参考》中的 AWS CLI 命令。

  > **Note**
  >
  > 如果设备处于某种DELETE_IN_PROGRESS状态，请将该--force标志附加到。delete-managed-thing command

  > **Note**
  >
  > 对于 Z-Wave 设备，您需要在执行命令后将设备置于配对模式。

使用用户指导设置来载入和操作设备

通过创建托管设备并将其连接到集线器，将设备设置为加载到托管集成中心。本节介绍使用用户指导设置完成设备上线流程的步骤。

先决条件

在尝试加载设备之前，请完成以下步骤：

- 将集线器设备载入托管集成中心。
- 安装《托管集成 AWS CLI 命令 AWS CLI参考》中的最新版本的
- 订阅设备发现状态事件通知。

用户指导的设置步骤

步骤 1：开始设备发现

为您的集线器启动设备发现，以获取可用于加载设备的发现任务 ID。

开始设备发现

- 使用start-device-discovery命令获取发现任务 ID。

  start-device-discovery 示例

  ```
  #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
  ```

  响应：

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

> **Note**
>
> Z-Wave 和 Zigbee 设备有不同的命令。

有关更多信息，请参阅《托管集成 AWS CLI 命令参考》中的 start-device-discoveryAPI。

步骤 2：查询发现任务 ID

使用`list-discovered-devices`命令获取设备的身份验证材料。

查询您的发现任务 ID

- 使用发现任务 ID 和 list-discovered-devices命令获取设备的身份验证材料。

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

响应：

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

第 3 步：为您的设备创建托管内容

使用`create-managed-thing`命令为您的设备创建托管事物。每台设备都需要自己的托管设备。

创建托管事物

- 使用`create-managed-thing`命令为您的设备创建托管事物。

  create-managed-thing 示例

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

```
--authentication-material "AUTHENTICATION_MATERIAL"
```

响应：

```
{
  "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"
}
```

有关更多信息，请参阅《托管集成create-managed-thing命令参考》中的 AWS CLI 命令。

步骤 4：查询托管事物

您可以使用get-managed-thing命令检查托管事物是否已激活。

查询托管事物

- 使用get-managed-thing命令检查托管事物的配置状态是否设置为ACTIVATED。

  get-managed-thing 示例

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

  响应：

```
{
    "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": {}
}
```

有关更多信息，请参阅《托管集成get-managed-thing命令参考》中的 AWS CLI 命令。

第 5 步：获取托管事物功能

您可以使用查看托管事物的可用操作列表get-managed-thing-capabiltiies。

获取设备的功能

- 使用get-managed-thing-capabilities命令获取端点 ID。另请注意可能的操作列表。

  get-managed-thing-capabilities 示例

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

  响应：

```
{
    "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"
                        ],
                        ...
}
```

有关更多信息，请参阅《托管集成get-managed-thing-capabilities命令参考》中的 AWS CLI命令。

步骤 6：向托管事物发送命令

您可以使用该 send-managed-thing-command命令向您的托管事物发送切换操作命令。

使用切换操作向托管事物发送命令。

- 使用send-managed-thing-command命令发送切换操作命令。

  send-managed-thing-command 示例

```
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**
>
> 这个例子使用了 jq cli，但你也可以传递整个字符串 endpointId

响应：

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

有关更多信息，请参阅《托管集成send-managed-thing-command命令参考》中的 AWS CLI 命令。

第 7 步：检查托管事物的状态

检查托管事物的状态以验证切换操作是否成功。

检查托管事物的设备状态

- 使用get-managed-thing-state命令验证切换操作是否成功。

  get-managed-thing-state 示例

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

响应：

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

有关更多信息，请参阅《托管集成get-managed-thing-state命令参考》中的 AWS CLI命令。

第 8 步：从集线器中移除托管内容

通过移除托管的东西来清理集线器。

删除托管事物

- 使用delete-managed-thing命令移除托管事物。

  delete-managed-thing 示例

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

  有关更多信息，请参阅《托管集成delete-managed-thing命令参考》中的 AWS CLI 命令。

  > **Note**
  >
  > 如果设备停滞在DELETE_IN_PROGRESS状态，delete-managed-thing请在命令后面附加--force标志。

> **Note**
>
> 对于 Z-Wave 设备，您需要在执行命令后将设备置于配对模式。

创建用于安全存储的自定义证书处理程序

在加入托管集成中心时，设备证书管理至关重要。虽然默认情况下证书存储在文件系统中，但您可以创建自定义证书处理程序以增强安全性和灵活的凭据管理。

托管集成 End device SDK 为安全存储接口提供了证书处理程序，您可以将其实现为共享对象 (.so) 库。构建安全存储实现以读取和写入证书，然后在运行时将库文件链接到 HubOnboarding 进程。

API 定义和组件

查看以下`secure_storage_cert_handler_interface.hpp`文件，了解您的实现的 API 组件和要求

主题

API 定义

`secure_storage_cert_hander_interface.hpp` 的内容

```
/*
    * 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
```

关键组件

- CERT_TYPE_T-集线器上不同类型的证书。
  - CLAIM-最初在集线器上的索赔证书将兑换成永久证书。
  - DHA-暂时未使用。

  - 永久-用于连接托管集成端点的永久证书。
- read_cert_and_private_key-（函数待实现）将证书和密钥值读入参考输入。此函数必须能够读取 CLAIM 和永久证书，并根据上述证书类型进行区分。
- write_permanent_cert_and_private_key-（函数待实现）将永久证书和密钥值写入所需的位置。

示例构建

将内部实现标头与公共接口 (secure_storage_cert_handler_interface.hpp) 分开，以保持干净的项目结构。通过这种分离，您可以在构建证书处理程序的同时管理公用和私有组件。

> **Note**
>
> 宣布secure_storage_cert_handler_interface.hpp为公开。

主题

- 项目结构
- 继承接口
- 实施
- CMakeList.txt

项目结构

继承接口

创建一个继承接口的具体类。将此头文件和其他文件隐藏在单独的目录下，以便在构建时可以轻松区分私有和公共标头。

```
#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
```

实施

实现上面定义的存储类，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;
 }
```

实现接口中定义的工厂函数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
    )
```

使用量

编译完成后，您将拥有一个libSecureStorageCertHandler.so共享的对象库文件及其关联的符号链接。将库文件和符号链接复制到 HubOnboarding 二进制文件所需的库位置。

主题

重要注意事项

- 验证您的用户帐户是否具有 HubOnboarding 二进制文件和libSecureStorageCertHandler.so库的读写权限。
- 保留secure_storage_cert_handler_interface.hpp为唯一的公共头文件。所有其他头文件都应保留在您的私有实现中。
- 验证您的共享对象库名称。在构建时libSecureStorageCertHandler.so，HubOnboarding 可能需要在文件名中使用特定的版本，例如libSecureStorageCertHandler.so.1.0。使用ldd命令检查库依赖关系并根据需要创建符号链接。
- 如果共享库的实现具有外部依赖关系，请将其存储在 HubOnboarding 可以访问的目录中，例如/usr/lib or the iotmi_common目录。

使用安全存储

通过将iot_claim_cert_path和iot_claim_pk_path都设置为来更新您的iotmi_config.json文件**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"
```

```
  }
}
```

进程间通信 (IPC) 客户端 APIs

托管集成中心上的外部组件可以使用其代理组件和进程间通信 (IPC) 与托管集成 Hub SDK 通信。集线器上的一个外部组件示例是管理本地例程的守护程序（一个持续运行的后台进程）。在通信过程中，IPC 客户端是发布命令或其他请求以及订阅事件的外部组件。IPC 服务器是托管集成中心 SDK 中的代理组件。有关更多信息，请参阅设置 IPC 客户端。

为了构建 IPC 客户端，我们提供了 IPC 客户端库`IotmiLocalControllerClient`。此库提供客户端， APIs 用于在 Agent 中与 IPC 服务器进行通信，包括发送命令请求、查询设备状态、订阅事件（如设备状态事件）以及处理基于事件的交互。

主题

- 设置 IPC 客户端
- IPC 接口定义和有效载荷

设置 IPC 客户端

该`IotmiLocalControllerClient`库是基本 IPC 的包装 APIs，它简化和简化了在应用程序中实现 IPC 的过程。以下各节描述了 APIs 它所提供的。

> **Note**
>
> 本主题专门针对作为 IPC 客户端的外部组件，而不是 IPC 服务器的实现。

1. 创建 IPC 客户端

   必须先初始化 IPC 客户端，然后才能使用它来处理请求。可以在`IotmiLocalControllerClient`库中使用构造函数，该构造函数将订阅者上下文 `char *subscriberCtx`作为参数，并基于该构造函数创建 IPC 客户端管理器。以下是创建 IPC 客户端的示例：

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

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

2. 订阅活动

   您可以向 IPC 客户端订阅目标 IPC 服务器的事件。当 IPC 服务器发布客户端已订阅的事件时，客户端将收到该事件。要订阅，请使用`registerSubscriber`函数并提供 IDs 要订阅的事件以及自定义的回调。

   以下是该`registerSubscriber`函数的定义及其用法示例：

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

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

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

   的定义`status`是为了检查操作（如订阅或发送请求）是否成功。如果操作成功，则返回的状态为`IOTMI_STATUS_OK (= 0)`。

> **Note**
>
> 对于订阅中的订阅者和事件的最大数量，IPC 库具有以下服务配额：
>
> - 每个进程的最大订阅者数：5
>
>   定义`IOTMI_IPC_MAX_SUBSCRIBER`在 IPC 库中。
> - 定义的最大事件数：32

定义IOTMI_IPC_EVENT_PUBLIC_END在 IPC 库中。

- 每个订阅者都有一个 32 位的事件字段，其中每个位对应一个定义的事件。

3. 将 IPC 客户端连接到服务器

   IotmiLocalControllerClient库中的 connect 函数执行诸如初始化 IPC 客户端、注册订阅者和订阅函数中提供的事件之类的工作。registerSubscriber您可以在 IPC 客户端上调用连接函数。

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

   在发送请求或进行其他操作IOTMI_STATUS_OK之前，请确认返回的状态为。

4. 发送命令请求和设备状态查询

   代理中的 IPC 服务器可以处理命令请求和设备状态请求。

   - 命令请求

     形成命令请求有效载荷字符串，然后调用sendCommandRequest函数将其发送。例如：

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

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

     有关命令请求格式的更多信息，请参阅命令请求。

     Example 回调函数

     IPC 服务器首先向 IPC 客户端发送消息确认Command received, will send command response back。收到此确认后，IPC 客户端可以预期会出现命令响应事件。

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

    if (ret_status != IOTMI_STATUS_OK)
        return;

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

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

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

- 设备状态请求

  与该sendCommandRequest函数类似，此sendDeviceStateQuery函数还接受有效载荷字符串、相应的回调和客户端上下文。

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

IPC 接口定义和有效载荷

本节重点介绍专门用于代理和外部组件之间通信的 IPC 接口，并提供了这两个组件 APIs 之间的 IPC 实现示例。在以下示例中，外部组件管理本地例程。

在IoTManagedIntegrationsDevice-IPC库中，为代理和外部组件之间的通信定义了以下命令和事件。

```
typedef enum {
```

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

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

命令请求

命令请求格式

- Example 命令请求

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

```
        }]
    }
}
```

命令响应格式

- 如果来自外部组件的命令请求有效，则代理会将其发送到 CDMB（通用数据模型桥）。由于处理命令需要时间，因此包含命令执行时间和其他信息的实际命令响应不会立即发送回外部组件。此命令响应是代理的即时响应（如确认）。该响应告诉外部组件托管集成已收到该命令，如果没有有效的本地令牌，则会对其进行处理或将其丢弃。命令响应以字符串格式发送。

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

设备状态请求

外部组件向代理发送设备状态请求。请求会提供设备managedThingId的，然后代理会回复该设备的状态。

设备状态请求格式

- 您的设备状态请求必须包含managedThingId所查询设备的。

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

设备状态响应格式

- 如果设备状态请求有效，代理将以字符串格式发送回状态。

  Example 有效请求的设备状态响应

```
{
```

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

如果设备状态请求无效（例如没有有效的令牌、无法处理有效负载或其他错误情况），则代理会将响应发回。响应包括错误代码和错误消息。

Example 无效请求的设备状态响应

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

命令响应

Example 命令响应格式

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

```
            ]
        }]
    }
}
```

通知事件

Example 通知事件格式

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

集线器控制

集线器控件是托管集成终端设备 SDK 的扩展，允许它与 Hub SDK 中的`MQTTProxy`组件交互。借助集线器控制，您可以使用终端设备 SDK 实现代码，并通过托管集成云作为单独的设备控制您的集线器。集线器控制 SDK 将作为单独的软件包在 Hub SDK 中提供，标记为`iot-managed-integrations-hub-control-x.x.x`。

主题

先决条件

要设置集线器控制，您需要满足以下条件：

- 已载入 Hub SDK 的集线器，版本 0.4.0 或更高版本。
- 从下载最新版本的终端设备 SDK AWS Management Console。
- 在集线器上运行的 MQTT 代理组件，版本 0.5.0 或更高版本。

终端设备 SDK 组件

使用终端设备 SDK 中的以下组件：

- 数据模型的代码生成器
- 数据模型处理器

由于 Hub SDK 已经具有入门流程和云端连接，因此您不需要以下组件：

- 预备人
- PKCS 接口
- 作业处理者
- MQTT 代理

与终端设备 SDK 集成

1. 按照数据模型代码生成器中的说明生成低级 C 代码。
2. 按照集成终端设备 SDK 中的说明执行以下操作：

a. 设置构建环境

作为开发主机，在亚马逊 Linux 2023/x86_64 上构建代码。安装必要的编译依赖项：

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

b. 开发硬件回调函数

在实现硬件回调函数之前，请先了解 API 的工作原理。此示例使用 On/Off 群集和 OnOff 属性来控制设备功能。有关 API 的详细信息，请参阅低级 C 函数的 API 操作。

```
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. 设置端点并挂接硬件回调函数

实现函数后，创建端点并注册您的回调。完成以下任务：

i. 创建设备代理

ii. 为要支持的每个集群结构填充回调函数点

iii. 设置终端节点并注册支持的集群

```
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);
}
```

示例：构建集线器控件

集线器控件作为 Hub SDK 包的一部分提供。集线器控制子包标有与未经修改的设备 SDK 不同的库，`iot-managed-integrations-hub-control-x.x.x`并且包含不同的库。

1. 将代码生成的文件移到文件`example`夹：

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

2. 要构建集线器控件，请运行以下命令：

   ```
   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. 使用集线器上的MQTTProxy组件运行示例，并运行HubOnboarding和MQTTProxy组件。

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

托管集成数据模型有关数据模型，请参阅。按照中的步骤 5 开始使用终端设备 SDK 设置终端并管理最终用户与之间的通信 iot-managed-integrations。

支持的示例

已构建并测试了以下示例：

- iotmi_device_dm_air_purifier_demo
- iotmi 设备基本诊断
- iotmi_device_dm_camera_demo

支持的平台

下表显示了支持的集线器控制平台。

架构	操作系统	海湾合作委员会版本	Binutils 版本
X86_64	Linux	10.5.0	2.37
aarch64	Linux	10.5.0	2.37

启用 CloudWatch 日志

Hub SDK 提供全面的日志记录功能。默认情况下，Hub SDK 会将日志写入本地文件系统。但是，您可以利用云 API 将日志流配置为 CloudWatch 日志，它提供：

- 监控设备性能：捕获详细的运行时日志，进行主动设备管理。在您的设备群中启用高级日志分析和监控
- 故障排除：生成精细的日志条目以进行快速诊断分析。记录系统和应用程序级事件以进行深入调查。
- 灵活而集中的日志记录：无需直接访问设备即可进行远程日志管理。将来自多个设备的日志聚合到一个可搜索的存储库中。

先决条件

- 将受管设备载入云端。有关详细信息，请参阅Hub 入职设置。
- 验证 Hub 代理已启动并成功初始化。有关详细信息，请参阅安装并验证托管集成 Hub SDK。

> **Note**
>
> 要创建日志配置，详情请参阅 PutRuntimeLogConfiguration API。

> **Warning**
>
> 启用日志计入分层配额计量。增加日志级别将导致更高的消息量和额外的成本。

设置 Hub SDK 日志配置

通过调用 API 来设置运行时日志配置，配置 Hub SDK 日志设置。

Example API 请求示例

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

RuntimeLogConfigurations 属性

以下属性是可选的，可以在 RuntimeLogConfigurations API 中进行配置。

LogLevel

设置运行时跟踪的最低严重性级别。值：DEBUG, ERROR, INFO, WARN

默认：WARN（已发布版本）

LogFlushLevel

确定立即将数据刷新到本地存储的严重性级别。值：DEBUG, ERROR, INFO, WARN

默认值：DISABLED

LocalStoreLocation

指定运行时跟踪的存储位置。默认值：/var/log/awsiotmi

- 活动日志：/var/log/awsiotmi/ManagedIntegrationsDeviceSdkHub.log
- 轮换日志：/var/log/awsiotmi/ManagedIntegrationsDeviceSdkHub.N.log（N 表示轮换顺序）

LocalStoreFileRotationMaxBytes

当当前文件超过指定大小时触发文件轮换。

> **Important**
> 为了获得最佳效率，请将文件大小保持在 125 KB 以下。将自动限制大于 125 KB 的值。

LocalStoreFileRotationMaxFiles,

设置日志守护程序允许的最大轮换文件数。

UploadLog

控制将运行时跟踪传输到云端。日志存储在/aws/iotmanagedintegration CloudWatch 日志组中。

默认值：false。

UploadPeriodMinutes

定义运行时跟踪上传的频率。默认值：5

DeleteLocalStoreAfterUpload

控制上传后的文件删除。默认值：true

> **Note**
>
> 如果设置为 false，则上传的文件将重命名为：`/var/log/awsiotmi/ManagedIntegrationsDeviceSdkHub.uploaded.{uploaded_timestamp}`

示例日志文件

参见下面的 CloudWatch 日志文件示例：

支持的 Zigbee 和 Z-Wave 设备类型

本页列出了已通过托管集成测试并受支持的与集线器连接的设备类型。托管集成同时支持简单设置 (SS)和用户指导设置 (UGS)适用于这些设备。

此表列出了支持的 ZigBee 设备。

Zigbee 设备类型	支持的功能
智能灯泡/可调光灯/RGB 灯	OnOff, LevelControl, ColorControl
智能插头	OnOff
智能开关	OnOff
LED 灯条	OnOff, LevelControl, ColorControl
水阀	OnOff
散热器阀	恒温器，计时 OnOff器
恒温器	恒温器、、 FanControl、定时 OnOff器
车库门开启器	WindowCovering, OnOff, LevelControl
烟雾报警器	BooleanState, OnOff TemperatureMeasurement, 计时器, 烟雾 COAlarm
运动传感器	BooleanState

Zigbee 设备类型	支持的功能
占用/人体存在传感器	BooleanState, OccupancySensing
门窗传感器	BooleanState
漏水传感器	BooleanState
振动传感器	BooleanState
温度和湿度传感器	TemperatureMeasurement, RelativeHumidityMeasurement

下表列出了支持的 Z-Wave 设备。

Z-Wave 设备类型	支持的功能
智能灯泡/可调光灯	OnOff, LevelControl
智能插头	OnOff
车库门控制器	OnOff, LevelControl
能量计	ElectricalEnergyMeasurement, ElectricalPowerMeasurement
电池	LevelControl
警笛	LevelControl
运动传感器	BooleanState
门窗传感器	BooleanState
漏水传感器	BooleanState
温度传感器	TemperatureMeasurement
一氧化碳传感器	抽烟 COAlarm

Z-Wave 设备类型	支持的功能
烟雾传感器	抽烟 COAlarm

场外托管集成中心

Hub SDK 板外流程概述

集线器离线过程会将集线器从 AWS 云 管理系统中移除。当云端发送DeleteManagedThing请求时，该过程可以实现两个主要目标：

设备端操作：

- 重置集线器的内部状态
- 删除所有本地保存的数据
- 为设备做好准备，以备将来可能重新上线

云端操作：

- 移除与中心关联的所有云资源
- 完全断开与先前账户的连接

客户通常在以下情况下启动集线器下线：

- 更改中心的关联账户
- 用新设备替换现有集线器

该过程可确保在集线器配置之间实现干净、安全的过渡，从而实现无缝的设备管理和帐户灵活性。

先决条件

- 你必须有一个已载入的集线器。有关说明，请参阅 Hub 入门设置。
- 在位于/data/aws/iotmi/config/**iotmi_config.json**的文件中，验证是否`iot_provisioning_state`显示`PROVISIONED`。

- 确认中引用的永久证书和密钥`iotmi_config.json`存在于其指定路径中。
- 确保代理 HubOnboarding、置备器和 MQTT 代理已正确配置并正在运行。
- 确认集线器没有子设备。在继续操作之前，请使用 DeleteManagedThingAPI 移除所有子设备。

Hub SDK 场外流程

请按照以下步骤退出集线器：

检索 hub_managed_thing ID

该`iotmi_config.json`文件用于存储托管集成中心的托管事物 ID。此标识符是允许集线器与 AWS IoT 托管集成服务通信的关键信息。托管事物 ID 存储在 JSON 文件的 rw（读写）部分的字段下`managed_thing_id`。在以下示例配置中可以看到这一点：

```
{
      "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
          }
      }
}
```

向机外集线器发送命令

使用您的账户凭证，并使用上一节中`managed_thing_id`检索到的凭据运行命令：

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

```
    --identifier HUB_MANAGED_THING_ID
```

验证集线器已下线

使用您的账户凭证，并使用上一节中managed_thing_id检索到的凭据运行命令：

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

成功和失败场景

成功场景

如果成功执行了移出集线器的命令，则预计会出现以下示例响应：

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

此外，如果集线器离线命令成功，则iotmi_config.json会观察到以下示例。验证 rw 部分是否仅包含可选iot_provisioning_state的元数据。缺少元数据是可以接受的。iot_provisioning_state必须是 NOT_ PROVISIONED。

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

失败场景

如果下线集线器的命令失败，则预计会出现以下示例响应：

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

- 如果ProvisioningStatus是DELETE_IN_PROGRESS，请按照 Hub 恢复中的说明进行操作。
- 如果不ProvisioningStatus是DELETE_IN_PROGRESS，则在托管集成云中关闭集线器的命令要么失败，要么未被托管集成云接收。按照 Hub 恢复中的说明进行操作。
- 如果离线失败，则您的iotmi_config.json文件将类似于下面的示例文件。

```
{
      "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
          }
      }
```

```
}
```

（可选）下线后 Hub SDK

> **Important**
>
> 以下场景列出了在离线 Hub SDK 失败后要采取的可选操作，或者您是否想在离线后重新加入集线器时要采取的操作。

重新登机

如果成功下线，请按照第 3 步：创建托管事物（队列配置）以及其余的板载流程进行加载 Hub SDK。

集线器恢复

设备中心成功下线和云端下线失败

如果 GetManagedThingAPI 调用未返回`Managed Thing resource not found`消息，但文件`iotmi_config.json`已被移除。有关示例 json 文件，请参阅成功场景。

要从这种情况中恢复，请参阅强制删除。

设备中心下线失败

这种情况是指文件`iotmi_config.json`未正确卸载。有关示例 json 文件，请参阅失败场景。

要从这种情况中恢复，请参阅强制删除。如果仍未脱机，`iotmi_config.json`则必须将集线器恢复出厂设置。

设备中心离线和云端离线失败

在这种情况下，仍`iotmi_config.json`未脱机，集线器状态为`ACTIVATED`、或。`DISCOVERED`

要从这种情况中恢复，请参阅强制删除。如果强制删除失败或仍未脱机，`iotmi_config.json`则必须将集线器恢复出厂设置。

集线器处于离线状态且集线器状态为 DELETE_IN_PROGRESS

在这种情况下，集线器处于离线状态，云端收到离线命令。

要从这种情况中恢复，请参阅强制删除。

强制删除

要在设备中心未成功下线的情况下删除云资源，请按照以下步骤操作。此操作可能会导致云端和设备状态不一致，从而可能导致将来的操作出现问题。

使用集线器managed_thing_id和 f orc e 参数调用 DeleteManagedThing API：

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

接下来，调用 GetManagedThing API 并验证它是否返回Managed Thing resource not found。这确认云资源已被删除。

> **Note**
>
> 不建议使用这种方法，因为它可能导致云和设备状态不一致。通常，在尝试删除云资源之前，最好确保设备中心成功下线。

特定于协议的中间件

> **Important**
>
> 此处提供的文档和代码描述了中间件的参考实现。它不是作为 SDK 的一部分提供给您的。

特定于协议的中间件在与底层协议栈交互方面起着至关重要的作用。托管集成 Hub SDK 的设备入门和设备控制组件都使用它来与终端设备进行交互。

中间件执行以下功能。

- 通过提供一 APIs 组通用的协议，从不同供应商的设备协议堆栈中抽出来。 APIs
- 提供软件执行管理，例如线程调度器、事件队列管理和数据缓存。

中间件架构

下面的方框图代表了 Zigbee 中间件的架构。其他协议（如 Z-Wave）的中间件的架构也类似。

特定于协议的中间件有三个主要组件。

- ACS Zigbee DPK：Zigbee 设备移植套件 (DPK) 用于提供对底层硬件和操作系统的抽象，从而实现可移植性。基本上，这可以被视为硬件抽象层 (HAL)，它提供了一组通用集 APIs 来控制来自不同供应商的 Zigbee 无线电并与之通信。Zigbee 中间件包含 Silicon Labs Zigbee 应用程序框架的 DPK API 实现。
- ACS Zigbee 服务：Zigbee 服务作为专用守护程序运行。它包括一个 API 处理程序，通过 IPC 通道为来自客户端应用程序的 API 调用提供服务。AIPC 用作 Zigbee 适配器和 Zigbee 服务之间的 IPC 通道。它还提供其他功能，例如处理这两个 async/sync 命令、处理来自 HAL 的事件以及使用 ACS 事件管理器进行事件注册/发布。
- ACS Zigbee 适配器：Zigbee 适配器是在应用程序进程中运行的库（在本例中，应用程序是 CDMB 插件）。Zigbee 适配器提供了一组供客户端应用程序（例如 CDMB/Provisioner 协议插件）使用，用于控制终端设备并与之通信。 APIs

End-to-end 中间件命令流示例

以下是通过 ZigBee 中间件的命令流示例。

以下是通过 Z-Wave 中间件执行命令流的示例。

特定于协议的中间件代码组织

本节包含有关`IotManagedIntegrationsDeviceSDK-Middleware`存储库中每个组件的代码位置的信息。以下是此存储库中文件夹结构的示例。

```
./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
```

主题

- Zigbee 中间件代码组织

- Z-Wave 中间件代码组织

Zigbee 中间件代码组织

以下显示了 ZigBee 参考中间件代码组织。

主题

- ACS Zigbee DPK
- 硅实验室 Zigbee SDK
- ACS Zigbee 服务
- ACS Zigbee 适配器

ACS Zigbee DPK

Zigbee DPK 的代码位于以下示例中列出的目录中：

```
./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
```

硅实验室 Zigbee SDK

Silicon Labs SDK 显示在IotManagedIntegrationsDeviceSDK-Middleware/*example*-iot-ace-z3-gateway文件夹中。这个 ACS Zigbee DPK 层是为这个 Silicon Labs SDK 实现的。

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

ACS Zigbee 服务

ZigBee 服务的代码位于该文件夹内。IotManagedIntegrationsDeviceSDK-Middleware/*example*-iot-ace-general/middleware/zigbee/此位置的src和include子文件夹包含与 ACS Zigbee 服务相关的所有文件。

```
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 适配器

ACS Zigbee 适配器的代码位于文件夹内。IotManagedIntegrationsDeviceSDK-Middleware/*example*-iot-ace-general/middleware/zigbee/api此位置的src和include子文件夹包含与 ACS Zigbee Adaptor 库相关的所有文件。

```
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 中间件代码组织

以下显示了 Z-Wave 参考中间件代码组织。

主题

- ACS Z-Wave DPK
- 芯科实验室 ZWare 和 Zip 网关
- ACS Z-Wave 服务
- ACS Z-Wave 适配器

ACS Z-Wave DPK

Z-Wave DPK 的代码位于该文件夹内。IotManagedIntegrationsDeviceSDK-Middleware/*example*-iot-ace-dpk/*example*/dpk/ace_hal/zwave

```
./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
```

芯科实验室 ZWare 和 Zip 网关

Silicon Labs ZWare 和 Zip Gateway 的代码位于IotManagedIntegrationsDeviceSDK-Middleware/*example*-iot-ace-z3-gateway文件夹内。这个 ACS Z-Wave DPK 层是为 Z-Wave C APIs 和 Zip 网关实现的。

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

ACS Z-Wave 服务

Z-Wave 服务的代码位于该文件夹中列出的文件夹内。IoTmanagedintegrationsMiddlewares/*example*iot-ace-zwave-mw/此位置的src和include文件夹包含与 ACS Z-Wave 服务相关的所有文件。

```
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 适配器

ACS Zigbee 适配器的代码位于文件夹内。IotManagedIntegrationsDeviceSDK-Middleware/*example*-iot-ace-zwave-mw/cli/此位置的src和include文件夹包含与 ACS Z-Wave 适配器库相关的所有文件。

```
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
```

将中间件与 SDK 集成

以下各节将讨论新集线器上的中间件集成。

主题

设备移植套件 (DPK) API 集成

为了将任何芯片组供应商的 SDK 与中间件集成，中间的 DPK（设备移植套件）层提供了标准的 API 接口。托管集成服务提供商或 ODMs 需要 APIs 根据其物联网中心上使用的 Zigbee/Z-wave/Wi Wi-Fi 芯片组支持的供应商 SDK 来实现这些服务。

参考实现和代码组织

除中间件外，所有其他设备 SDK 组件，例如托管集成 Device Agent 和通用数据模型桥 (CDMB)，无需任何修改即可使用，只需要交叉编译即可。

中间件的实现基于适用于 Zigbee 和 Z-Wave 的 Silicon Labs SDK。如果中间件中的 Silicon Labs SDK 支持新集线器中使用的 Z-Wave 和 Zigbee 芯片组，则无需任何修改即可使用参考中间件。你只需要交叉编译中间件，然后它就可以在新的集线器上运行。

Zigbee 的 DPK（设备移植套件）APIs 可以在中找到`acehal_zigbee.c`，DPK 的参考实现 APIs 位于该文件夹中。`zigbee`

```
IotManagedIntegrationsDeviceSDK-Middleware/example-iot-ace-dpk/example/dpk/ace_hal/
zigbee/
|— CMakeLists.txt
|— include
|—   |— zigbee_log.h
|— src
|—   |— acehal_zigbee.c
|— zigbeed
|—   |— CMakeLists.txt
|—   |— ember
|—   |—   |— ace_ember_common.c
|—   |—   |— ace_ember_ctrl.c
|—   |—   |— ace_ember_hal_callbacks.c
|—   |—   |— ace_ember_network_creator.c
|—   |—   |— ace_ember_power_settings.c
|—   |—   |— ace_ember_zts.c
|—   |— include
|—   |—   |— zbd_api.h
|—   |—   |— zbd_callbacks.h
|—   |—   |— zbd_common.h
|—   |—   |— zbd_network_creator.h
|—   |—   |— zbd_power_settings.h
|—   |—   |— zbd_zts.h
```

Z-Wave APIs 的 DPK 可以在中找到`acehal_zwave.c`，文件夹中有 DPK APIs 的参考实现。`zwaved`

```
IotManagedIntegrationsDeviceSDK-Middleware/example-iot-ace-dpk/example/dpk/ace_hal/
zwave/
|— CMakeLists.txt
|— include
```

```
|—   |— zwave_log.h
|— src
|—   |— acehal_zwave.c
|— zwaved
|—   |— CMakeLists.txt
|—   |— fxnZwaveClient
|—   |—   |— zwave_client.c
|—   |—   |— zwave_client.h
|—   |— include
|—   |—   |— zwaved_cc_intf_api.h
|—   |—   |— zwaved_common_utils.h
|—   |—   |— zwaved_ctrl_api.h
|—   |— zware
|—   |—   |— ace_zware_cc_intf.c
|—   |—   |— ace_zware_common_utils.c
|—   |—   |— ace_zware_ctrl.c
|—   |—   |— ace_zware_debug.c
|—   |—   |— ace_zware_debug.h
|—   |—   |— ace_zware_internal.h
```

作为为不同供应商 SDK 实现 DPK 层的起点，可以使用和修改参考实现。要支持不同的供应商 SDK，需要进行以下两项修改：

1. 将当前供应商 SDK 替换为存储库中的新供应商 SDK。
2. APIs 根据新的供应商 SDK 实现中间件 DPK（设备移植套件）。

托管集成终端设备 SDK

构建一个物联网平台，将智能设备连接到托管集成，并通过统一的控制界面处理命令。终端设备 SDK 可与您的设备固件集成，并通过 SDK 边缘组件提供简化的设置， AWS IoT Core 以及与 AWS IoT 设备管理的安全连接。从下载最新版本的终端设备 SDK AWS Management Console

本指南介绍如何在固件中实现终端设备 SDK。查看架构、组件和集成步骤，开始构建您的实现。

主题

什么是终端设备 SDK？

什么是终端设备 SDK？

终端设备 SDK 是由提供的源代码、库和工具的集合 AWS IoT。该软件开发工具包专为资源有限的环境而构建，支持内存低至 512 KB 和 4 MB 闪存的设备，例如在嵌入式 Linux 和实时操作系统 (RTOS) 上运行的摄像头和空气净化器。从AWS IoT 管理控制台下载最新版本的终端设备 SDK。

核心组件

SDK 结合了用于云通信的 MQTT 代理、用于任务管理的作业处理程序和托管集成（数据模型处理器）。这些组件协同工作，可在您的设备和托管集成之间提供安全的连接和自动数据转换。

有关详细的技术要求，请参阅技术参考。

终端设备 SDK 架构和组件

本节介绍终端设备 SDK 架构及其组件如何与低级 C 函数交互。下图说明了 SDK 框架中的核心组件及其关系。

终端设备 SDK 组件

终端设备 SDK 架构包含以下用于托管集成功能集成的组件：

预备人

在托管集成云中创建设备资源，包括用于安全 MQTT 通信的设备证书和私钥。这些凭证可在您的设备与托管集成之间建立可信连接。

MQTT 代理

通过线程安全 C 客户端库管理 MQTT 连接。此后台进程处理多线程环境中的命令队列，可为内存受限的设备配置队列大小。消息通过托管集成进行处理。

作业处理者

处理设备固件、安全补丁和文件传送的 over-the-air (OTA) 更新。此内置服务管理所有已注册设备的软件更新。

数据模型处理器

使用 AWS“物质数据模型” 的实现，在托管集成和低级 C 函数之间转换操作。有关更多信息，请参阅上的 M atter 文档GitHub。

密钥和证书

通过 PKCS #11 API 管理加密操作，同时支持硬件安全模块和核心等软件实现。PKCS11此 API 在 TLS 连接期间处理 Provisionee 和 MQTT 代理等组件的证书操作。

预备人

provisionee 是托管集成的组成部分，支持按声明进行队列配置。使用配置者，您可以安全地配置您的设备。SDK 为设备配置创建了必要的资源，其中包括从托管集成云中获取的设备证书和私钥。当您想要配置设备时，或者如果有任何更改可能需要您重新配置设备，则可以使用预配者。

主题

- 置备人工作流程
- 设置环境变量
- 注册自定义终端节点

- 创建配置文件
- 创建托管事物
- SDK 用户 Wi-Fi 配置
- 按索赔提供舰队
- 托管事物功能

置备人工作流程

该过程需要在云端和设备端进行设置。客户配置云需求，例如自定义端点、配置文件和托管事物。设备首次开机时，供应者：

1. 使用声明证书连接到托管集成端点
2. 通过队列配置挂钩验证设备参数
3. 在设备上获取并存储永久证书和私钥
4. 设备使用永久证书重新连接
5. 发现设备功能并将其上传到托管集成

成功配置后，设备将直接与托管集成进行通信。预配者仅在重新配置任务时激活。

设置环境变量

在您的云环境中设置以下 AWS 凭据：

```
$ 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
```

注册自定义终端节点

在您的云环境中使用 RegisterCustomEndpointAPI 命令创建用于 device-to-cloud通信的自定义终端节点。

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

响应示例

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

> **Note**
>
> 存储用于配置配置参数的端点地址。使用 GetCustomEndpoint API 返回端点信息。有关更多信息，请参阅GetCustomEndpoint《托管集成 RegisterCustomEndpointAPI 参考指南》中的 API 和 API。

创建配置文件

创建用于定义您的队列配置方法的配置文件。在您的云环境中运行 CreateProvisioningProfileAPI 以返回用于设备身份验证的声明证书和私钥：

```
aws iot-managed-integrations create-provisioning-profile \
--provisioning-type "FLEET_PROVISIONING" \
--name "PROVISIONING-PROFILE-NAME"
```

响应示例

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

您可以实现核心PKCS11 平台抽象库 (PAL)，使核心PKCS11 库与您的设备配合使用。核心 PKCS11 PAL 端口必须提供存储索赔证书和私钥的位置。使用此功能，您可以安全地存储设备的私钥和证书。您可以将私钥和证书存储在硬件安全模块 (HSM) 或可信平台模块 (TPM) 上。

创建托管事物

使用 CreateManagedThingAPI 将您的设备注册到托管集成云。包括设备的序列号 (SN) 和通用产品代码 (UPC)：

```
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;"
```

以下显示了 API 响应示例。

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

API 返回可用于配置验证的托管事物 ID。您需要提供设备序列号 (SN) 和通用产品代码 (UPC)，这些序列号和通用产品代码 (UPC) 在置备交易期间与批准的托管事物相匹配。该交易返回的结果类似于以下内容：

```
/**
 * @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 用户 Wi-Fi 配置

设备制造商和解决方案提供商拥有自己的专有Wi-Fi配置服务，用于接收和配置Wi-Fi凭证。Wi-Fi 配置服务包括使用专用的移动应用程序、低功耗蓝牙 (BLE) 连接和其他专有协议，在初始设置过程中安全地传输 Wi-Fi 凭证。

终端设备 SDK 的使用者必须实现 Wi-Fi 配置服务，设备才能连接到 Wi-Fi 网络。

按索赔提供舰队

使用 provisionee，最终用户可以配置唯一的证书，并使用按声明配置将其注册到托管集成。

客户端 ID 可以从配置模板响应中获取，也可以从设备证书中获取 <common name>“_”<serial number>

托管事物功能

Provisionee 发现托管事物功能，然后将这些功能上传到托管集成。它使应用程序和其他服务可以访问这些功能。设备、其他 Web 客户端和服务可以使用 MQTT 和保留的 MQTT 主题更新功能，也可以使用 REST API 使用 HTTP 来更新功能。

作业处理者

托管集成任务处理程序是一个用于接收现场设备 over-the-air更新的组件。它提供了下载自定义作业文档以进行固件更新或执行远程操作的功能。OTA 更新可用于更新设备固件、修复受影响设备中的安全问题，甚至可以将文件发送到注册了托管集成的设备。

作业处理器的工作原理

在使用作业处理程序之前，需要在云端和设备端执行以下设置步骤。

- 在设备方面，设备制造商为 over-the-air (OTA) 更新准备固件更新方法。
- 在云端，客户准备一份自定义的作业文档，描述远程操作和创建作业。

该过程需要在云端和设备端进行设置。设备制造商在客户准备工作文档和创建更新任务的同时实施固件更新方法。当设备连接时：

1. 设备检索待处理任务列表
2. 作业处理程序会检查列表中是否有一个或多个任务执行，然后选择一个任务
3. 作业处理程序执行作业文档中指定的操作
4. 作业处理程序监视任务执行情况，然后使用SUCCESS或更新作业状态 FAILED

作业处理程序实现

在您的设备上实施用于处理 over-the-air (OTA) 更新的关键操作。您将为任务文档设置 Amazon S3 访问权限，通过 API 创建 OTA 任务，在设备上处理任务文档，并集成 OTA 代理。下图中的步骤说明了终端设备 SDK 与该功能之间的交互如何处理 over-the-air (OTA) 更新请求。

作业处理程序通过以下关键操作处理 OTA 更新：

运营

- 上传并启动更新
- 配置 Amazon S3 访问权限
- 处理工作文档
- 实施 OTA 代理

上传并启动更新

将您的自定义任务文档（JSON 格式）上传到 Amazon S3 存储桶，然后使用 CreateOtaTaskAPI 创建 OTA 任务。包括以下参数：

- `S3Url`: 您的工作文档的 URL 位置
- `Target`: ARN 托管设备阵列（最多 100 台设备）

示例请求

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

配置 Amazon S3 访问权限

添加 Amazon S3 存储桶策略，授予托管集成访问您的任务文档的权限：

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

处理工作文档

创建 OTA 任务时，任务处理程序会在您的设备上运行以下步骤。当有更新可用时，它会通过 MQTT 请求任务文档。

1. 订阅 MQTT 通知主题
2. 为待处理的任务调用 `StartNextPendingJobExecution` API
3. 接收可用的工作文件
4. 根据您指定的超时时间处理更新

使用作业处理程序，应用程序可以决定是立即采取行动，还是等到指定的超时时间。

实施 OTA 代理

当您收到来自托管集成的任务文档时，您必须已实现自己的 OTA 代理，该代理可以处理任务文档、下载更新和执行任何安装操作。OTA 代理需要执行以下步骤。

1. 解析固件 Amazon S3 的任务文档 URLs
2. 通过 HTTP 下载固件更新
3. 验证数字签名
4. 安装经过验证的更新
5. `iotmi_JobsHandler_updateJobStatus`使用`SUCCESS`或`FAILED`状态拨打电话

当您的设备成功完成 OTA 操作后，它必须调用状态为的 `iotmi_JobsHandler_updateJobStatus` API `JobSucceeded` 才能报告成功的作业。

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

数据模型代码生成器

学习如何使用数据模型的代码生成器。生成的代码可用于序列化和反序列化在云端和设备之间交换的数据模型。

项目存储库包含用于创建 C 代码数据模型处理程序的代码生成工具。以下主题描述了代码生成器和工作流程。

主题

- 代码生成过程

- 环境设置
- 为设备生成代码

代码生成过程

代码生成器根据三个主要输入创建 C 源文件： AWS“来自 Zigbee 集群库 (ZCL) 高级平台的物质数据模型（.matter 文件）的实现、处理预处理的 Python 插件和定义代码结构的 Jinja2 模板。在生成过程中，Python 插件通过添加全局类型定义、根据数据类型的依赖关系组织数据类型以及格式化模板渲染信息来处理您的.matter 文件。

下图描述了创建 C 源文件的代码生成器。

终端设备 SDK 包括可在项目codegen.py中使用的 Python 插件和 Jinja2 模板。connectedhomeip这种组合会根据您的.matter 文件输入为每个集群生成多个 C 文件。

以下子主题描述了这些文件。

- Python 插件
- Jinja2 模板
- （可选）自定义架构

Python 插件

代码生成器解析.matter 文件，并将这些信息作为 Python 对象发送到插件。`codegen.py`插件文件会对这些数据`iotmi_data_model.py`进行预处理，并使用提供的模板呈现源文件。预处理包括：

1. 添加不可用的信息`codegen.py`，例如全局类型
2. 对数据类型执行拓扑排序以建立正确的定义顺序

> **Note**
>
> 拓扑排序可确保依赖类型在依赖关系之后定义，无论其原始顺序如何。

Jinja2 模板

终端设备 SDK 提供专为数据模型处理程序和低级 C 函数量身定制的 Jinja2 模板。

Jinja2 模板

模板	生成的来源	备注
cluster.h.jinja	iotmi_device_<cluster>.h	创建低级 C 函数头文件。
cluster.c.jinja	iotmi_device_<cluster>.c	使用数据模型处理程序实现和注册回调函数指针。
cluster_type_helpers.h.jinja	iotmi_device_type_helpers_<cluster>.h	定义数据类型的函数原型。
cluster_type_helpers.c.jinja	iotmi_device_type_helpers_<cluster>.c	为特定于集群的枚举、位图、列表和结构生成数据类型函数原型。
iot_device_dm_types.h.jinja	iotmi_device_dm_types.h	为全局数据类型定义 C 数据类型。
iot_device_type_helpers_global.h.jinja	iotmi_device_type_helpers_global.h	为全局操作定义 C 数据类型。
iot_device_type_helpers_global.c.jinja	iotmi_device_type_helpers_global.c	声明标准数据类型，包括布尔值、整数、浮点数、字符串、位图、列表和结构。

（可选）自定义架构

终端设备 SDK 将标准化代码生成过程与自定义架构相结合。这可以为您的设备和设备软件扩展 Matter 数据模型。自定义架构可以帮助描述设备的 device-to-cloud通信能力。

有关托管集成数据模型的详细信息，包括格式、结构和要求，请参阅托管集成数据模型。

使用codegen.py工具为自定义架构生成 C 源文件，如下所示：

> **Note**
>
> 对于以下三个文件，每个自定义集群都需要相同的集群 ID。

- 创建自定义架构，其JSON格式应为能力报告提供集群的表示形式，以便在云中创建新的自定义集群。示例文件位于`codegen/custom_schemas/custom.SimpleLighting@1.0`。
- 以包含与自定义架构相同信息的XML格式创建 ZCL（Zigbee 集群库）定义文件。使用 ZAP 工具从 ZCL XML 生成你的 Matter IDL 文件。示例文件位于`codegen/zcl/custom.SimpleLighting.xml`。
- ZAP 工具的输出是，它定义了`Matter IDL File (.matter)`与您的自定义架构相对应的 Matter 集群。这是为终端设备 SDK 生成 C 源文件的`codegen.py`工具的输入。示例文件位于`codegen/matter_files/custom-light.matter`。

有关如何将自定义托管集成数据模型集成到代码生成工作流程中的详细说明，请参阅为设备生成代码。

环境设置

了解如何配置您的环境以使用`codegen.py`代码生成器。

主题

先决条件

在配置环境之前，请安装以下项目：

- Git
- Python 3.10 或更高版本
- 诗歌 1.2.0 或更高版本

配置环境

使用以下过程将您的环境配置为使用 codegen.py 代码生成器。

1. 从下载最新版本的终端设备 SDK AWS Management Console。

2. 设置 Python 环境。代码生成项目基于 python，使用 Poetry 进行依赖关系管理。

   - 在codegen目录中使用 poetry 安装项目依赖项：

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

3. 设置您的存储库。

   a. 克隆connectedhomeip存储库。它使用位于connectedhomeip/scripts/文件夹中的codegen.py脚本生成代码。欲了解更多信息，请参阅上的 c onnectedhomeip。GitHub

      ```
      git clone -b v1.4.0.0 https://github.com/project-chip/connectedhomeip.git
      ```

   b. 将其克隆到与IoT-managed-integrations-End-Device-SDK 根文件夹相同的级别。您的文件夹结构应与以下内容相匹配：

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

> **Note**
>
> 你不需要递归克隆子模块。

为设备生成代码

使用托管集成代码生成工具为您的设备创建自定义 C 代码。本节介绍如何从 SDK 附带的示例文件或您自己的规范中生成代码。学习如何使用生成脚本、了解工作流程以及如何创建符合设备要求的代码。

主题

先决条件

1. Python 3.10 或更高版本。

2. 从.matter 文件开始生成代码。终端设备 SDK 在以下文件中提供了两个示例文件codgen/matter_files folder：

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

> **Note**
>
> 这些示例文件为演示应用程序集群生成代码。

生成代码

运行以下命令在 out 文件夹中生成代码：

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

为自定义.matter 文件生成代码

要为特定.matter文件生成代码或提供您自己的.matter文件，请执行以下任务。

为自定义.matter 文件生成代码

1. 准备好你的.matter 文件
2. 运行生成命令：

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

（可选）使用自定义架构生成代码

1. 按JSON格式准备自定义架构
2. 运行生成命令：

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

上面的命令使用多个组件将您的.matter文件转换为C代码：

- `codegen.py`来自ConnectedHome知识产权项目
- Python 插件位于 `codegen/py_scripts/iotmi_data_model.py`
- 文件夹中的 Jinja2 模板 `codegen/py_scripts/templates`

该插件定义了要传递给 Jinja2 模板的变量，然后使用这些变量生成最终的 C 代码输出。添加该`--format`标志会将 Clang 格式应用于生成的代码。

代码生成工作流程

代码生成过程使用实用函数和拓扑排序来组织您的.matter 文件数据结构。`topsort.py`这样可以确保数据类型及其依赖关系的正确排序。

然后，该脚本将您的.matter 文件规范与 Python 插件处理相结合，以提取和格式化必要的信息。最后，它应用 Jinja2 模板格式来创建最终的 C 代码输出。

此工作流程可确保将.matter 文件中的设备特定要求准确地转换为与托管集成系统集成的功能性 C 代码。

低级 C 函数的 API 操作

使用提供的低级 C-Function 将您的设备专用代码与托管集成集成。 APIs本节介绍 AWS 数据模型中每个集群可用的 API 操作，以实现设备到云端的高效交互。了解如何实现回调函数、发出事件、通知属性更改以及为设备终端节点注册集群。

关键的 API 组件包括：

1. 属性和命令的回调函数指针结构
2. 事件发射函数
3. 属性变更通知功能
4. 集群注册功能

通过实现这些功能 APIs，您可以在设备的物理操作和托管集成云功能之间架起一座桥梁，从而确保无缝通信和控制。

以下部分说明了OnOff集群 API。

OnOff 集群 API

集OnOff.xml群支持以下属性和命令:.

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

  对于每个命令，我们都提供了 1:1 映射的函数指针，你可以用它来挂钩你的实现。

属性和命令的所有回调都是在以集群命名的 C 结构中定义的。

示例 C 结构

```
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);
};
```

除了 C 结构外，还为所有属性定义了属性变更报告函数。

```
/* 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);
```

事件报告功能是为所有特定于集群的事件定义的。由于集OnOff群未定义任何事件，因此以下是该CameraAvStreamManagement集群的示例。

```
/* 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);
```

每个集群还具有寄存器功能。

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

传递给寄存器函数的用户指针将传递给回调函数。

托管集成中的功能和设备交互

本节介绍了 C-Function 实现的作用以及设备与托管集成设备功能之间的交互。

主题

- 处理远程命令
- 处理不请自来的事件

处理远程命令

远程命令由终端设备 SDK 与该功能之间的交互来处理。以下操作描述了如何使用此交互打开灯泡的示例。

MQTT 客户端接收有效负载并传递给数据模型处理器

> 当您发送远程命令时，MQTT 客户端会接收 JSON 格式的托管集成消息。然后，它将有效载荷传递给数据模型处理程序。例如，假设你想使用托管集成来打开灯泡。灯泡有一个支持OnOff集群的终端节点 #1。在这种情况下，当您发送打开灯泡的命令时，托管集成会通过 MQTT 向设备发送请求，表示它要在端点 #1 上调用 On 命令。

数据模型处理程序检查回调函数并调用它们

> 数据模型处理程序解析 JSON 请求。如果请求包含属性或操作，则数据模型处理程序会找到端点并按顺序调用相应的回调函数。例如，对于灯泡，当数据模型处理程序收到 MQTT 消息时，它会检查与OnOff集群中定义的 On 命令相对应的回调函数是否已注册到终端节点 #1 上。

处理程序和 C 函数实现执行命令

数据模型处理程序调用它找到的相应回调函数并调用它们。然后，C-Function 实现调用相应的硬件函数来控制物理硬件并返回执行结果。例如，对于灯泡，数据模型处理程序调用回调函数并存储执行结果。然后，回调函数会打开灯泡。

数据模型处理程序返回执行结果

调用所有回调函数后，数据模型处理程序会合并所有结果。然后，它以 JSON 格式打包响应，并使用 MQTT 客户端将结果发布到托管集成云中。对于灯泡，响应中的 MQTT 消息将包含回调函数打开灯泡的结果。

处理不请自来的事件

终端设备 SDK 与该功能之间的交互也会处理未经请求的事件。以下操作描述了操作方法。

设备向数据模型处理器发送通知

当发生属性更改或事件时，例如在设备上按下物理按钮时，C-Function 实现会生成未经请求的事件通知，并调用相应的通知函数将通知发送给数据模型处理程序。

数据模型处理程序翻译通知

数据模型处理程序处理收到的通知并将其转换为 AWS 数据模型。

数据模型处理程序向云端发布通知

然后，数据模型处理程序使用 MQTT 客户端将未经请求的事件发布到托管集成云中。

开始使用终端设备 SDK

按照以下步骤在 Linux 设备上运行终端设备 SDK。本节将指导您完成环境设置、网络配置、硬件功能实现和端点配置。

> **Important**
>
> `examples`目录中的演示应用程序及其中的平台抽象层 (PAL) 实现`platform/posix`仅供参考。请勿在生产环境中使用它们。

仔细查看以下过程的每个步骤，以确保设备与托管集成的正确集成。

集成终端设备 SDK

1. 设置 Amazon EC2 实例

   登录 AWS Management Console 并使用亚马逊 Linux AMI 启动亚马逊 EC2 实例。请参阅《亚马逊弹性容器注册表用户指南》 EC2中的 “亚马逊入门”。

2. 设置构建环境

   作为开发主机，在亚马逊 Linux 2023/x86_64 上构建代码。安装必要的编译依赖项：

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

3. （可选）设置网络

   终端设备 SDK 最好与物理硬件配合使用。如果使用 Amazon EC2，请不要执行此步骤。

   如果您在使用示例应用程序 EC2 之前未使用 Amazon，请初始化网络并将您的设备连接到可用的 Wi-Fi 网络。在设备配置之前完成网络设置：

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

4. 配置配置参数

   > **Note**
   >
   > 在继续操作之前，请按照 Provisionee 获取索赔证书和私钥。

   `example/project_name/device_config.sh`使用以下配置参数修改配置文件：

   配置参数

宏观参数	描述	如何获取此信息
`IOTMI_ROOT_CA_PATH`	根 CA 证书文件。	您可以从AWS IoT Core 开发者指南的下载 Amazon 根 CA 证书部分下载此文件。

宏观参数	描述	如何获取此信息
IOTMI_CLAIM_CERTIFICATE_PATH	索赔证书文件的路径。	要获取声明证书和私钥，请使用CreateProvisioningProfileAPI 创建配置文件。有关说明，请参阅创建配置文件。
IOTMI_CLAIM_PRIVATE_KEY_PATH	声明私钥文件的路径。	
IOTMI_MANAGEDINTEGRATIONS_ENDPOINT	托管集成的终端节点 URL。	要获取托管集成端点，请使用RegisterCustomEndpointAPI。有关说明，请参阅注册自定义终端节点。
IOTMI 托管集成_端点_端口	托管集成端点的端口号	默认情况下，端口 8883 用于 MQTT 发布和订阅操作。端口 443 设置为设备使用的应用层协议协商 (ALPN) TLS 扩展。

5. 构建并运行演示应用程序

本节演示了两个 Linux 演示应用程序：一个简单的安全摄像头和一个空气净化器，两者都 CMake 用作构建系统。

a. 简单的安全摄像头应用程序

要生成并运行应用程序，请执行以下命令：

```
>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
```

此演示为带有 RTC 会话控制器和录制集群的模拟摄像机实现了低级 C 函数。在运行置备人工作流程之前完成中提到的流程。

演示应用程序的输出示例：

```
[2406832727][MAIN][INFO] ======= Device initialization and WIFI provisioning
 =======
[2406832728][MAIN][INFO] fleetProvisioningTemplateName: XXXXXXXXXXX
[2406832728][MAIN][INFO] managedintegrationsEndpoint: XXXXXXXXX.account-prefix-
ats.iot.region.amazonaws.com
[2406832728][MAIN][INFO] pDeviceSerialNumber: XXXXXXXXXXXX
[2406832728][MAIN][INFO] universalProductCode: XXXXXXXXXXXX
[2406832728][MAIN][INFO] rootCertificatePath: XXXXXXXXX
[2406832728][MAIN][INFO] pClaimCertificatePath: XXXXXXXX
[2406832728][MAIN][INFO] pClaimKeyPath: XXXXXXXXXXXXXXXXX
[2406832728][MAIN][INFO] deviceInfo.serialNumber XXXXXXXXXXXX
[2406832728][MAIN][INFO] deviceInfo.universalProductCode XXXXXXXXXXXXXXX
[2406832728][PKCS11][INFO] PKCS #11 successfully initialized.
[2406832728][MAIN][INFO] ============= Start certificate provisioning
 =============
[2406832728][PKCS11][INFO] ======== Loading Root CA and claim credentials
 through PKCS#11 interface ========
[2406832728][PKCS11][INFO] Writing certificate into label "Root Cert".
[2406832728][PKCS11][INFO] Creating a 0x1 type object.
[2406832728][PKCS11][INFO] Writing certificate into label "Claim Cert".
[2406832728][PKCS11][INFO] Creating a 0x1 type object.
[2406832728][PKCS11][INFO] Creating a 0x3 type object.
[2406832728][MAIN][INFO] ======== Fleet-provisioning-by-Claim ========
[2025-01-02 01:43:11.404995144][iotmi_device_sdkLog][INFO] [2406832728]
[MQTT_AGENT][INFO]
[2025-01-02 01:43:11.405106991][iotmi_device_sdkLog][INFO] Establishing a TLS
 session to XXXXXXXXXXXXXXX.account-prefix-ats.iot.region.amazonaws.com
[2025-01-02 01:43:11.405119166][iotmi_device_sdkLog][INFO]
[2025-01-02 01:43:11.844812513][iotmi_device_sdkLog][INFO] [2406833168]
[MQTT_AGENT][INFO]
[2025-01-02 01:43:11.844842576][iotmi_device_sdkLog][INFO] TLS session
 connected
[2025-01-02 01:43:11.844852105][iotmi_device_sdkLog][INFO]
[2025-01-02 01:43:12.296421687][iotmi_device_sdkLog][INFO] [2406833620]
[MQTT_AGENT][INFO]
[2025-01-02 01:43:12.296449663][iotmi_device_sdkLog][INFO] Session present: 0.
[2025-01-02 01:43:12.296458997][iotmi_device_sdkLog][INFO]
[2025-01-02 01:43:12.296467793][iotmi_device_sdkLog][INFO] [2406833620]
[MQTT_AGENT][INFO]
[2025-01-02 01:43:12.296476275][iotmi_device_sdkLog][INFO] MQTT connect with
 clean session.
```

```
[2025-01-02 01:43:12.296484350][iotmi_device_sdkLog][INFO]
[2025-01-02 01:43:13.171056119][iotmi_device_sdkLog][INFO] [2406834494]
[FLEET_PROVISIONING][INFO]
[2025-01-02 01:43:13.171082442][iotmi_device_sdkLog][INFO] Received accepted
 response from Fleet Provisioning CreateKeysAndCertificate API.
[2025-01-02 01:43:13.171092740][iotmi_device_sdkLog][INFO]
[2025-01-02 01:43:13.171122834][iotmi_device_sdkLog][INFO] [2406834494]
[FLEET_PROVISIONING][INFO]
[2025-01-02 01:43:13.171132400][iotmi_device_sdkLog][INFO] Received privatekey
 and certificate with Id: 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 :
 XXXXXXXXXXXXXXXXXXXX_XXXXXXXXXXXXXXXXXXXXXXXX
[2406840540][MAIN][INFO] Managed thing ID : XXXXXXXXXXXXXXXXXXXXXXX
[2406840540][MAIN][INFO] ======================== application loop
 =================
[2025-01-02 01:43:19.217094828][iotmi_device_sdkLog][INFO] [2406840540]
[MQTT_AGENT][INFO]
[2025-01-02 01:43:19.217124600][iotmi_device_sdkLog][INFO] Establishing a TLS
 session to XXXXXXXXX.account-prefix-ats.iot.region.amazonaws.com:8883
[2025-01-02 01:43:19.217138724][iotmi_device_sdkLog][INFO]
[2406840540][Cluster OnOff][INFO] exampleOnOffInitCluster() for endpoint#1
```

```
[2406840540][MAIN][INFO] Press Ctrl+C when you finish testing...
[2406840540][Cluster ActivatedCarbonFilterMonitoring][INFO]
 exampleActivatedCarbonFilterMonitoringInitCluster() for endpoint#1
[2406840540][Cluster AirQuality][INFO] exampleAirQualityInitCluster() for
 endpoint#1
[2406840540][Cluster CarbonDioxideConcentrationMeasurement][INFO]
 exampleCarbonDioxideConcentrationMeasurementInitCluster() for endpoint#1
[2406840540][Cluster FanControl][INFO] exampleFanControlInitCluster() for
 endpoint#1
[2406840540][Cluster HepaFilterMonitoring][INFO]
 exampleHepaFilterMonitoringInitCluster() for endpoint#1
[2406840540][Cluster Pm1ConcentrationMeasurement][INFO]
 examplePm1ConcentrationMeasurementInitCluster() for endpoint#1
[2406840540][Cluster Pm25ConcentrationMeasurement][INFO]
 examplePm25ConcentrationMeasurementInitCluster() for endpoint#1
[2406840540][Cluster TotalVolatileOrganicCompoundsConcentrationMeasurement]
[INFO]
 exampleTotalVolatileOrganicCompoundsConcentrationMeasurementInitCluster() for
 endpoint#1
[2025-01-02 01:43:19.648185488][iotmi_device_sdkLog][INFO] [2406840971]
[MQTT_AGENT][INFO]
[2025-01-02 01:43:19.648211988][iotmi_device_sdkLog][INFO] TLS session
 connected
[2025-01-02 01:43:19.648225583][iotmi_device_sdkLog][INFO]

[2025-01-02 01:43:19.938281231][iotmi_device_sdkLog][INFO] [2406841261]
[MQTT_AGENT][INFO]
[2025-01-02 01:43:19.938304799][iotmi_device_sdkLog][INFO] Session present: 0.
[2025-01-02 01:43:19.938317404][iotmi_device_sdkLog][INFO]
```

b. 简单的空气净化器应用

要生成并运行应用程序，请运行以下命令：

```
>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
```

此演示为具有两个端点和以下支持的集群的模拟空气净化器实现了低级 C 函数：

空气净化器端点支持的集群

终端节点	集群
终点 #1: 空气净化器	OnOff
	风扇控制
	HEPA 过滤器监测
	活性炭过滤器监测
终点 #2: 空气质量传感器	空气质量
	二氧化碳浓度测量
	甲醛浓度测量
	Pm25 浓度测量
	Pm1 浓度测量
	总挥发性有机化合物浓度测量

输出与相机演示应用程序类似，但支持的集群不同。

6. 后续步骤：

托管集成终端设备软件开发工具包和演示应用程序现已在您的 Amazon EC2 实例上运行。这使您可以在自己的物理硬件上开发和测试应用程序。通过此设置，您可以利用托管集成服务来控制您的 AWS IoT 设备。

a. 开发硬件回调函数

在实现硬件回调函数之前，请先了解 API 的工作原理。此示例使用 On/Off 群集和OnOff属性来控制设备功能。有关 API 的详细信息，请参阅低级 C 函数的 API 操作。

```
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. 设置端点并挂接硬件回调函数

实现函数后，创建端点并注册您的回调。完成以下任务：

i. 创建设备代理。

A. 在调用任何其他 SDK 函数`iotmiDev_Agent_new()`之前，使用创建设备代理。

B. 您的配置必须至少包含 t hingID 和 clientID 参数。

C. 使用该`iotmiDev_Agent_initDefaultConfig()`函数为队列大小和最大端点等参数设置合理的默认值。

D. 使用完资源后，使用该`iotmiDev_Agent_free()`函数将其释放。这样可以防止内存泄漏并确保在应用程序中进行适当的资源管理。

ii. 为要支持的每个集群结构填充回调函数指针。

iii. 设置终端节点并注册支持的集群。

使用创建终端节点`iotmiDev_Agent_addEndpoint()`，这需要：

A. 唯一的终端节点 ID。

B. 描述性端点名称

C. 一种或多种符合 AWS 数据模型定义的设备类型。

D. 创建终端节点后，使用相应的集群特定注册功能注册集群。

E. 每个集群注册都需要使用属性和命令的回调函数。系统会将您的用户上下文指针传递给回调，以在两次调用之间保持状态。

```
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. 使用作业处理程序获取作业文档

i. 发起对您的 OTA 应用程序的调用：

```
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. 调用iotmi_JobsHandler_start以初始化作业处理程序。

iii. iotmi_JobsHandler_getJobDocument致电从托管集成中检索任务文档。

iv. 成功获取任务文档后，在processOTA函数中写入您的自定义 OTA 操作并返回JobSucceeded状态。

```
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." ) );
```

```
}
```

将终端设备 SDK 移植到您的设备上

将终端设备 SDK 移植到您的设备平台。按照以下步骤将您的设备与 AWS IoT 设备管理连接起来。

下载并验证终端设备 SDK

1. 从托管集成控制台下载最新版本的终端设备 SDK。
2. 验证您的平台是否在支持的平台列表中参考：支持的平台。

> **Note**
>
> 终端设备 SDK 已在指定平台上进行了测试。其他平台可能有效，但尚未经过测试。

3. 将 SDK 文件提取（解压缩）到您的工作区。
4. 使用以下设置配置您的构建环境：
   - 源文件路径
   - 头文件目录
   - 所需的库
   - 编译器和链接器标志
5. 在移植平台抽象层 (PAL) 之前，请确保平台的基本功能已初始化。功能包括：
   - 操作系统任务
   - 外围设备
   - 网络接口
   - 特定于平台的要求

将 PAL 移植到您的设备上

1. 在现有平台目录中为特定于平台的实现创建一个新目录。例如，如果您使用 FreeRTOS，请在上创建一个目录。`platform/freertos`

Example SDK 目录结构

```
### <SDK_ROOT_FOLDER>
#   ### CMakeLists.txt
#   ### LICENSE.txt
#   ### cmake
#   ### commonDependencies
#   ### components
#   ### docs
#   ### examples
#   ### include
#   ### lib
#   ### platform
#   ### test
#   ### tools
```

2. 将 POSIX 参考实现文件（.c 和.h）从 posix 文件夹复制到新的平台目录中。这些文件为你需要实现的函数提供了一个模板。

   - 凭据存储的闪存管理
   - PKCS #11 的实现
   - 网络传输接口
   - 时间同步
   - 系统重启和重置功能
   - 日志记录机制
   - 特定于设备的配置

3. 使用 TLS 设置传输层安全 (TLS) 身份验证。 MBed

   - 如果您已经有与平台上的 SDK 版本相匹配的 MBed TLS 版本，请使用提供的 POSIX 实现。
   - 使用不同的 TLS 版本，您可以使用堆栈为 TLS 堆栈实现传输挂钩。 TCP/IP

4. 将您平台的 mbedTLS 配置与中的软件开发工具包要求进行比较。`platform/posix/mbedtls/mbedtls_config.h`确保所有必需的选项都已启用。

5. 该软件开发工具包依赖 CoreMQTT 与云端进行交互。因此，您必须实现使用以下结构的网络传输层：

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

有关更多信息，请参阅 FreeRTOS 网站上的传输接口文档。

6. （可选）SDK 使用 PCKS #11 API 来处理证书操作。CorePKCS 是用于原型设计的非硬件特定的 PKCS #11 实现。我们建议您在生产环境中使用安全的加密处理器，例如可信平台模块 (TPM)、硬件安全模块 (HSM) 或安全元素：

   - 查看使用 Linux 文件系统进行凭据管理的 PKCS #11 实现示例，网址为。`platform/posix/corePKCS11-mbedtls`
   - 在以下位置实现 PKCS #11 PAL 层。`commonDependencies/core_pkcs11/corePKCS11/source/include/core_pkcs11.h`
   - 在上实现 Linux 文件系统`platform/posix/corePKCS11-mbedtls/source/iotmi_pal_Pkcs11Operations.c`。
   - 在上实现您的存储类型的存储和加载功能`platform/include/iotmi_pal_Nvm.h`。
   - 在中实现标准文件访问权限`platform/posix/source/iotmi_pal_Nvm.c`。

   有关详细的移植说明，请参阅 FreeRTOS 用户指南中的移植核心PKCS11库。

7. 将 SDK 静态库添加到您的构建环境中：

   - 设置库路径以解决任何链接器问题或符号冲突
   - 验证所有依赖关系是否正确关联

测试你的端口

您可以使用现有的示例应用程序来测试您的端口。编译完成时必须没有任何错误或警告。

> **Note**
>
> 我们建议您从尽可能简单的多任务处理应用程序开始。示例应用程序提供了等效的多任务处理功能。

1. 在中查找示例应用程序`examples/[device_type_sample]`。
2. 将`main.c`文件转换为您的项目，然后添加一个条目来调用现有的 main () 函数。
3. 确认您可以成功编译演示应用程序。

技术参考

主题

参考：支持的平台

下表显示了 SDK 支持的平台。

支持的平台

平台	架构	操作系统
Linux x86_64	x86_64	Linux
Ambarella	Armv8 () AArch64	Linux
ameBad	armv8-M 32 位	FreeRTOS
ESP32S3	Xtensa 32 位 LX7	FreeRTOS

参考：技术要求

下表显示了 SDK 的技术要求，包括 RAM 空间。使用相同的配置时，终端设备 SDK 本身需要大约 5 到 10 MB 的 ROM 空间。

内存空间

软件开发工具包和组件	空间要求（已用字节）
终端设备 SDK 本身	180 KB

软件开发工具包和组件	空间要求（已用字节）
默认 MQTT 代理命令队列	480 字节（可以配置）
默认 MQTT 代理传入队列	320 字节（可以配置）

参考：常用 API

本部分列出了非特定于集群的 API 操作。

```
/* 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);
```

托管集成中的安全性 AWS IoT Device Management

云安全 AWS 是重中之重。作为 AWS 客户，您可以受益于专为满足大多数安全敏感型组织的要求而构建的数据中心和网络架构。

安全是双方共同承担 AWS 的责任。责任共担模式将其描述为云的安全性和云中的安全性：

- 云安全 — AWS 负责保护在云中运行 AWS 服务的基础架构 AWS 云。 AWS 还为您提供可以安全使用的服务。作为AWS 合规计划合规计划合规计划合的一部分，第三方审计师定期测试和验证我们安全的有效性。要了解适用于托管集成的合规性计划，请参阅按合规计划划分的范围内的AWS 服务按合规计划。
- 云端安全-您的责任由您使用的 AWS 服务决定。您还需要对其他因素负责，包括您的数据的敏感性、您公司的要求以及适用的法律法规。

本文档可帮助您了解在使用托管集成时如何应用责任共担模型。以下主题向您介绍如何配置托管集成以满足您的安全性和合规性目标。您还将学习如何使用其他 AWS 服务来帮助您监控和保护托管集成资源。

主题

托管集成中的数据保护

分 AWS 担责任模型适用于托管集成中的数据保护。 AWS IoT Device Management如本模型所述 AWS ，负责保护运行所有内容的全球基础架构 AWS 云。您负责维护对托管在此基础结构上的内容的控制。您还负责您所使用的 AWS 服务 的安全配置和管理任务。有关数据隐私的更多信息，请参阅数据隐私常见问题。有关欧洲数据保护的信息，请参阅 AWS Security Blog 上的 AWS Shared Responsibility Model and GDPR 博客文章。

出于数据保护目的，我们建议您保护 AWS 账户 凭证并使用 AWS IAM Identity Center 或 AWS Identity and Access Management (IAM) 设置个人用户。这样，每个用户只获得履行其工作职责所需的权限。还建议您通过以下方式保护数据：

- 对每个账户使用多重身份验证（MFA）。
- 用于 SSL/TLS 与 AWS 资源通信。我们要求使用 TLS 1.2，建议使用 TLS 1.3。
- 使用设置 API 和用户活动日志 AWS CloudTrail。有关使用 CloudTrail 跟踪捕获 AWS 活动的信息，请参阅《AWS CloudTrail 用户指南》中的使用跟 CloudTrail 踪。
- 使用 AWS 加密解决方案以及其中的所有默认安全控件 AWS 服务。
- 使用高级托管安全服务（例如 Amazon Macie），它有助于发现和保护存储在 Amazon S3 中的敏感数据。
- 如果您在 AWS 通过命令行界面或 API 进行访问时需要经过 FIPS 140-3 验证的加密模块，请使用 FIPS 端点。有关可用的 FIPS 端点的更多信息，请参阅《美国联邦信息处理标准（FIPS）第 140-3 版》。

强烈建议您切勿将机密信息或敏感信息（如您客户的电子邮件地址）放入标签或自由格式文本字段（如名称字段）。这包括您 AWS 服务 使用控制台、API AWS IoT Device Management 或为或其他人处理托管集成时。 AWS CLI AWS SDKs在用于名称的标签或自由格式文本字段中输入的任何数据都可能会用于计费或诊断日志。如果您向外部服务器提供网址，强烈建议您不要在网址中包含凭证信息来验证对该服务器的请求。

用于托管集成的静态数据加密

的托管集成默认 AWS IoT Device Management 提供数据加密，以使用加密密钥保护敏感的静态客户数据。

有两种类型的加密密钥用于保护托管集成客户的敏感数据：

客户管理的密钥 (CMK)

托管集成支持使用您可以创建、拥有和管理的对称客户托管密钥。您可以完全控制这些 KMS 密钥，包括建立和维护其密钥策略、IAM policy 和授权、启用和禁用它们、轮换其加密材料、添加标签、创建别名（引用了 KMS 密钥）以及计划删除 KMS 密钥。

AWS 拥有的密钥

默认情况下，托管集成使用这些密钥来自动加密敏感的客户数据。您无法查看、管理或审核其使用情况。您无需采取任何措施或更改任何程序即可保护加密数据的密钥。默认情况下，静态数据加密有助于

降低保护敏感数据的操作开销和复杂性。同时，它还支持构建符合严格加密合规性和监管要求的安全应用程序。

使用的默认加密密钥是 AWS 自有密钥。或者，更新加密密钥的可选 API 是PutDefaultEncryptionConfiguration。

有关 AWS KMS 加密密钥类型的更多信息，请参阅AWS KMS 密钥。

AWS KMS 用于托管集成

托管集成使用信封加密来加密和解密所有客户数据。这种类型的加密将获取您的纯文本数据，并使用数据密钥对其进行加密。接下来，称为包装密钥的加密密钥将对用于加密纯文本数据的原始数据密钥进行加密。在信封加密中，可以使用其他包装密钥来加密与原始数据密钥分离程度更接近的现有包装密钥。由于原始数据密钥由单独存储的包装密钥加密，因此您可以将原始数据密钥和加密的纯文本数据存储在同一位置。密钥环用于生成、加密和解密数据密钥，此外还使用包装密钥来加密和解密数据密钥。

> **Note**
>
> AWS 数据库加密 SDK 为您的客户端加密实现提供信封加密。有关 AWS 数据库加密 SDK 的更多信息，请参阅什么是 AWS 数据库加密 SDK？

有关信封加密、数据密钥、包装密钥和密钥环的更多信息，请参阅信封加密、数据密钥、包装密钥和密钥环。

托管集成要求服务使用您的客户托管密钥进行以下内部操作：

- 向发送DescribeKey请求， AWS KMS 以验证轮换数据密钥时提供的对称客户托管密钥 ID。
- 向发送GenerateDataKeyWithoutPlaintext请求 AWS KMS 以生成由您的客户托管密钥加密的数据密钥。
- 向发送ReEncrypt*请求， AWS KMS 要求使用您的客户托管密钥重新加密数据密钥。
- 向发送Decrypt请求，要求 AWS KMS 使用您的客户托管密钥解密数据。

使用加密密钥加密的数据类型

托管集成使用加密密钥来加密静态存储的多种类型的数据。以下列表概述了使用加密密钥进行静态加密的数据的类型：

- 云到云 (C2C) 连接器事件，例如设备发现和设备状态更新。

- 创建代表物理设备的托管事物和包含特定设备类型功能的设备配置文件。有关设备和设备配置文件的更多信息，请参阅设备和设备。
- 有关设备实现各个方面的托管集成通知。有关托管集成通知的更多信息，请参阅设置托管集成通知。
- 最终用户的个人身份信息 (PII)，例如设备身份验证材料、设备序列号、最终用户姓名、设备标识符和设备 Amazon 资源名称 (arn)。

托管集成如何使用关键策略 AWS KMS

对于分支密钥轮换和异步调用，托管集成需要密钥策略才能使用您的加密密钥。使用密钥策略的原因如下：

- 以编程方式授权其他 AWS 委托人使用加密密钥。

有关在托管集成中用于管理加密密钥访问权限的密钥策略示例，请参阅 创建加密密钥

> **Note**
>
> 对于 AWS 拥有的密钥，不需要密钥策略，因为 AWS 拥有的密钥归其所有 AWS ，您无法查看、管理或使用它。默认情况下，托管集成使用 AWS 自有的密钥来自动加密您的敏感客户数据。

除了使用密钥策略来管理带有 AWS KMS 密钥的加密配置外，托管集成还使用 IAM 策略。有关 IAM 策略的更多信息，请参阅中的策略和权限 AWS Identity and Access Management。

创建加密密钥

您可以使用 AWS Management Console 或创建加密密钥 AWS KMS APIs。

创建加密密钥

按照 AWS Key Management Service 开发人员指南中创建 KMS 密钥的步骤进行操作。

密钥策略

密钥政策声明控制对 AWS KMS 密钥的访问权限。每个 AWS KMS 密钥将仅包含一个密钥策略。该密钥策略决定了哪些 AWS 委托人可以使用密钥以及他们如何使用密钥。有关使用密钥策略声明管理 AWS KMS 密钥访问和使用的更多信息，请参阅使用策略管理访问权限。

以下是密钥政策声明的示例，您可以使用它来管理托管集成中存储的 AWS KMS 密钥 AWS 账户 的访问和使用情况：

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

有关密钥库的更多信息，请参阅密钥库。

更新加密配置

无缝更新加密配置的能力对于管理托管集成的数据加密实施至关重要。当您最初使用托管集成时，系统将提示您选择加密配置。您可以选择默认 AWS 拥有的密钥或创建自己的密 AWS KMS 钥。

AWS Management Console

要在中更新您的加密配置 AWS Management Console，请打开 AWS IoT 服务主页，然后导航到统一控制的托管集成 > 设置 > 加密。在加密设置窗口中，您可以通过选择新的密 AWS KMS 钥来更新您的加密配置，以获得额外的加密保护。选择“自定义加密设置（高级）”以选择现有 AWS KMS 密钥，也可以选择“创建 AWS KMS 密钥”来创建自己的客户托管密钥。

API 命令

在托管集成中，有两种方法 APIs 用于管理 AWS KMS 密钥的加密配置：PutDefaultEncryptionConfiuration和GetDefaultEncryptionConfiguration。

要更新默认加密配置，请调用PutDefaultEncryptionConfiuration。有关 PutDefaultEncryptionConfiuration 的更多信息，请参阅PutDefaultEncryptionConfiuration。

要查看默认加密配置，请致电GetDefaultEncryptionConfiguration。有关 GetDefaultEncryptionConfiguration 的更多信息，请参阅GetDefaultEncryptionConfiguration。

托管集成的身份和访问管理

AWS Identity and Access Management (IAM) AWS 服务 可帮助管理员安全地控制对 AWS 资源的访问权限。IAM 管理员控制谁可以进行身份验证（登录）和授权（有权限）使用托管集成资源。您可以使用 IAM AWS 服务 ，无需支付额外费用。

主题

受众

您的使用方式 AWS Identity and Access Management (IAM) 会有所不同，具体取决于您在托管集成中所做的工作。

服务用户-如果您使用托管集成服务完成工作，则您的管理员会为您提供所需的凭证和权限。当你使用更多的托管集成功能来完成工作时，你可能需要额外的权限。了解如何管理访问权限有助于您向管理员请求适合的权限。如果您无法访问托管集成中的某项功能，请参阅对托管集成、身份和访问进行故障排除。

服务管理员-如果您负责公司的托管集成资源，则可能拥有对托管集成的完全访问权限。您的工作是确定您的服务用户应访问哪些托管集成、功能和资源。然后，您必须向 IAM 管理员提交请求以更改服务用户的权限。请查看该页面上的信息以了解 IAM 的基本概念。要详细了解您的公司如何将 IAM 与托管集成一起使用，请参阅托管集成如何与 IAM 配合使用。

IAM 管理员 — 如果您是 IAM 管理员，则可能需要详细了解如何编写策略来管理托管集成的访问权限。要查看您可以在 IAM 中使用的基于身份的托管集成策略示例，请参阅。托管集成的基于身份的策略示例

使用身份进行身份验证

身份验证是您 AWS 使用身份凭证登录的方式。您必须以 IAM 用户身份或通过担 AWS 账户根用户任 IAM 角色进行身份验证（登录 AWS）。

您可以使用通过身份源提供的凭据以 AWS 联合身份登录。 AWS IAM Identity Center （IAM Identity Center）用户、贵公司的单点登录身份验证以及您的 Google 或 Facebook 凭据就是联合身份的示例。当您以联合身份登录时，您的管理员以前使用 IAM 角色设置了身份联合验证。当你使用联合访问 AWS 时，你就是在间接扮演一个角色。

根据您的用户类型，您可以登录 AWS Management Console 或 AWS 访问门户。有关登录的更多信息 AWS，请参阅《AWS 登录 用户指南》中的如何登录到您 AWS 账户的。

如果您 AWS 以编程方式访问，则会 AWS 提供软件开发套件 (SDK) 和命令行接口 (CLI)，以便使用您的凭据对请求进行加密签名。如果您不使用 AWS 工具，则必须自己签署请求。有关使用推荐的方法自行签署请求的更多信息，请参阅《IAM 用户指南》中的用于签署 API 请求的AWS 签名版本 4。

无论使用何种身份验证方法，您都可能需要提供其他安全信息。例如， AWS 建议您使用多重身份验证 (MFA) 来提高账户的安全性。要了解更多信息，请参阅《AWS IAM Identity Center 用户指南》中的多重身份验证和《IAM 用户指南》中的 IAM 中的AWS 多重身份验证。

AWS 账户 root 用户

创建时 AWS 账户，首先要有一个登录身份，该身份可以完全访问账户中的所有资源 AWS 服务 和资源。此身份被称为 AWS 账户 root 用户，使用您创建账户时使用的电子邮件地址和密码登录即可访问该身份。强烈建议您不要使用根用户执行日常任务。保护好根用户凭证，并使用这些凭证来执行仅根用户可以执行的任务。有关要求您以根用户身份登录的任务的完整列表，请参阅 IAM 用户指南中的需要根用户凭证的任务。

联合身份

作为最佳实践，要求人类用户（包括需要管理员访问权限的用户）使用与身份提供商的联合身份验证 AWS 服务 通过临时证书进行访问。

联合身份是指您的企业用户目录、Web 身份提供商、Identity C enter 目录中的用户，或者任何使用 AWS 服务 通过身份源提供的凭据进行访问的用户。 AWS Directory Service当联合身份访问时 AWS 账户，他们将扮演角色，角色提供临时证书。

要集中管理访问权限，建议您使用 AWS IAM Identity Center。您可以在 IAM Identity Center 中创建用户和群组，也可以连接并同步到您自己的身份源中的一组用户和群组，以便在您的所有 AWS 账户 和

应用程序中使用。有关 IAM Identity Center 的信息，请参阅 AWS IAM Identity Center 用户指南中的什么是 IAM Identity Center？。

IAM 用户和群组

I AM 用户是您 AWS 账户 内部对个人或应用程序具有特定权限的身份。在可能的情况下，我们建议使用临时凭证，而不是创建具有长期凭证（如密码和访问密钥）的 IAM 用户。但是，如果您有一些特定的使用场景需要长期凭证以及 IAM 用户，建议您轮换访问密钥。有关更多信息，请参阅《IAM 用户指南》中的对于需要长期凭证的用例，应在需要时更新访问密钥。

IAM 组是一个指定一组 IAM 用户的身份。您不能使用组的身份登录。您可以使用组来一次性为多个用户指定权限。如果有大量用户，使用组可以更轻松地管理用户权限。例如，您可以拥有一个名为的群组，IAMAdmins并向该群组授予管理 IAM 资源的权限。

用户与角色不同。用户唯一地与某个人员或应用程序关联，而角色旨在让需要它的任何人代入。用户具有永久的长期凭证，而角色提供临时凭证。要了解更多信息，请参阅《IAM 用户指南》中的 IAM 用户的使用案例。

IAM 角色

I AM 角色是您内部具有特定权限 AWS 账户 的身份。它类似于 IAM 用户，但与特定人员不关联。要在中临时担任 IAM 角色 AWS Management Console，您可以从用户切换到 IAM 角色（控制台）。您可以通过调用 AWS CLI 或 AWS API 操作或使用自定义 URL 来代入角色。有关使用角色的方法的更多信息，请参阅《IAM 用户指南》中的代入角色的方法。

具有临时凭证的 IAM 角色在以下情况下很有用：

- 联合用户访问：要向联合身份分配权限，请创建角色并为角色定义权限。当联合身份进行身份验证时，该身份将与角色相关联并被授予由此角色定义的权限。有关用于联合身份验证的角色的信息，请参阅《IAM 用户指南》中的针对第三方身份提供商创建角色（联合身份验证）。如果您使用 IAM Identity Center，则需要配置权限集。为控制您的身份在进行身份验证后可以访问的内容，IAM Identity Center 将权限集与 IAM 中的角色相关联。有关权限集的信息，请参阅《AWS IAM Identity Center 用户指南》中的权限集。
- 临时 IAM 用户权限：IAM 用户可代入 IAM 用户或角色，以暂时获得针对特定任务的不同权限。
- 跨账户存取：您可以使用 IAM 角色以允许不同账户中的某个人（可信主体）访问您的账户中的资源。角色是授予跨账户访问权限的主要方式。但是，对于某些资源 AWS 服务，您可以将策略直接附加到资源（而不是使用角色作为代理）。要了解用于跨账户访问的角色和基于资源的策略之间的差别，请参阅 IAM 用户指南中的 IAM 中的跨账户资源访问。

- 跨服务访问 — 有些 AWS 服务 使用其他 AWS 服务服务中的功能。例如，当您在服务中拨打电话时，该服务通常会在 Amazon 中运行应用程序 EC2 或在 Amazon S3 中存储对象。服务可能会使用发出调用的主体的权限、使用服务角色或使用服务相关角色来执行此操作。
  - 转发访问会话 (FAS) — 当您使用 IAM 用户或角色在中执行操作时 AWS，您被视为委托人。使用某些服务时，您可能会执行一个操作，然后此操作在其他服务中启动另一个操作。FAS 使用调用委托人的权限以及 AWS 服务 向下游服务发出请求的请求。 AWS 服务只有当服务收到需要与其他 AWS 服务 或资源交互才能完成的请求时，才会发出 FAS 请求。在这种情况下，您必须具有执行这两项操作的权限。有关发出 FAS 请求时的策略详情，请参阅转发访问会话。
  - 服务角色 - 服务角色是服务代表您在您的账户中执行操作而分派的 IAM 角色。IAM 管理员可以在 IAM 中创建、修改和删除服务角色。有关更多信息，请参阅《IAM 用户指南》中的创建向 AWS 服务委派权限的角色。
  - 服务相关角色-服务相关角色是一种与服务相关联的服务角色。 AWS 服务服务可以代入代表您执行操作的角色。服务相关角色出现在您的中 AWS 账户 ，并且归服务所有。IAM 管理员可以查看但不能编辑服务相关角色的权限。
- 在 A@@ mazon 上运行的应用程序 EC2 — 您可以使用 IAM 角色管理在 EC2 实例上运行并发出 AWS CLI 或 AWS API 请求的应用程序的临时证书。这比在 EC2 实例中存储访问密钥更可取。要为 EC2 实例分配 AWS 角色并使其可供其所有应用程序使用，您需要创建一个附加到该实例的实例配置文件。实例配置文件包含该角色，并允许在 EC2 实例上运行的程序获得临时证书。有关更多信息，请参阅 IAM 用户指南中的使用 IAM 角色向在 A mazon EC2 实例上运行的应用程序授予权限。

使用策略管理访问

您可以 AWS 通过创建策略并将其附加到 AWS 身份或资源来控制中的访问权限。策略是其中的一个对象 AWS ，当与身份或资源关联时，它会定义其权限。 AWS 在委托人（用户、root 用户或角色会话）发出请求时评估这些策略。策略中的权限确定是允许还是拒绝请求。大多数策略都以 JSON 文档的 AWS 形式存储在中。有关 JSON 策略文档的结构和内容的更多信息，请参阅 IAM 用户指南中的 JSON 策略概览。

管理员可以使用 AWS JSON 策略来指定谁有权访问什么。也就是说，哪个主体可以对什么资源执行操作，以及在什么条件下执行。

默认情况下，用户和角色没有权限。要授予用户对所需资源执行操作的权限，IAM 管理员可以创建 IAM 策略。管理员随后可以向角色添加 IAM 策略，用户可以代入角色。

IAM 策略定义操作的权限，无关乎您使用哪种方法执行操作。例如，假设您有一个允许 `iam:GetRole` 操作的策略。拥有该策略的用户可以从 AWS Management Console AWS CLI、或 AWS API 获取角色信息。

基于身份的策略

基于身份的策略是可附加到身份（如 IAM 用户、用户组或角色）的 JSON 权限策略文档。这些策略控制用户和角色可在何种条件下对哪些资源执行哪些操作。要了解如何创建基于身份的策略，请参阅《IAM 用户指南》中的使用客户托管策略定义自定义 IAM 权限。

基于身份的策略可以进一步归类为内联策略或托管式策略。内联策略直接嵌入单个用户、组或角色中。托管策略是独立的策略，您可以将其附加到中的多个用户、群组和角色 AWS 账户。托管策略包括 AWS 托管策略和客户托管策略。要了解如何在托管策略和内联策略之间进行选择，请参阅《IAM 用户指南》中的在托管策略与内联策略之间进行选择。

基于资源的策略

基于资源的策略是附加到资源的 JSON 策略文档。基于资源的策略的示例包括 IAM 角色信任策略和 Amazon S3 存储桶策略。在支持基于资源的策略的服务中，服务管理员可以使用它们来控制对特定资源的访问。对于在其中附加策略的资源，策略定义指定主体可以对该资源执行哪些操作以及在什么条件下执行。您必须在基于资源的策略中指定主体。委托人可以包括账户、用户、角色、联合用户或 AWS 服务。

基于资源的策略是位于该服务中的内联策略。您不能在基于资源的策略中使用 IAM 中的 AWS 托管策略。

访问控制列表 (ACLs)

访问控制列表 (ACLs) 控制哪些委托人（账户成员、用户或角色）有权访问资源。 ACLs 与基于资源的策略类似，尽管它们不使用 JSON 策略文档格式。

Amazon S3 和 Amazon VPC 就是支持的服务示例 ACLs。 AWS WAF要了解更多信息 ACLs，请参阅《亚马逊简单存储服务开发者指南》中的访问控制列表 (ACL) 概述。

其他策略类型

AWS 支持其他不太常见的策略类型。这些策略类型可以设置更常用的策略类型向您授予的最大权限。

- 权限边界：权限边界是一个高级特征，用于设置基于身份的策略可以为 IAM 实体（IAM 用户或角色）授予的最大权限。您可为实体设置权限边界。这些结果权限是实体基于身份的策略及其权限边界的交集。在 `Principal` 中指定用户或角色的基于资源的策略不受权限边界限制。任一项策略中的显式拒绝将覆盖允许。有关权限边界的更多信息，请参阅IAM 用户指南中的 IAM 实体的权限边界。
- 服务控制策略 (SCPs)- SCPs 是指定组织或组织单位 (OU) 的最大权限的 JSON 策略 AWS Organizations。 AWS Organizations 是一项用于对您的企业拥有的多 AWS 账户 项进行分组和集中

管理的服务。如果您启用组织中的所有功能，则可以将服务控制策略 (SCPs) 应用于您的任何或所有帐户。SCP 限制成员账户中的实体（包括每个 AWS 账户根用户实体）的权限。有关 Organization SCPs s 和的更多信息，请参阅《AWS Organizations 用户指南》中的服务控制策略。

- 资源控制策略 (RCPs) — RCPs 是 JSON 策略，您可以使用它来设置账户中资源的最大可用权限，而无需更新附加到您拥有的每个资源的 IAM 策略。RCP 限制成员账户中资源的权限，并可能影响身份（包括身份）的有效权限 AWS 账户根用户，无论这些身份是否属于您的组织。有关 Organizations 的更多信息 RCPs，包括 AWS 服务 该支持的列表 RCPs，请参阅《AWS Organizations 用户指南》中的资源控制策略 (RCPs)。
- 会话策略：会话策略是当您以编程方式为角色或联合用户创建临时会话时作为参数传递的高级策略。结果会话的权限是用户或角色的基于身份的策略和会话策略的交集。权限也可以来自基于资源的策略。任一项策略中的显式拒绝将覆盖允许。有关更多信息，请参阅 IAM 用户指南中的会话策略。

多个策略类型

当多个类型的策略应用于一个请求时，生成的权限更加复杂和难以理解。要了解在涉及多种策略类型时如何 AWS 确定是否允许请求，请参阅 IAM 用户指南中的策略评估逻辑。

AWS 托管集成的托管策略

要向用户、群组和角色添加权限，使用 AWS 托管策略比自己编写策略要容易得多。创建仅为团队提供所需权限的 IAM 客户管理型策略需要时间和专业知识。要快速入门，您可以使用我们的 AWS 托管策略。这些策略涵盖常见使用案例，可在您的 AWS 账户中使用。有关 AWS 托管策略的更多信息，请参阅 IAM 用户指南中的AWS 托管策略。

AWS 服务维护和更新 AWS 托管策略。您无法更改 AWS 托管策略中的权限。服务偶尔会向 AWS 托管式策略添加额外权限以支持新特征。此类更新会影响附加策略的所有身份（用户、组和角色）。当启动新特征或新操作可用时，服务最有可能会更新 AWS 托管式策略。服务不会从 AWS 托管策略中移除权限，因此策略更新不会破坏您的现有权限。

此外，还 AWS 支持跨多个服务的工作职能的托管策略。例如，ReadOnlyAccess AWS 托管策略提供对所有 AWS 服务和资源的只读访问权限。当服务启动新功能时， AWS 会为新操作和资源添加只读权限。有关工作职能策略的列表和说明，请参阅 IAM 用户指南中的适用于工作职能的AWS 托管式策略。

AWS 托管策略： AWSIoTManagedIntegrationsFullAccess

您可以将 AWSIoTManagedIntegrationsFullAccess 策略附加到 IAM 身份。

此政策授予对托管集成和相关服务的完全访问权限。要在中查看此政策 AWS Management Console，请参阅AWSIoTManagedIntegrationsFullAccess。

权限详细信息

该策略包含以下权限：

- iotmanagedintegrations— 向您添加此策略的 IAM 用户、群组和角色提供对托管集成和相关服务的完全访问权限。
- iam— 允许分配的 IAM 用户、群组和角色在中创建 AWS 账户服务相关角色。

```
{
  "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 托管策略： AWS Io TManaged IntegrationsRolePolicy

您可以将 AWS IoTManagedIntegrationsRolePolicy 策略附加到 IAM 身份。

该政策授予托管集成代表您发布 Amazon CloudWatch 日志和指标的权限。

要在中查看此政策 AWS Management Console，请参阅AWSIoTManagedIntegrationsRolePolicy。

权限详细信息

该策略包含以下权限。

- logs— 提供创建 Amazon CloudWatch 日志组并将日志流式传输到这些组的功能。
- cloudwatch— 提供发布 Amazon CloudWatch 指标的功能。有关亚马逊 CloudWatch 指标的更多信息，请参阅亚马逊指标 CloudWatch。

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

托管集成对托 AWS 管策略的更新

查看自该服务开始跟踪 AWS 托管集成的托管策略更新以来这些变更的详细信息。要获得有关此页面变更的自动提醒，请在托管集成文档历史记录页面上订阅 RSS feed。

更改	描述	日期
托管集成已开始跟踪更改	托管集成开始跟踪其 AWS 托管策略的更改。	2025 年 3 月 3 日

托管集成如何与 IAM 配合使用

在使用 IAM 管理托管集成的访问权限之前，请先了解托管集成可以使用哪些 IAM 功能。

可在托管集成中使用的 IAM 功能

IAM 特征	托管集成支持
基于身份的策略	是

IAM 特征	托管集成支持
基于资源的策略	否
策略操作	是
策略资源	是
策略条件键	是
ACLs	否
ABAC（策略中的标签）	否
临时凭证	是
主体权限	是
服务角色	是
服务相关角色	是

要全面了解托管集成和其他 AWS 服务如何与大多数 IAM 功能配合使用，请参阅 IAM 用户指南中的与 IAM 配合使用的AWS 服务。

用于托管集成的基于身份的策略

支持基于身份的策略：是

基于身份的策略是可附加到身份（如 IAM 用户、用户组或角色）的 JSON 权限策略文档。这些策略控制用户和角色可在何种条件下对哪些资源执行哪些操作。要了解如何创建基于身份的策略，请参阅《IAM 用户指南》中的使用客户管理型策略定义自定义 IAM 权限。

通过使用 IAM 基于身份的策略，您可以指定允许或拒绝的操作和资源以及允许或拒绝操作的条件。您无法在基于身份的策略中指定主体，因为它适用于其附加的用户或角色。要了解可在 JSON 策略中使用的所有元素，请参阅《IAM 用户指南》中的 IAM JSON 策略元素引用。

托管集成的基于身份的策略示例

要查看托管集成基于身份的策略的示例，请参阅。托管集成的基于身份的策略示例

托管集成中基于资源的策略

支持基于资源的策略：否

基于资源的策略是附加到资源的 JSON 策略文档。基于资源的策略的示例包括 IAM 角色信任策略和 Amazon S3 存储桶策略。在支持基于资源的策略的服务中，服务管理员可以使用它们来控制对特定资源的访问。对于在其中附加策略的资源，策略定义指定主体可以对该资源执行哪些操作以及在什么条件下执行。您必须在基于资源的策略中指定主体。委托人可以包括账户、用户、角色、联合用户或 AWS 服务。

要启用跨账户访问，您可以将整个账户或其他账户中的 IAM 实体指定为基于资源的策略中的主体。将跨账户主体添加到基于资源的策略只是建立信任关系工作的一半而已。当委托人和资源处于不同位置时 AWS 账户，可信账户中的 IAM 管理员还必须向委托人实体（用户或角色）授予访问资源的权限。他们通过将基于身份的策略附加到实体以授予权限。但是，如果基于资源的策略向同一个账户中的主体授予访问权限，则不需要额外的基于身份的策略。有关更多信息，请参阅《IAM 用户指南》中的 IAM 中的跨账户资源访问。

托管集成的政策措施

支持策略操作：是

管理员可以使用 AWS JSON 策略来指定谁有权访问什么。也就是说，哪个主体可以对什么资源执行操作，以及在什么条件下执行。

JSON 策略的 `Action` 元素描述可用于在策略中允许或拒绝访问的操作。策略操作通常与关联的 AWS API 操作同名。有一些例外情况，例如没有匹配 API 操作的仅限权限 操作。还有一些操作需要在策略中执行多个操作。这些附加操作称为相关操作。

在策略中包含操作以授予执行关联操作的权限。

要查看托管集成操作列表，请参阅《服务授权参考》中的托管集成定义的操作。

托管集成中的策略操作在操作前使用以下前缀：

```
iot-mi
```

要在单个语句中指定多项操作，请使用逗号将它们隔开。

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

```
      "iot-mi:action2"
         ]
```

要查看托管集成基于身份的策略的示例，请参阅。托管集成的基于身份的策略示例

托管集成的策略资源

支持策略资源：是

管理员可以使用 AWS JSON 策略来指定谁有权访问什么。也就是说，哪个主体可以对什么资源执行操作，以及在什么条件下执行。

Resource JSON 策略元素指定要向其应用操作的一个或多个对象。语句必须包含 Resource 或 NotResource 元素。作为最佳实践，请使用其 Amazon 资源名称（ARN）指定资源。对于支持特定资源类型（称为资源级权限）的操作，您可以执行此操作。

对于不支持资源级权限的操作（如列出操作），请使用通配符（*）指示语句应用于所有资源。

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

要查看托管集成资源类型及其列表 ARNs，请参阅《服务授权参考》中的“托管集成定义的资源”。要了解您可以使用哪些操作来指定每种资源的 ARN，请参阅托管集成定义的操作。

要查看托管集成基于身份的策略的示例，请参阅。托管集成的基于身份的策略示例

托管集成的策略条件密钥

支持特定于服务的策略条件键：是

管理员可以使用 AWS JSON 策略来指定谁有权访问什么。也就是说，哪个主体可以对什么资源执行操作，以及在什么条件下执行。

在 Condition 元素（或 Condition 块）中，可以指定语句生效的条件。Condition 元素是可选的。您可以创建使用条件运算符（例如，等于或小于）的条件表达式，以使策略中的条件与请求中的值相匹配。

如果您在一个语句中指定多个 Condition 元素，或在单个 Condition 元素中指定多个键，则 AWS 使用逻辑 AND 运算评估它们。如果您为单个条件键指定多个值，则使用逻辑OR运算来 AWS 评估条件。在授予语句的权限之前必须满足所有的条件。

在指定条件时，您也可以使用占位符变量。例如，只有在使用 IAM 用户名标记 IAM 用户时，您才能为其授予访问资源的权限。有关更多信息，请参阅《IAM 用户指南》中的 IAM 策略元素：变量和标签。

AWS 支持全局条件密钥和特定于服务的条件密钥。要查看所有 AWS 全局条件键，请参阅 IAM 用户指南中的AWS 全局条件上下文密钥。

要查看托管集成条件密钥列表，请参阅《服务授权参考》中的托管集成的条件密钥。要了解可以使用条件键的操作和资源，请参阅托管集成定义的操作。

要查看托管集成基于身份的策略的示例，请参阅。托管集成的基于身份的策略示例

ACLs 在托管集成中

支持 ACLs：否

访问控制列表 (ACLs) 控制哪些委托人（账户成员、用户或角色）有权访问资源。 ACLs 与基于资源的策略类似，尽管它们不使用 JSON 策略文档格式。

带有托管集成的ABAC

支持 ABAC（策略中的标签）：部分支持

基于属性的访问控制（ABAC）是一种授权策略，该策略基于属性来定义权限。在中 AWS，这些属性称为标签。您可以向 IAM 实体（用户或角色）和许多 AWS 资源附加标签。标记实体和资源是 ABAC 的第一步。然后设计 ABAC 策略，以在主体的标签与他们尝试访问的资源标签匹配时允许操作。

ABAC 在快速增长的环境中非常有用，并在策略管理变得繁琐的情况下可以提供帮助。

要基于标签控制访问，您需要使用 `aws:ResourceTag/key-name`、`aws:RequestTag/key-name` 或 `aws:TagKeys` 条件键在策略的条件元素中提供标签信息。

如果某个服务对于每种资源类型都支持所有这三个条件键，则对于该服务，该值为是。如果某个服务仅对于部分资源类型支持所有这三个条件键，则该值为部分。

有关 ABAC 的更多信息，请参阅《IAM 用户指南》中的使用 ABAC 授权定义权限。要查看设置 ABAC 步骤的教程，请参阅《IAM 用户指南》中的使用基于属性的访问权限控制（ABAC）。

在托管集成中使用临时证书

支持临时凭证：是

当你使用临时证书登录时，有些 AWS 服务 不起作用。有关更多信息，包括哪些 AWS 服务 适用于临时证书，请参阅 IAM 用户指南中的AWS 服务 与 IAM 配合使用的信息。

如果您使用除用户名和密码之外的任何方法登录，则 AWS Management Console 使用的是临时证书。例如，当您 AWS 使用公司的单点登录 (SSO) 链接进行访问时，该过程会自动创建临时证书。当您以用户身份登录控制台，然后切换角色时，您还会自动创建临时凭证。有关切换角色的更多信息，请参阅《IAM 用户指南》中的从用户切换到 IAM 角色（控制台）。

您可以使用 AWS CLI 或 AWS API 手动创建临时证书。然后，您可以使用这些临时证书进行访问 AWS。 AWS 建议您动态生成临时证书，而不是使用长期访问密钥。有关更多信息，请参阅 IAM 中的临时安全凭证。

托管集成的跨服务主体权限

支持转发访问会话（FAS）：是

当您使用 IAM 用户或角色在中执行操作时 AWS，您被视为委托人。使用某些服务时，您可能会执行一个操作，然后此操作在其他服务中启动另一个操作。FAS 使用调用委托人的权限以及 AWS 服务 向下游服务发出请求的请求。 AWS 服务只有当服务收到需要与其他 AWS 服务 或资源交互才能完成的请求时，才会发出 FAS 请求。在这种情况下，您必须具有执行这两项操作的权限。有关发出 FAS 请求时的策略详情，请参阅转发访问会话。

托管集成的服务角色

支持服务角色：是

服务角色是由一项服务担任、代表您执行操作的 IAM 角色。IAM 管理员可以在 IAM 中创建、修改和删除服务角色。有关更多信息，请参阅《IAM 用户指南》中的创建向 AWS 服务委派权限的角色。

> **Warning**
>
> 更改服务角色的权限可能会破坏托管集成功能。只有当托管集成提供了相关指导时，才可以编辑服务角色。

托管集成的服务相关角色

支持服务相关角色：是

服务相关角色是一种与服务相关联的 AWS 服务服务角色。服务可以代入代表您执行操作的角色。服务相关角色出现在您的中 AWS 账户 ，并且归服务所有。IAM 管理员可以查看但不能编辑服务相关角色的权限。

有关创建或管理服务相关角色的详细信息，请参阅能够与 IAM 搭配使用的AWS 服务。在表中查找服务相关角色列中包含 Yes 的表。选择是链接以查看该服务的服务相关角色文档。

托管集成的基于身份的策略示例

默认情况下，用户和角色无权创建或修改托管集成资源。他们也无法使用 AWS Management Console、 AWS Command Line Interface (AWS CLI) 或 AWS API 执行任务。要授予用户对所需资源执行操作的权限，IAM 管理员可以创建 IAM 策略。管理员随后可以向角色添加 IAM 策略，用户可以代入角色。

要了解如何使用这些示例 JSON 策略文档创建基于 IAM 身份的策略，请参阅《IAM 用户指南》中的创建 IAM 策略（控制台）。

有关托管集成定义的操作和资源类型（包括每种资源类型的格式）的详细信息，请参阅《服务授权参考》中的 “托管集成的操作、资源和条件密钥”。 ARNs

主题

- 策略最佳实践
- 使用托管集成控制台
- 允许用户查看他们自己的权限

策略最佳实践

基于身份的策略决定了某人是否可以在您的账户中创建、访问或删除托管集成资源。这些操作可能会使 AWS 账户产生成本。创建或编辑基于身份的策略时，请遵循以下指南和建议：

- 开始使用 AWS 托管策略并转向最低权限权限 — 要开始向用户和工作负载授予权限，请使用为许多常见用例授予权限的AWS 托管策略。它们在你的版本中可用 AWS 账户。我们建议您通过定义针对您的用例的 AWS 客户托管策略来进一步减少权限。有关更多信息，请参阅《IAM 用户指南》中的AWS 托管式策略或工作职能的AWS 托管式策略。
- 应用最低权限：在使用 IAM 策略设置权限时，请仅授予执行任务所需的权限。为此，您可以定义在特定条件下可以对特定资源执行的操作，也称为最低权限许可。有关使用 IAM 应用权限的更多信息，请参阅《IAM 用户指南》中的 IAM 中的策略和权限。
- 使用 IAM 策略中的条件进一步限制访问权限：您可以向策略添加条件来限制对操作和资源的访问。例如，您可以编写策略条件来指定必须使用 SSL 发送所有请求。如果服务操作是通过特定 AWS 服务的（例如）使用的，则也可以使用条件来授予对服务操作的访问权限 AWS CloudFormation。有关更多信息，请参阅《IAM 用户指南》中的 IAM JSON 策略元素：条件。

- 使用 IAM Access Analyzer 验证您的 IAM 策略，以确保权限的安全性和功能性 – IAM Access Analyzer 会验证新策略和现有策略，以确保策略符合 IAM 策略语言（JSON）和 IAM 最佳实践。IAM Access Analyzer 提供 100 多项策略检查和可操作的建议，以帮助您制定安全且功能性强的策略。有关更多信息，请参阅《IAM 用户指南》中的使用 IAM Access Analyzer 验证策略。
- 需要多重身份验证 (MFA)-如果 AWS 账户您的场景需要 IAM 用户或根用户，请启用 MFA 以提高安全性。若要在调用 API 操作时需要 MFA，请将 MFA 条件添加到您的策略中。有关更多信息，请参阅《IAM 用户指南》中的使用 MFA 保护 API 访问。

有关 IAM 中的最佳实操的更多信息，请参阅《IAM 用户指南》中的 IAM 中的安全最佳实践。

使用托管集成控制台

要访问托管集成控制台，您必须拥有一组最低权限。这些权限必须允许您列出和查看有关您的 AWS 账户托管集成资源的详细信息。如果创建比必需的最低权限更为严格的基于身份的策略，对于附加了该策略的实体（用户或角色），控制台将无法按预期正常运行。

对于仅调用 AWS CLI 或 AWS API 的用户，您无需为其设置最低控制台权限。相反，只允许访问与其尝试执行的 API 操作相匹配的操作。

为确保用户和角色仍然可以使用托管集成控制台，还需要将托管集成*ConsoleAccess*或托*ReadOnly* AWS 管策略附加到实体。有关更多信息，请参阅《IAM 用户指南》中的为用户添加权限。

允许用户查看他们自己的权限

该示例说明了您如何创建策略，以允许 IAM 用户查看附加到其用户身份的内联和托管式策略。此策略包括在控制台上或使用 AWS CLI 或 AWS API 以编程方式完成此操作的权限。

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

对托管集成、身份和访问进行故障排除

使用以下信息来帮助您诊断和修复在使用托管集成和 IAM 时可能遇到的常见问题。

主题

- 我无权在托管集成中执行任何操作
- 我无权执行 iam：PassRole
- 我想允许我以外的人 AWS 账户 访问我的托管集成资源

我无权在托管集成中执行任何操作

如果您收到错误提示，指明您无权执行某个操作，则必须更新策略以允许执行该操作。

当 mateojackson IAM 用户尝试使用控制台查看有关虚构 *my-example-widget* 资源的详细信息，但不拥有虚构 iot-mi:*GetWidget* 权限时，会发生以下示例错误。

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

在此情况下，必须更新 mateojackson 用户的策略，以允许使用 iot-mi:*GetWidget* 操作访问 *my-example-widget* 资源。

如果您需要帮助，请联系您的 AWS 管理员。您的管理员是提供登录凭证的人。

我无权执行 iam：PassRole

如果您收到错误消息，指出您无权执行该iam:PassRole操作，则必须更新您的策略以允许您将角色传递给托管集成。

有些 AWS 服务 允许您将现有角色传递给该服务，而不是创建新的服务角色或服务相关角色。为此，您必须具有将角色传递到服务的权限。

当名为的 IAM 用户marymajor尝试使用控制台在托管集成中执行操作时，会出现以下示例错误。但是，服务必须具有服务角色所授予的权限才可执行此操作。Mary 不具有将角色传递到服务的权限。

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

在这种情况下，必须更新 Mary 的策略以允许她执行 iam:PassRole 操作。

如果您需要帮助，请联系您的 AWS 管理员。您的管理员是提供登录凭证的人。

我想允许我以外的人 AWS 账户 访问我的托管集成资源

您可以创建一个角色，以便其他账户中的用户或您组织外的人员可以使用该角色来访问您的资源。您可以指定谁值得信赖，可以代入角色。对于支持基于资源的策略或访问控制列表 (ACLs) 的服务，您可以使用这些策略向人们授予访问您的资源的权限。

要了解更多信息，请参阅以下内容：

- 要了解托管集成是否支持这些功能，请参阅托管集成如何与 IAM 配合使用。
- 要了解如何提供对您拥有的资源的访问权限 AWS 账户 ，请参阅 IAM 用户指南中的向您拥有 AWS 账户 的另一个 IAM 用户提供访问权限。
- 要了解如何向第三方提供对您的资源的访问权限 AWS 账户，请参阅 IAM 用户指南中的向第三方提供访问权限。 AWS 账户
- 要了解如何通过身份联合验证提供访问权限，请参阅《IAM 用户指南》中的为经过外部身份验证的用户（身份联合验证）提供访问权限。

- 要了解使用角色和基于资源的策略进行跨账户访问之间的差别，请参阅《IAM 用户指南》中的 IAM 中的跨账户资源访问。

使用服务相关角色进行托管集成

AWS IoT 设备管理用户 AWS Identity and Access Management (IAM) 服务相关角色的托管集成。服务相关角色是一种独特的 IAM 角色，直接链接到托管集成。服务相关角色由托管集成预定义，包括该服务代表您调用其他 AWS 服务所需的所有权限。

服务相关角色可以更轻松地设置托管集成，因为您不必手动添加必要的权限。 AWS IoT 设备管理的托管集成定义了其服务相关角色的权限，除非另有定义，否则只有托管集成才能扮演其角色。定义的权限包括信任策略和权限策略，以及不能附加到任何其他 IAM 实体的权限策略。

只有在首先删除相关资源后，您才能删除服务相关角色。这可以保护您的托管集成资源，因为您不会无意中移除对资源的访问权限。

有关支持服务相关角色的其他服务的信息，请参阅与 IAM 配合使用的AWS 服务，并在服务相关角色列中查找标有 “是” 的服务。选择是和链接，查看该服务的服务相关角色文档。

托管集成的服务相关角色权限

AWS IoT 设备管理托管集成使用名为 Inte AWSServiceRoleForIoTManagedgrat ions 的服务相关角色——为 AWS IoT 设备管理提供托管集成，允许您发布日志和指标。

AWSServiceRoleForIoTManaged集成服务相关角色信任以下服务来代替该角色：

- `iotmanagedintegrations.amazonaws.com`

名为的角色权限策略 AWSIoTManagedIntegrationsServiceRolePolicy 允许托管集成对指定资源完成以下操作：

- 操作：`on all of your managed integrations resources.` 上的 `logs:CreateLogGroup, logs:DescribeLogGroups, logs:CreateLogStream, logs:PutLogEvents, logs:DescribeLogStreams, cloudwatch:PutMetricData`

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

您必须配置使用户、组或角色能够创建、编辑或删除服务相关角色的权限。有关更多信息，请参阅《IAM 用户指南》中的服务相关角色权限。

为托管集成创建服务相关角色

您无需手动创建服务相关角色。当您创建事件类型（例如在PutRuntimeLogConfiguration、或 API 中调用CreateEventLogConfiguration AWS Management Console、或RegisterCustomEndpoint AP AWS I 命令）时，托管集成会为您创建服务相关角色。 AWS CLI有关PutRuntimeLogConfiguration、CreateEventLogConfiguration或的更多信息RegisterCustomEndpoint，请参阅PutRuntimeLogConfigurationCreateEventLogConfiguration、或RegisterCustomEndpoint。

如果您删除该服务相关角色，然后需要再次创建，您可以使用相同流程在账户中重新创建此角色。当您创建事件类型（例如调用PutRuntimeLogConfigurationCreateEventLogConfiguration、或RegisterCustomEndpoint API 命令）时，托管集成会再次为您创建服务相关角色。或者，您可以通过以下方式联系 AWS 客户支持 AWS Support Center Console。有关 AWS 支持计划的更多信息，请参阅比较 AWS Support 计划。

您还可以使用 IAM 控制台通过 I oT ManagedIntegrations -托管角色用例创建服务相关角色。在 AWS CLI 或 AWS API 中，使用服务名称创建服务相关角色。iotmanagedintegrations.amazonaws.com有关更多信息，请参阅 IAM 用户指南 中的创建服务相关角色。如果您删除了此服务相关角色，可以使用同样的过程再次创建角色。

编辑托管集成的服务相关角色

托管集成不允许您编辑 AWSServiceRoleForIoTManaged集成服务相关角色。创建服务相关角色后，您将无法更改角色的名称，因为可能有多种实体引用该角色。但是可以使用 IAM 编辑角色描述。有关更多信息，请参阅《IAM 用户指南》中的编辑服务相关角色。

删除托管集成的服务相关角色

如果不再需要使用某个需要服务相关角色的功能或服务，我们建议您删除该角色。这样就没有未被主动监控或维护的未使用实体。但是，必须先清除服务相关角色的资源，然后才能手动删除它。

> **Note**
>
> 如果您尝试删除资源时，托管集成正在使用该角色，则删除可能会失败。如果发生这种情况，请等待几分钟后重试。

使用 IAM 手动删除服务相关角色

使用 IAM 控制台 AWS CLI、或 AWS API 删除 AWSServiceRoleForIoTManaged集成服务相关角色。有关更多信息，请参阅《IAM 用户指南》中的删除服务相关角色。

托管集成服务相关角色支持的区域

AWS IoT 设备管理托管集成支持在提供服务的所有区域中使用服务相关角色。有关更多信息，请参阅 AWS 区域和端点。

用 AWS Secrets Manager 于 C2C 工作流程的数据保护

AWS Secrets Manager 是一项密钥存储服务，可用于保护数据库凭据、API 密钥和其他机密信息。然后，您可以将硬编码凭证改为对 Secrets Manager 进行 API 调用。这有助于确保别人在检查您的代码时不会泄露密钥，因为代码中没有密钥。有关概述，请参阅《AWS Secrets Manager 用户指南》https://docs.aws.amazon.com/secretsmanager/latest/userguide/intro.html。

Secrets Manager 使用密 AWS Key Management Service 钥对机密进行加密。有关更多信息，请参阅 Secret encryption and decryption in AWS Key Management Service。

的托管 AWS IoT Device Management 集成与集成， AWS Secrets Manager 因此您可以将数据存储在 Secrets Manager 中，并在配置中使用密钥 ID。

托管集成如何使用机密

Open Authorization (OAuth) 是委托访问授权的开放标准，允许用户在不共享密码的情况下授予网站或应用程序访问其在其他网站上的信息的权限。这是第三方应用程序代表用户访问用户数据的安全方式，为共享密码提供了一种更安全的替代方案。

在中 OAuth，客户端 ID 和客户端密钥是在客户端应用程序请求访问令牌时对其进行识别和身份验证的凭证。

托管集成，用于 OAuth 与 AWS IoT Device Management 使用 C2C 工作流程的客户进行沟通。客户需要提供客户端 ID 和客户端密钥才能进行通信。托管集成客户将在其 AWS 账户中存储客户端 ID 和客户密钥，托管集成会读取我们客户账户中的客户端 ID 和客户机密。

如何创建密钥

要创建密钥，请按照《 AWS Secrets Manager 用户指南》中创建 AWS Secrets Manager 密钥中的步骤进行操作。

您必须使用客户管理的密钥创建您的密 AWS KMS 钥，以便进行托管集成，才能读取密钥值。有关更多信息，请参阅《 AWS Secrets Manager 用户指南》中的 AWS KMS 密钥权限。

您还必须使用下一节中的 IAM 策略。

授予托管集成的访问权限 AWS IoT Device Management 以检索密钥

要允许托管集成从 Secrets Manager 检索密钥值，请在创建密钥时在密钥的资源策略中加入以下权限。

```
{
  "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"

        }
      }
  } ]
}
```

将以下声明添加到您的客户管理 AWS KMS 密钥的策略中。

```
{
    "Version": "2012-10-17",
    "Statement": [{
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt",
                "kms:DescribeKey"
            ],
            "Principal": {
                "Service": [
                    "iotmanagedintegrations.amazonaws.com"
                ]
            },
            "Resource": [
```

```
                "arn:aws:kms:AWS Region:account-id:key/*"
            ]
        }

    ]
}
```

托管集成的合规性验证

要了解是否属于特定合规计划的范围，请参阅AWS 服务 “按合规计划划分的范围” ”，然后选择您感兴趣的合规计划。 AWS 服务 有关一般信息，请参阅AWS 合规计划AWS。

您可以使用下载第三方审计报告 AWS Artifact。有关更多信息，请参阅中的 “下载报告” 中的 “ AWS Artifact。

您在使用 AWS 服务 时的合规责任取决于您的数据的敏感性、贵公司的合规目标以及适用的法律和法规。 AWS 提供了以下资源来帮助实现合规性：

- Security Compliance & Governance：这些解决方案实施指南讨论了架构考虑因素，并提供了部署安全性和合规性功能的步骤。
- 符合 HIPAA 要求的服务参考：列出符合 HIPAA 要求的服务。并非所有 AWS 服务 人都符合 HIPAA 资格。
- AWS 合规资源AWS — 此工作簿和指南集可能适用于您所在的行业和所在地区。
- AWS 客户合规指南 — 从合规角度了解责任共担模式。这些指南总结了保护的最佳实践， AWS 服务并将指南映射到跨多个框架（包括美国国家标准与技术研究院 (NIST)、支付卡行业安全标准委员会 (PCI) 和国际标准化组织 (ISO)）的安全控制。
- 使用AWS Config 开发人员指南中的规则评估资源 — 该 AWS Config 服务评估您的资源配置在多大程度上符合内部实践、行业准则和法规。
- AWS Security Hub— 这 AWS 服务 可以全面了解您的安全状态 AWS。Security Hub 通过安全控制措施评估您的 AWS 资源并检查其是否符合安全行业标准和最佳实践。有关受支持服务及控制措施的列表，请参阅 Security Hub 控制措施参考。
- Amazon GuardDuty — 它通过监控您的 AWS 账户环境中是否存在可疑和恶意活动，来 AWS 服务检测您的工作负载、容器和数据面临的潜在威胁。 GuardDuty 通过满足某些合规性框架规定的入侵检测要求，可以帮助您满足各种合规性要求，例如 PCI DSS。
- AWS Audit Manager— 这 AWS 服务 可以帮助您持续审计 AWS 使用情况，从而简化风险管理以及对法规和行业标准的合规性。

使用与接口 VPC 终端节点的托管集成

您可以通过创建接口 Amazon VPC 终端节点在您的 Amazon VPC 和 AWS IoT 托管集成之间建立私有连接。接口端点由一项技术提供支持 AWS PrivateLink，该技术使您能够使用私有 IP 地址私密访问服务。AW PrivateLink S 将您的 VPC 和物联网托管集成之间的所有网络流量限制在亚马逊网络上。您不需要互联网网关、NAT 设备或 VPN 连接。

您无需使用 AWS PrivateLink，但建议使用。有关 AWS PrivateLink 和 VPC 终端节点的更多信息，请参阅AWS PrivateLink 指南 AWS PrivateLink中的通过访问 AWS 服务。

主题

AWS IoT 托管集成 VPC 终端节点的注意事项

在为 AWS IoT 托管集成设置接口 VPC 终端节点之前，请查看AWS PrivateLink 指南中的接口终端节点属性和限制。

AWS IoT 托管集成支持通过接口 VPC 终端节点从您的 VPC 调用其所有 API 操作。

支持的终端节点

AWS IoT 托管集成支持以下服务接口的 VPC 终端节点：

- 控制平面 API：`com.amazonaws.region.iotmanagedintegrations.api`

不支持的终端节点

以下 AWS IoT 托管集成终端节点不支持 VPC 终端节点：

- MQTT 端点：MQTT 设备通常部署在终端用户环境中，而不是部署在终端用户环境中 AWS VPCs，因此无需 AWS PrivateLink 集成。

- OAuth 回调端点：许多第三方平台不在 AWS 基础设施内运行，这降低了 AWS PrivateLink 支持 OAuth 流程的好处。

可用性

AWS IoT 托管集成 VPC 终端节点可在以下 AWS 区域使用：

- 加拿大（中部）– ca-central-1
- 欧洲（爱尔兰）– eu-west-1

随着 AWS IoT 托管集成的可用性不断扩大，将支持其他区域。

双栈支持

AWS IoT 托管集成 VPC 终端节点同时支持 IPv4 和 IPv6 流量。您可以使用以下 IP 地址类型创建 VPC 终端节点：

- IPv4: 为端点网络接口分配 IPv4 地址
- IPv6：为端点网络接口分配 IPv6 地址（ IPv6仅需要子网）
- Dualstack：为端点网络接口分配 IPv4 和 IPv6 地址

为 AWS IoT 托管集成创建接口 VPC 终端节点

您可以使用 Amazon VPC 控制台或 AWS CLI (AWS CLI) 为 AWS IoT 托管集成服务创建 VPC 终端节点。

为 AWS IoT 托管集成创建接口 VPC 终端节点（控制台）

1. 在亚马逊 VPC 控制台上打开亚马逊 VPC 控制台。
2. 在导航窗格中，选择端点。
3. 选择 创建端点。
4. 对于服务类别，选择 AWS 服务。
5. 在服务名称中，选择与您所在 AWS 地区对应的服务名称。例如：
   - `com.amazonaws.ca-central-1.iotmanagedintegrations.api`
   - `com.amazonaws.eu-west-1.iotmanagedintegrations.api`
6. 对于 VPC，请选择您要从中访问 AWS IoT 托管集成的 VPC。

7. 对于其他设置，默认情况下选中 “启用 DNS 名称”。我们建议您保留此设置。这样可以确保对 AWS IoT 托管集成公共服务终端节点的请求解析到您的 Amazon VPC 终端节点。
8. 对于子网，选择要在其中创建端点网络接口的子网。您可以为每个可用区选择一个子网。
9. 对于 IP address type（IP 地址类型），可从以下选项中进行选择：
   - IPv4: 为端点网络接口分配 IPv4 地址
   - IPv6：为端点网络接口分配 IPv6 地址（仅当所有选定的子网均为 IPv6-only 时才支持）
   - Dualstack：为端点网络接口分配 IPv4 和 IPv6 地址
10. 对于 Security groups（安全组），选择要与端点网络接口关联的安全组。安全组规则必须允许终端节点网络接口与您的 VPC 中与服务通信的资源之间进行通信。
11. 对于策略，选择完全访问权限以允许所有委托人通过接口终端节点对所有资源进行所有操作。要限制访问，请选择 “自定义” 并指定策略。
12. （可选）若要添加标签，请选择 Add new tag（添加新标签），然后输入该标签的键和值。
13. 选择创建端点。

为物联网托管集成 (AWS CLI) 创建接口 VPC 终端节点

使用create-vpc-endpoint命令并指定 VPC ID、VPC 终端节点类型（接口）、服务名称、将使用终端节点的子网以及与终端节点网络接口关联的安全组。

```
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
```

测试您的 VPC 终端节点

创建 VPC 终端节点后，您可以通过从 VPC 中的 EC2 实例对 AWS IoT 托管集成发出 API 调用来测试连接。

先决条件

- 您的 VPC 内私有子网中的 EC2 实例
- AWS IoT 托管集成操作的相应 IAM 权限

- 允许 HTTPS 流量（端口 443）到达 VPC 终端节点的安全组规则

测试 连接

1. 在私有子网中连接到您的 Amazon EC2 实例。
2. 验证私有 DNS 名称的 DNS 解析：

```
dig api.iotmanagedintegrations.region.api.aws
```

3. 测试 HTTPS 连接：

```
curl -v https://api.iotmanagedintegrations.region.api.aws
```

4. 调用 AWS IoT 托管集成 API：

```
aws iot-managed-integrations list-destinations \
  --region region \
  --endpoint-url https://api.iotmanagedintegrations.region.api.aws
```

`region`替换为您 AWS 所在的地区（例如，`ca-central-1`）。

控制通过 VPC 终端节点访问服务

VPC 终端节点策略是您在创建或修改接口 VPC 终端节点时附加到接口 VPC 终端节点的 IAM 资源策略。如果在创建端点时未附加策略，我们将为您附加默认策略以允许对服务进行完全访问。端点策略不会覆盖或替换 IAM 用户策略或服务特定的策略。这是一个单独的策略，用于控制从端点中对指定服务进行的访问。

端点策略必须采用 JSON 格式编写。有关更多信息，请参阅 Amazon VPC 用户指南中的使用 VPC 终端节点控制对服务的访问。

示例： AWS IoT 托管集成操作的 VPC 终端节点策略

以下是 AWS IoT 托管集成的端点策略示例。此策略允许通过 VPC 终端节点连接到 AWS IoT 托管集成的用户访问目标，但拒绝访问凭证柜。

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

示例：限制访问特定 IAM 角色的 VPC 终端节点策略

以下 VPC 终端节点策略仅允许在其信任链中具有指定 IAM 角色的 IAM 委托人访问 AWS IoT 托管集成。所有其他 IAM 委托人均被拒绝访问。

```
{
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "*",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:PrincipalArn": "arn:aws:iam::123456789012:role/
IoTManagedIntegrationsVPCRole"
                }
```

```
            }
        }
    ]
}
```

定价

创建和使用带有 AWS IoT 托管集成的接口 VPC 终端节点需要按标准费率付费。有关更多信息，请参阅AWS PrivateLink 定价。

限制

- 该 CreateAccountAssociationAPI旨在 OAuth 与第三方云服务一起运行，这需要请求离开Amazon网络。这对于使用 AWS PrivateLink 在 VPC 内控制流量的客户来说非常重要，因为 AWS PrivateLink 无法完全 end-to-end控制此 API 调用。
- AWS IoT 托管集成的 VPC 终端节点在中不可用。 AWS GovCloud (US) Regions

有关 VPC 终端节点的一般限制，请参阅 Amazon VPC 用户指南中的接口终端节点属性和限制。

连接到 AWS IoT Device Management FIPS 端点的托管集成

AWS IoT 提供了支持联邦信息处理标准 (FIPS) 140 -2 的控制平面端点。符合 FIPS 标准的端点与标准 AWS 端点不同。要以符合 FIPS 的方式与托管集成进行 AWS IoT Device Management 交互，您必须将下述端点与兼容 FIPS 的客户端一起使用。该 AWS IoT 主机不兼容 FIPS。

以下各节介绍如何使用 REST API、软件开发工具包或访问符合 FIPS 标准的 AWS IoT 终端节点。 AWS CLI

控制面板端点

支持托管集成操作的 FIPS 兼容控制平面端点及其相关 AWS CLI 命令列在按服务划分的 FIPS 端点中。在 FIPS 按服务划分的终端节点中，找到 AWS IoT Device Management -托管集成 > 服务，然后查找您的终端节点。 AWS 区域

要在访问托管集成操作时使用符合 FIPS 标准的终端节点，请使用适合您的端点的 AWS SDK 或 REST API。 AWS 区域

要在运行托管集成 CLI 命令时使用符合 FIPS 的终端节点，请在命令中添加带有相应终端节点的--endpoint参数。 AWS 区域

监控托管集成

监控是维护托管集成和您的其他 AWS 解决方案的可靠性、可用性和性能的重要组成部分。AWS 提供以下监控工具，用于监视托管集成，在出现问题时进行报告，并在适当时自动采取措施：

- AWS CloudTrail捕获由您的账户或代表您的 AWS 账户进行的 API 调用和相关事件，并将日志文件传输到您指定的 Amazon S3 存储桶。您可以识别哪些用户和账户拨打了电话 AWS、发出呼叫的源 IP 地址以及呼叫发生的时间。有关更多信息，请参阅 AWS CloudTrail 用户指南。

使用记录托管集成 API 调用 AWS CloudTrail

托管集成与AWS CloudTrail一项服务集成，该服务提供用户、角色或角色所执行操作的 AWS 服务记录。 CloudTrail 将托管集成的所有 API 调用捕获为事件。捕获的调用包括来自托管集成控制台的调用以及对托管集成 API 操作的代码调用。使用收集的信息 CloudTrail，您可以确定向托管集成发出的请求、发出请求的 IP 地址、发出请求的时间以及其他详细信息。

每个事件或日志条目都包含有关生成请求的人员信息。身份信息有助于您确定以下内容：

- 请求是使用根用户凭证还是用户凭证发出的。
- 请求是否代表 IAM Identity Center 用户发出。
- 请求是使用角色还是联合用户的临时安全凭证发出的。
- 请求是否由其他 AWS 服务发出。

CloudTrail 在您创建账户 AWS 账户 时在您的账户中处于活动状态，并且您自动可以访问 CloudTrail 活动历史记录。 CloudTrail 事件历史记录提供了过去 90 天中记录的管理事件的可查看、可搜索、可下载且不可变的记录。 AWS 区域有关更多信息，请参阅《AWS CloudTrail 用户指南》中的 “使用 CloudTrail 事件历史记录”。查看活动历史记录不 CloudTrail收取任何费用。

要持续记录 AWS 账户 过去 90 天内的事件，请创建跟踪或 CloudTrailLake 事件数据存储。

CloudTrail 步道

跟踪允许 CloudTrail 将日志文件传输到 Amazon S3 存储桶。使用创建的所有跟踪 AWS Management Console 都是多区域的。您可以通过使用 AWS CLI创建单区域或多区域跟踪。建议创建多区域跟踪，因为您可以捕获账户 AWS 区域 中的所有活动。如果您创建单区域跟踪，则只能查看跟踪的 AWS 区域中记录的事件。有关跟踪的更多信息，请参阅《AWS CloudTrail 用户指南》中的为您的 AWS 账户创建跟踪和为组织创建跟踪。

通过创建跟踪，您可以免费将正在进行的管理事件的一份副本传送到您的 Amazon S3 存储桶，但是，会收取 Amazon S3 存储费用。 CloudTrail 有关 CloudTrail 定价的更多信息，请参阅AWS CloudTrail 定价。有关 Amazon S3 定价的信息，请参阅 Amazon S3 定价。

CloudTrail 湖泊事件数据存储

CloudTrail L@@ ak e 允许你对自己的事件运行基于 SQL 的查询。 CloudTrail Lake 将基于行的 JSON 格式的现有事件转换为 Apache ORC 格式。ORC 是一种针对快速检索数据进行优化的列式存储格式。事件将被聚合到事件数据存储中，它是基于您通过应用高级事件选择器选择的条件的不可变的事件集合。应用于事件数据存储的选择器用于控制哪些事件持续存在并可供您查询。有关 CloudTrail Lake 的更多信息，请参阅《AWS CloudTrail 用户指南》中的 “使用 AWS CloudTrail Lake”。

CloudTrail 湖泊事件数据存储和查询会产生费用。创建事件数据存储时，您可以选择要用于事件数据存储的定价选项。定价选项决定了摄取和存储事件的成本，以及事件数据存储的默认和最长保留期。有关 CloudTrail 定价的更多信息，请参阅AWS CloudTrail 定价。

中的管理活动 CloudTrail

管理事件提供有关对中的资源执行的管理操作的信息 AWS 账户。这些也称为控制面板操作。默认情况下， CloudTrail 记录管理事件。

托管集成将以下托管集成控制平面操作记录 CloudTrail 为管理事件。

- `CreateCloudConnector`
- `UpdateCloudConnector`
- `GetCloudConnector`
- `DeleteCloudConnector`
- `ListCloudConnectors`
- `CreateConnectorDestination`
- `UpdateConnectorDestination`
- `GetConnectorDestination`
- `DeleteConnectorDestination`
- `ListConnectorDestinations`
- `CreateAccountAssociation`
- `UpdateAccountAssociation`

- GetAccountAssociation
- DeleteAccountAssociation
- ListAccountAssociations
- StartAccountAssociationRefresh
- ListManagedThingAccountAssociations
- RegisterAccountAssociation
- DeregisterAccountAssociation
- SendConnectorEvent
- ListDeviceDiscoveries
- ListDiscoveredDevices

事件示例

事件代表来自任何来源的单个请求，包括有关所请求的 API 操作、操作的日期和时间、请求参数等的信息。 CloudTrail 日志文件不是公共 API 调用的有序堆栈跟踪，因此事件不会按任何特定顺序出现。

以下示例显示了一个演示 CreateCloudConnector API 操作成功 CloudTrail 的事件。

CreateCloudConnectorAPI 操作成功 CloudTrail 事件。


```
{
    "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"
}
```

以下示例显示了一个演示 ListDiscoveredDevices API 操作成功 CloudTrail 的事件。

ListDiscoveredDevicesAPI 操作成功 CloudTrail 事件。


```
{
    "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"
```

```
}
```

有关 CloudTrail 录音内容的信息，请参阅《AWS CloudTrail 用户指南》中的CloudTrail录制内容。

托管集成的文档历史记录《开发者指南》

下表描述了托管集成的文档版本。

变更	说明	日期
公开发行版本	托管集成开发者指南正式发布版	2025 年 6 月 25 日
初始预览版	托管集成开发者指南的初始预览版	2025 年 3 月 3 日

本文属于机器翻译版本。若本译文内容与英语原文存在差异，则一律以英文原文为准。