aws

開發人員指南

的受管整合 AWS IoT Device Management

的受管整合 AWS IoT Device Management: 開發人員指南

Table of Contents

什麼是 受管整合 AWS IoT Device Management？

的受管整合 AWS IoT Device Management 可協助 IoT 解決方案供應商統一數百個製造商對 IoT 裝置的控制和管理。您可以使用受管整合來自動化裝置設定工作流程，並支援許多裝置的互通性，無論裝置廠商或連線通訊協定為何。透過 受管整合，您可以使用單一使用者介面和一組 APIs 來控制、管理和操作各種裝置。

主題

支援的地區

下列 區域 AWS IoT Device Management 支援 的受管整合：

- 加拿大 (中部)
- 歐洲 (愛爾蘭)

您是第一次受管整合使用者嗎？

如果您是第一次使用 受管整合，建議您先閱讀以下章節：

- 設定受管整合
- 開始使用 的 受管整合 AWS IoT Device Management

受管整合概觀

下圖提供受管整合的高階概觀

受管整合術語

在受管整合中，有許多對於管理您自己的裝置實作至關重要的概念和術語。下列各節概述了這些關鍵概念和術語，以便更好地了解 受管整合。

一般受管整合術語

與 AWS IoT Core 物件相比，受管物件是了解受管整合的重要概念。

- AWS IoT Core 物件： AWS IoT Core 物件是一種提供數位表示法的 AWS IoT Core 建構。開發人員應管理政策、資料儲存、規則、動作、MQTT 主題，以及將裝置狀態交付至資料儲存體。如需物件的詳細資訊 AWS IoT Core ，請參閱使用 管理裝置 AWS IoT。
- 受管整合 受管物件：透過受管物件，我們提供簡化裝置互動的抽象概念，不需要開發人員建立規則、動作、MQTT 主題和政策等項目。

Cloud-to-cloud術語

與受管整合整合整合整合的實體裝置可能來自第三方雲端供應商。若要將這些裝置加入受管整合並與第三方雲端供應商通訊，下列術語涵蓋支援這些工作流程的一些重要概念：

- Cloud-to-cloud(C2C) 連接器：C2C 連接器會在受管整合與第三方雲端提供者之間建立連線。
- 第三方雲端提供者：對於在受管整合之外製造和管理的裝置，第三方雲端提供者可為最終使用者控制這些裝置，且受管整合會針對裝置命令等各種工作流程與第三方雲端提供者通訊。

資料模型術語

受管整合使用資料模型來組織資料，並在裝置之間end-to-end通訊。下列術語涵蓋了解這兩個資料模型的一些重要概念：

- 裝置：代表實體裝置的實體（例如影片門鈴），其具有多個節點一起運作以提供完整的功能集。
- 端點：端點封裝獨立功能（響鈴、動作偵測、影片門鈴照明）。
- 功能：代表在端點中使用功能所需的元件的實體（視訊門鈴鈴鈴功能中的按鈕或燈光和鈴聲）。
- 動作：代表與裝置功能互動的實體（鈴鐺或檢視門口的人物）。
- 事件：代表來自裝置功能的事件的實體。裝置可以傳送事件來報告事件/警示、來自感應器的活動等（例如門上有爆震/環）。
- 屬性：代表裝置狀態中特定屬性的實體（鈴聲響起、手電筒燈開啟、攝影機正在錄製）。

- 資料模型：資料層對應於有助於支援應用程式功能的資料和動詞元素。當有與裝置互動的意圖時，應用程式會在這些資料結構上運作。如需詳細資訊，請參閱 GitHub 網站上的 connectedhomeip。
- 結構描述：結構描述是以 JSON 格式呈現的資料模型。

設定受管整合

下列各節會引導您使用 受管整合的初始設定 AWS IoT Device Management。

主題

註冊 AWS 帳戶

如果您沒有 AWS 帳戶，請完成下列步驟來建立一個。

註冊 AWS 帳戶

1. 開啟 https://portal.aws.amazon.com/billing/signup。
2. 請遵循線上指示進行。

   部分註冊程序需接收來電或簡訊，並在電話鍵盤輸入驗證碼。

   當您註冊 時 AWS 帳戶，AWS 帳戶根使用者會建立 。根使用者有權存取該帳戶中的所有 AWS 服務 和資源。作為安全最佳實務，請將管理存取權指派給使用者，並且僅使用根使用者來執行需要根使用者存取權的任務。

AWS 會在註冊程序完成後傳送確認電子郵件給您。您可以隨時登錄 https://aws.amazon.com/ 並選擇我的帳戶，以檢視您目前的帳戶活動並管理帳戶。

建立具有管理存取權的使用者

註冊 後 AWS 帳戶，請保護 AWS 帳戶根使用者、啟用 AWS IAM Identity Center和建立管理使用者，以免將根使用者用於日常任務。

保護您的 AWS 帳戶根使用者

1. 選擇根使用者並輸入 AWS 帳戶 您的電子郵件地址，以帳戶擁有者AWS 管理主控台身分登入 。在下一頁中，輸入您的密碼。

   如需使用根使用者登入的說明，請參閱 AWS 登入 使用者指南中的以根使用者身分登入。

2. 若要在您的根使用者帳戶上啟用多重要素驗證 (MFA)。

   如需說明，請參閱《IAM 使用者指南》中的為您的 AWS 帳戶 根使用者 （主控台） 啟用虛擬 MFA 裝置。

建立具有管理存取權的使用者

1. 啟用 IAM Identity Center。

   如需指示，請參閱《AWS IAM Identity Center 使用者指南》中的啟用 AWS IAM Identity Center。

2. 在 IAM Identity Center 中，將管理存取權授予使用者。

   如需使用 IAM Identity Center 目錄 做為身分來源的教學課程，請參閱AWS IAM Identity Center 《 使用者指南》中的使用預設值設定使用者存取 IAM Identity Center 目錄。

以具有管理存取權的使用者身分登入

- 若要使用您的 IAM Identity Center 使用者簽署，請使用建立 IAM Identity Center 使用者時傳送至您電子郵件地址的簽署 URL。

  如需使用 IAM Identity Center 使用者登入的說明，請參閱AWS 登入 《 使用者指南》中的登入 AWS 存取入口網站。

指派存取權給其他使用者

1. 在 IAM Identity Center 中，建立一個許可集來遵循套用最低權限的最佳實務。

   如需指示，請參閱《AWS IAM Identity Center 使用者指南》中的建立許可集。

2. 將使用者指派至群組，然後對該群組指派單一登入存取權。

   如需指示，請參閱《AWS IAM Identity Center 使用者指南》中的新增群組。

開始使用 的 受管整合 AWS IoT Device Management

下列各節概述開始使用 受管整合所需的步驟。

主題

裝置類型

受管整合可管理許多類型的裝置。每個裝置都屬於下列三個類別之一：

- 直接連線裝置：這種類型的裝置會直接連線至 受管整合端點。一般而言，這些裝置是由裝置製造商建置和管理，其中包含用於直接連線的受管整合終端裝置 SDK。
- 中樞連線裝置：這些裝置透過執行受管整合中樞 SDK 的中樞連線至受管整合，而中樞 SDK 可管理裝置探索、加入和控制功能。最終使用者可以使用按鈕啟動或條碼掃描來加入這些裝置。

  下列兩個工作流程支援加入中樞連線裝置：

  - 最終使用者啟動的按鈕按下以開始裝置探索
  - 以條碼為基礎的掃描，以執行裝置關聯
- Cloud-to-cloud(C2C) 裝置：這些裝置是由維護自己的雲端基礎設施和品牌行動應用程式以進行裝置控制的廠商所設計和管理。受管整合客戶可以存取預先建置的 C2C 連接器目錄或自行建立，透過統一的界面開發與多個第三方供應商雲端搭配使用的 IoT 解決方案。

  當最終使用者第一次開啟 C2C 裝置的電源時，必須佈建其個別的第三方雲端提供者，以進行受管整合，以取得其裝置功能和中繼資料。完成該佈建工作流程後，受管整合可以代表最終使用者與雲端裝置和第三方雲端供應商通訊。

> **Note**
>
> 中樞不是上述的特定裝置類型。其目的是做為智慧家庭裝置的控制者，並促進受管整合與第三方雲端供應商之間的連線。它可以充當上述裝置類型和中樞。

設定加密金鑰

安全性對於最終使用者、受管整合和第三方雲端之間路由的資料至關重要。我們支援保護您裝置資料的其中一種方法是利用安全加密金鑰end-to-end加密，以路由您的資料。

身為受管整合的客戶，您有下列兩個選項可使用加密金鑰：

- 使用預設的受管整合受管加密金鑰。
- 提供您建立 AWS KMS key 的 。

如需 AWS KMS 服務的詳細資訊，請參閱金鑰管理服務 (KMS)

《 受管整合 API 參考指南》中的呼叫 PutDefaultEncryptionConfiguration API，可讓您存取以更新要使用的加密金鑰選項。 根據預設，受管整合會使用預設的受管整合受管加密金鑰。您可以隨時使用 PutDefaultEncryptionConfiguration API 更新加密金鑰組態。

此外，呼叫 GetDefaultEncryptionConfiguration API 命令會傳回預設或指定區域中 AWS 帳戶加密組態的相關資訊。

加入技巧

下列是加入的類型：

直接連線的裝置加入

如需加入直接連線裝置的步驟，佈建者請參閱 。

Hub 加入

如需加入中樞的步驟，將您的中樞加入受管整合請參閱 。

中樞連線裝置加入

如需加入中樞連線裝置的步驟，加入裝置並在中樞中操作它們請參閱 。

Cloud-to-cloud裝置加入

使用 C2C Cloud-to-Cloud) 連接器 如需從第三方雲端廠商將雲端裝置加入受管整合的步驟，請參閱 。

裝置佈建

裝置佈建可促進裝置加入程序、監督整個裝置生命週期，並為受管整合的其他層面可存取的裝置資訊建立集中式儲存庫。受管整合提供統一的界面來管理各種裝置類型，容納透過裝置軟體開發套件 (SDK) 直接連線的第一方客戶裝置，或間接透過中樞裝置連結commercial-off-the-shelf(COTS) 裝置。

無論裝置類型為何，受管整合中的每個裝置都有一個全域唯一識別符，稱為 `managedThingId`。此識別符用於整個裝置生命週期的裝置加入和管理。它由受管整合進行完全管理，並且在所有受管整合中，該特定裝置是唯一的 AWS 區域。當裝置最初新增至受管整合時，會建立此識別符，並連接到受管整合中的受管物件。受管物件是受管整合中實體裝置的數位表示法，以鏡射實體裝置的所有裝置中繼資料。對於第三方裝置，除了`managedThingId`存放在代表實體裝置的受管整合中之外，他們還可能有自己的獨立唯一識別符，專用於第三方雲端。

佈建的裝置可能具有不同的狀態，取決於其加入流程的階段。下列清單說明每個佈建狀態：

- ACTIVATED：找到裝置，且可使用命令和控制項。
- 已探索：找到裝置，但尚未提供命令和控制項。
- UNASSOCIATED：已建立受管物件，但需要探索進一步的動作。無法從 AWS 雲端 或 AWS IoT 受管整合控制器 （ 中樞 ） 連線
- PRE_ASSOCIATED：已建立受管物件，一旦開啟電源或連線，即可自動探索。無法從 AWS 雲端 或 AWS IoT 受管整合控制器 （ 中樞 ） 連線。
- DELETE_IN_PROGRESS：非同步刪除程序已啟動。
- 已刪除：裝置已從 中刪除 AWS 雲端。
- ISOLATED：先前探索或啟用且無法再連線的受管物件。例如，第三方雲端的裝置，其連接器關聯皆已刪除。

下列加入流程用於佈建具有受管整合的中樞：

將您的中樞加入受管整合：設定核心佈建器和通訊協定特定的外掛程式，這些外掛程式可共同處理裝置身分驗證、通訊和設定。

提供下列入門流程，用於佈建具有受管整合的中樞連線裝置：

- 簡單設定 (SS)：最終使用者開啟 IoT 裝置的電源，並使用裝置製造商應用程式掃描其 QR 碼。然後，裝置會註冊到受管整合雲端，並連接到 IoT 中樞。

- 零接觸設定 (ZTS)：裝置在供應鏈中已預先關聯上游。例如，此步驟會提早完成，以預先將裝置連結至客戶帳戶，而不是掃描裝置 QR 碼的最終使用者。
- 使用者引導設定 (UGS)：最終使用者會開啟裝置的電源，並遵循互動式步驟將其加入受管整合。這可能包括按下 IoT 中樞上的按鈕、使用裝置製造商應用程式，或同時按下中樞和裝置上的按鈕。如果簡易設定失敗，您可以使用此方法。

> **Note**
>
> 受管整合中的裝置佈建工作流程與裝置的加入需求無關。受管整合提供簡化的使用者介面來加入和管理裝置，無論裝置類型或裝置通訊協定為何。

裝置和裝置設定檔生命週期

管理裝置和裝置設定檔的生命週期可確保您的裝置機群安全且有效率地執行。

主題

裝置

在初始加入期間，受管整合會建立實體裝置的數位孿生，稱為受管物件。受管物件的 managedThingID提供全域唯一識別符，以識別所有區域的受管整合中的裝置。裝置會在佈建期間與本機中樞配對，以便與第三方裝置的受管整合或第三方雲端進行即時通訊。裝置也會與擁有者建立關聯，如 等受管物件的公有 APIs 中的 owner 參數所識別GetManagedThing。裝置會根據裝置類型連結至對應的裝置設定檔。

> **Note**
>
> 如果實體裝置在不同客戶下佈建多次，則可能有多個記錄。

裝置生命週期從使用 CreateManagedThing API 在受管整合中建立受管物件開始，並在客戶使用 DeleteManagedThing API 刪除受管物件時結束。裝置生命週期由下列公有 APIs管理：

- CreateManagedThing
- ListManagedThings
- GetManagedThing
- UpdateManagedThing
- DeleteManagedThing

裝置設定檔

裝置設定檔代表特定類型的裝置，例如燈泡或門鈴。它與製造商相關聯，並包含裝置的功能。裝置設定檔會存放具有受管整合之裝置連線設定請求所需的身分驗證資料。使用的身分驗證資料是裝置條碼。

在裝置製造過程中，製造商可以向受管整合註冊其裝置設定檔。這可讓製造商在加入和佈建工作流程期間，從受管整合取得裝置的必要資料。裝置描述檔的中繼資料會存放在實體裝置上，或列印在裝置標籤上。裝置描述檔的生命週期會在製造商在受管整合中將其刪除時結束。

資料模型

資料模型代表如何在系統中組織資料的組織階層。此外，它支援整個裝置實作的end-to-end通訊。對於受管整合，使用兩種資料模型。受管整合資料模型和事項資料模型的 AWS 實作。它們具有相似性，但也有以下主題中概述的細微差異。

對於第三方裝置，這兩種資料模型都用於最終使用者、受管整合和第三方雲端提供者之間的通訊。若要從兩個資料模型轉譯裝置命令和裝置事件等訊息，會利用 Cloud-to-Cloud Connector 功能。

主題

受管整合資料模型

受管整合資料模型會管理最終使用者與受管整合之間的所有通訊。

裝置階層

endpoint 和 capability資料元素用於描述受管整合資料模型中的裝置。

endpoint

endpoint 代表 功能提供的邏輯界面或服務。

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

Capability

capability 代表裝置功能。

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

對於capability資料元素，有三個項目構成該項目：property、 action和 event。它們可用來與裝置互動和監控裝置。

- 屬性：裝置保留的狀態，例如可調光光源的目前亮度屬性。
  - ```
    {
        "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

    }
    ```
- 動作：可能執行的任務，例如鎖定門鎖上的門。動作可能會產生回應和結果。
  - ```
    {
        "name": { "$ref": "/schema-versions/definition/aws.name@1.0" }, //required
        "parameters": Map<String name, JSONNode value>,
        "responseCode": HTTPResponseCode,
        "errors": {
            "code": "string",
            "message": "string"
        }
    }
    ```
- 事件：基本上是過去狀態轉換的記錄。雖然 property代表目前狀態，但事件是過去的日誌，並包含單調增加的計數器、時間戳記和優先順序。它們可以擷取狀態轉換，以及 無法立即實現的資料建模property。
  - ```
    {
        "name": { "$ref": "/schema-versions/definition/aws.name@1.0" },        //
    required
        "parameters": Map<String name, JSONNode value>
    }
    ```
    ```

AWS 實作事項資料模型

事項資料模型的 AWS 實作會管理受管整合和第三方雲端提供者之間的所有通訊。

如需詳細資訊，請參閱事項資料模型：開發人員資源。

裝置階層

有兩種資料元素可用來描述裝置： endpoint和 cluster。

endpoint

endpoint 代表 功能提供的邏輯界面或服務。

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

cluster

cluster 代表裝置功能。

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

對於cluster資料元素，有三個項目構成該項目：attribute、 command和 event。它們可用來與裝置互動和監控裝置。

- 屬性：裝置保留的狀態，例如可調光光源的目前亮度屬性。
  - ```
    {
        "id" (hexadecimalString): (JsonNode) value
    }
    ```
- 命令：可執行的任務，例如鎖定門鎖上的門。命令可能會產生回應和結果。
  - ```
    "id": {
    ```

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

- 事件：基本上是過去狀態轉換的記錄。雖然 attributes代表目前狀態，但事件是過去的日誌，並包含單調增加的計數器、時間戳記和優先順序。它們可以擷取狀態轉換，以及使用 無法立即實現的資料建模attributes。
  - ```
    "id": {
        "fieldId": "fieldValue",
        ...
    }
    ```

資料模型結構描述

受管整合支援兩種結構描述類型：功能和類型定義。如果您要建立自訂資料模型，您可以使用 JSON 結構描述文件來定義任一類型的結構描述。每個結構描述文件的限制為 50，000 個字元。

功能結構描述

功能是代表端點內特定功能的基本建置區塊。透過 功能，您可以使用屬性、動作和事件來建立裝置狀態和行為的模型。屬性可讓您彈性地使用任何宣告資料類型建立裝置狀態屬性的模型。動作和事件會建立裝置行為的模型，包括可以執行的命令，以及可以報告的訊號。

以下顯示功能結構描述的高階結構。

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

動作

代表與裝置功能互動的實體。例如，響鈴或檢視誰在門口。
```

事件

代表裝置功能事件的實體。裝置可以傳送事件，以報告來自感應器的事件、警示或活動，例如撞門。

屬性

代表裝置狀態中特定屬性的實體。例如，鈴聲正在響起或門廊燈已開啟

每個功能都包含唯一的命名空間識別符、版本資訊及其用途的描述。結構描述文件使用語意版本控制來維持回溯相容性，同時啟用新功能。

如需詳細資訊，請參閱功能定義的結構描述。

類型定義結構描述

類型定義是一種宣告式結構化資料類型，可實現可重複使用性和可編寫性。它定義了如何格式化和限制資訊。使用類型定義來建立整個 IoT 解決方案的標準化資料格式。

每個類型定義包括：

- 唯一的命名空間識別符
- Title
- 描述
- 定義資料格式和限制條件的屬性

類型可以是簡單的基本概念，例如具有定義限制的整數或字串，或是複雜結構，例如列舉或具有多個欄位的自訂物件。類型定義使用 JSON 結構描述語法來指定限制條件，包括最小值和最大值、字串長度和允許的模式。

如需詳細資訊，請參閱類型定義的結構描述。

功能定義的結構描述

使用宣告式 JSON 文件來記錄功能，該文件提供明確的合約，說明該功能在系統中應如何運作。

對於 功能，以下至少一個區段中的強制元素為 `$id`、`extrinsicId`、 `nameextrinsicVersion`和至少一個元素：

- `properties`

- actions
- events

功能中的選用元素為 $ref、title、version、 description$defs和 extrinsicProperties。對於 功能， $ref 必須參考 aws.capability。

下列各節詳細說明用於功能定義的結構描述。

$id （強制性）

$id 元素可識別結構描述定義。它必須遵循此結構：

- 從 /schema-versions/ URI 字首開始
- 包含capability結構描述類型
- 使用正斜線 (/) 做為 URI 路徑分隔符號
- 包含結構描述身分，片段以句點 (.) 分隔
- 使用 @ 字元來分隔結構描述 ID 和版本
- 以 semver 版本結尾，使用句點 (.) 分隔版本片段

結構描述身分的開頭必須是長度為 3-12 個字元的根命名空間，後面接著選用的子命名空間和名稱。

轉換器版本包含 MAJOR 版本 （最多 3 位數）、MINOR 版本 （最多 3 位數） 和選用的 PATCH 版本 （最多 4 位數）。

> **Note**
>
> 您無法使用預留命名空間aws或 matter

Example範例 $id

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

$ref

$ref 元素參考系統中的現有功能。它遵循與 $id元素相同的限制條件。

> **Note**
>
> 類型定義或功能必須使用 `$ref` 檔案中提供的值存在。

Example範例 $ref

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

名稱 （必要）

名稱元素是字串，代表結構描述文件中的實體名稱。它通常包含縮寫，並且必須遵循這些規則：

- 僅包含英數字元、句點 (.)、斜線 (/)、連字號 (-) 和空格
- 從字母開始
- 最多 64 個字元

名稱元素用於 Amazon Web Services 主控台 UI 和 文件。

Example範例名稱

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

標題

標題元素是結構描述文件所代表實體的描述性字串。它可以包含任何字元，並用於文件中。功能標題的長度上限為 256 個字元。

Example範例標題

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

description

description 元素提供結構描述文件所代表實體的詳細說明。它可以包含任何字元，並用於文件中。功能描述的長度上限為 2048 個字元

Example範例描述

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

version

version 元素是選用項目。這是代表結構描述文件版本的字串。它有下列限制條件：

- 使用轉換器格式，下列版本片段以 .（期間）分隔。
  - MAJOR 版本，最多 3 位數
  - MINOR 版本，最多 3 位數
  - PATCH 版本（選用），最多 4 位數
- 長度可以介於 3 到 12 個字元之間。

Example範例版本

```
1.0
```

```
1.12
```

```
1.4.1
```

使用 功能版本

功能是不可變的版本化實體。任何變更都應該建立新的版本。系統會使用 MAJOR.MINOR.PATCH 格式的語意版本控制，其中：

- 進行回溯不相容 API 變更時，主要版本會增加
- 以回溯相容的方式新增功能時，MINOR 版本會增加

- PATCH 版本會在 功能中進行次要且不影響的新增時增加。

從事項叢集衍生的功能是以 1.4 版為基礎，且每個事項版本預期都會匯入系統。由於Matter 版本同時使用主要和次要層級的轉換器，受管整合只能使用 PATCH 版本。

當您為事項新增 PATCH 版本時，請務必考量該事項使用循序修訂。所有 PATCH 版本都必須符合事項規格中記載的修訂，而且這些修訂必須回溯相容。

若要修正任何回溯不相容的問題，您必須與連線標準聯盟 (CSA) 合作，以解決規格中的問題，並發行新的修訂版本。

AWS受管功能已使用 的初始版本發行`1.0`。透過這些，可以使用這三個層級的版本。

extrinsicVersion （強制性）

這是代表 AWS IoT 系統外部受管版本的字串。對於事項功能， `extrinsicVersion` 映射到 `revision`

它以字串化整數值表示，長度可以是 1 到 10 個數字。

Example範例版本

```
7
```

```
1567
```

extrinsicId （強制性）

`extrinsicId` 元素代表在 Amazon Web Services IoT 系統外部管理的識別符。對於事項功能，它`fieldId`會根據內容對應至 `clusterId`、`eventId`、、 `attributeId` `commandId`或 。

`extrinsicId` 可以是字串化十進位整數 (1-10 位數） 或字串化十六進位整數 (0x 或 0X 字首，後面接著 1-8 個十六進位位數）。

> **Note**
>
> 對於 AWS，廠商 ID (VID) 為 0x1577，對於事項，則為 0。系統會確保自訂結構描述不會將這些預留 VIDs用於 功能。

Example extrinsicIds

```
0018
0x001A
0x15771002
```

$defs

`$defs` 區段是子結構描述的映射，可在 JSON 結構描述允許的結構描述文件中參考。在此映射中， 金鑰用於本機參考定義，而 值提供 JSON 結構描述。

> **Note**
>
> 系統只會強制執行 `$defs`是有效的映射，而且每個子結構描述都是有效的 JSON 結構描述。不會強制執行其他規則。

使用 定義時，請遵循下列限制條件：

- 在定義名稱中僅使用 URI 易記字元
- 確保每個值都是有效的子結構描述
- 包含符合結構描述文件大小限制的任何數量的子結構描述

extrinsicProperties

`extrinsicProperties` 元素包含一組在外部系統中定義的屬性，但在資料模型中維護。對於事項功能，它會映射到 ZCL 叢集內不同的未建模或部分建模元素、屬性、命令或事件。

外部屬性必須遵循下列限制：

- 屬性名稱必須為英數字元，不含空格或特殊字元
- 屬性值可以是任何 JSON 結構描述值
- 最多 20 個屬性

系統支援各種 `extrinsicProperties`，包括 `access`、`apiMaturity`、`cliFunctionName`、`cli`等。這些屬性有助於 ACL 轉換 AWS （反之亦然） 資料模型。

> **Note**
>
> 功能的 action、property、 event和 struct 欄位元素支援外部屬性，但功能或叢集本身不支援。

系統支援的外部屬性

系統會追蹤下列未建模或部分建模的叢集、屬性、命令或事件屬性，如同在往返 ZCL 的轉換extrinsicProperties期間：

access

每個存取物件都包含下列項目：

- op - 以 建模的操作，enum其值為：write、 read或 invoke
- privilege - 以 建模的權限enum，其值為：view、manage、、 proxy_view operate或 administer
- role - 代表運算子角色的未限制字串

apiMaturity

代表成熟度層級的無邊界純字串。這在 ZCL 中建模為 enum，其值為：stable、internal、provisional或 deprecated

side

模型化為具有值的列舉：server、 either和 client

布林值屬性

下列屬性是布林值旗標：

- isFabricScoped
- isFabricSensitive
- mustUseAtomicWrite
- mustUseTimedInvoke

字串屬性

下列屬性會以未繫結字串表示：

- cli

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

轉換考量

對於 ZCL 轉換， extrinsicProperties 會存放在地圖中，無需處理。使用 探索的自訂結構描述不會進行 ZCL 轉換。不過，如果您計劃在未來為自訂結構描述實作 ZCL 轉換，則必須為所有未限制的純字串類型建立模型，extrinsicProperties並定義限制，例如列舉、模式 (regex) 和長度。此準備可確保在轉換期間正確處理這些屬性。

相反地，對於 AWS 連接器轉換，完全extrinsicProperties不包含這些詳細資訊，因為連接器格式不需要這些詳細資訊。

屬性

屬性代表 功能的裝置受管狀態。每個狀態都定義為索引鍵/值對，其中索引鍵描述狀態的名稱，而值描述狀態的定義。

使用屬性時，請遵循下列限制條件：

- 在屬性名稱中僅使用英數字元，不含空格或特殊字元
- 包含符合結構描述文件大小限制的任何數量屬性

使用屬性

功能中的屬性是基本元素，代表由受管整合提供支援的裝置的特定狀態。它代表裝置目前的條件或組態。透過標準化這些屬性的定義和結構，智慧型家庭系統可確保來自不同製造商的裝置進行有效通訊，進而建立無縫且可互通的體驗。

對於功能屬性，強制性元素為 `extrinsicId`和 `value`。功能屬性中的選用元素為 `description`、`mutable`、 `retrievablereportable`和 `extrinsicProperties`。

Value

無限制的結構，可讓建置器放置任何 JSON 結構描述合規限制，以定義此屬性的資料類型。

定義值時，請遵循下列限制條件：

- 對於簡單類型，請使用 `type`和任何其他原生 JSON 結構描述限制，例如 `maxLength`或 `maximum`
- 對於複合類型，請使用 `oneOf`、 `allOf`或 `anyOf`。系統不支援 `not`關鍵字
- 若要參考任何全域類型，請使用 `$ref`搭配有效的可探索參考
- 如需 nullability，請遵循 OpenAPI 類型結構描述定義，方法是使用布林值旗標提供 nullable 屬性 (`true`如果 null 是允許的值）

範例：

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

可擷取

說明狀態是否可讀取的布林值。

狀態的可讀性方面會延遲至裝置的 功能實作。裝置會決定指定狀態是否可讀取。狀態的這個方面尚不支援在功能報告中報告，因此未在系統內強制執行。

範例： `true` 或 `false`

Mutable

說明狀態是否可寫入的布林值。

狀態的可寫入性方面會延遲至裝置的 功能實作。裝置會決定指定狀態是否可寫入。狀態的這個方面尚不支援在功能報告中報告，因此未在系統內強制執行。

範例： `true` 或 `false`

可報告

布林值，說明當狀態變更時，裝置是否報告狀態。

狀態的可報告性方面會延遲至裝置的 功能實作。裝置會決定指定狀態是否可報告。狀態的這個方面尚不支援在功能報告中報告，因此未在系統內強制執行。

範例： `true` 或 `false`

動作

動作是遵循請求回應模型的結構描述受管操作。每個動作代表裝置實作的操作。

實作動作時，請遵循下列限制：

- 在動作陣列中僅包含唯一動作
- 包含符合結構描述文件大小限制的任何數量動作

使用動作

動作是與受管整合系統中的裝置功能互動和控制其功能的標準化方式。它代表可在裝置上執行的特定命令或操作，並使用結構化格式來建立任何必要請求或回應參數的模型。這些動作可做為使用者意圖和裝置操作之間的橋樑，在不同類型的智慧型裝置上實現一致且可靠的控制。

對於 動作，必要元素為 `name`和 `extrinsicId`。選用元素為 `description`、`extrinsicPropertiesrequest`和 `response`。

描述

描述的長度限制上限為 1536 個字元。

請求

請求區段是選用的，如果沒有請求參數，則可以省略。如果省略，則系統支援傳送沒有任何承載的請求，只要使用 的名稱即可`Action`。這用於簡單的動作，例如開啟或關閉燈光。

複雜動作需要額外的參數。例如，串流攝影機影片的請求可能包含有關要使用的串流通訊協定或是否將串流傳送至特定顯示裝置的參數。

對於動作請求，強制性元素為 `parameters`。選用元素為 `description`、 `extrinsicId`和 `extrinsicProperties`。

請求說明

描述遵循與第 3.5 節相同的格式，長度上限為 2048 個字元。

回應

在受管整合中，對於透過 SendManagedThingCommand API 傳送的任何動作請求，請求會到達裝置並預期傳回非同步回應。動作回應會定義此回應的結構。

對於動作請求，強制性元素為 `parameters`。選用元素為 `name`、`description`、`extrinsicProperties`、 `extrinsicIderrors`和 `responseCode`。

回應描述

描述遵循與 相同的格式description，長度上限為 2048 個字元。

回應名稱

名稱遵循與 相同的格式名稱 （必要），其中包含這些其他詳細資訊：

- 回應的傳統名稱是透過附加`Response`至動作名稱來衍生。
- 如果您想要使用不同的名稱，您可以在此`name`元素中提供它。如果在回應`name`中提供 ，則此值的優先順序高於傳統名稱。

錯誤

如果處理請求時發生錯誤，回應中提供的唯一訊息的未限制陣列。

約束:

- 訊息項目會宣告為具有下列欄位的 JSON 物件：
  - `code`：包含英數字元 和 _（底線） 的字串，長度介於 1 到 64 個字元之間
  - `message`：未限制的字串值

Example錯誤訊息範例

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

回應代碼

整數代碼，顯示如何處理請求。我們建議裝置程式碼使用 HTTP 伺服器回應狀態程式碼規格傳回程式碼，以允許系統內的一致性。

限制條件：介於 100 到 599 之間的整數值。

請求或回應參數

參數區段定義為名稱和子結構描述對的映射。如果參數可以符合結構描述文件，則可以在請求參數中定義任意數量的參數。

參數名稱只能包含英數字元。不允許使用空格或任何其他字元。

參數欄位

中的必要元素parameter為 extrinsicId和 value。選用元素為 description和 extrinsicProperties。

描述元素遵循與 相同的格式description，長度上限為 1024 個字元。

extrinsicId 和 extrinsicProperties覆寫

extrinsicId 和 extrinsicProperties 遵循與 extrinsicId （強制性）和 相同的格式extrinsicProperties，其中包含這些其他詳細資訊：

- 如果在請求或回應中提供 extrinsicId ，則此值的優先順序高於動作層級提供的值。系統必須先使用請求/回應層級extrinsicId，如果遺失，請使用動作層級 extrinsicId
- 如果在請求或回應extrinsicProperties中提供 ，則這些屬性的優先順序高於動作層級提供的 VAR 值。系統必須採取動作層級，extrinsicProperties並取代在請求/回應層級提供的鍵值對 extrinsicProperties

Example extrinsicId 和 extrinsicProperties 覆寫範例

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

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

在上述範例中，動作請求的有效值為：

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

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

內建動作

對於所有功能，您可以使用關鍵字 ReadState和 執行自訂動作UpdateState。這兩個動作關鍵字將作用於資料模型中定義的功能屬性。

ReadState

將命令傳送至 managedThing，以讀取其狀態屬性的值。使用 ReadState作為強制更新裝置狀態的方法。

UpdateState

傳送命令來更新某些屬性。

在下列情況中，強制裝置狀態同步可能很有用：

1. 裝置已離線一段時間，且未發出事件。
2. 裝置剛佈建，尚未在雲端中維護任何狀態。
3. 裝置狀態與裝置的真實狀態不同步。

ReadState 範例

使用 SendManagedThingCommand API 檢查燈是否開啟：

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

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

讀取 matter.OnOff功能的所有狀態屬性：

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

```
  ]
}
```

UpdateState 範例

使用 SendManagedThingCommand API 變更光源OnTime的 ：

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

事件

事件是由裝置實作的結構描述受管、單向訊號。

根據這些限制條件實作事件：

- 在事件陣列中僅包含唯一事件
- 包含符合結構描述文件大小限制的任何數量的事件

受管整合系統中的事件

處理 事件

事件是主動了解裝置或其環境變更的一種標準化方式。它代表模型化事件，裝置將傳送至雲端，以提供在裝置上修改或在其環境中感知之物件的相關資訊。由於這些事件已建模，客戶可以在控制流程中使用這些事件，以對特定事件和其中提供的詳細資訊做出反應。

對於事件，強制性元素為 `name`和 `extrinsicId`。選用元素為 `description`、`extrinsicProperties`和 `request`。

描述

描述遵循與中所述相同的格式description，長度上限為 512 個字元。

請求

`request` 區段是選用的，如果沒有請求參數，則可以省略。如果省略，系統只要使用事件的名稱，即可支援裝置傳送事件請求而沒有任何承載。這用於簡單的事件，例如幫浦上的感應器故障，或如果煙霧或一氧化碳警示上的警示靜音。

複雜動作需要額外的參數。例如，串流攝影機影片的請求可能包含有關要使用的串流通訊協定或是否將串流傳送至特定顯示裝置的參數。

對於事件請求，強制性元素為 `parameters`。沒有選用的元素。

回應

目前不支援事件回應。

類型定義的結構描述

下列各節詳細說明用於類型定義的結構描述。

$id

$id 元素可識別結構描述定義。它必須遵循此結構：

- 從 `/schema-versions/` URI 字首開始
- 包含`definition`結構描述類型

- 使用正斜線 (/) 做為 URI 路徑分隔符號
- 包含結構描述身分，片段以句點 (.) 分隔
- 使用 @ 字元來分隔結構描述 ID 和版本
- 以 semver 版本結尾，使用句點 (.) 分隔版本片段

結構描述身分的開頭必須是長度為 3-12 個字元的根命名空間，後面接著選用的子命名空間和名稱。

轉換器版本包含 MAJOR 版本 （最多 3 位數）、MINOR 版本 （最多 3 位數） 和選用的 PATCH 版本 （最多 4 位數）。

> **Note**
>
> 您無法使用預留命名空間aws或 matter

Example範例 $id

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

$ref

$ref 元素參考系統中現有的類型定義。它遵循與 $id元素相同的限制條件。

> **Note**
>
> 類型定義或功能必須使用 $ref 檔案中提供的值存在。

Example範例 $ref

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

name

名稱元素是字串，代表結構描述文件中的實體名稱。它通常包含縮寫，並且必須遵循這些規則：

- 僅包含英數字元、句點 (.)、斜線 (/)、連字號 (-) 和空格

- 從字母開始
- 最多 192 個字元

名稱元素用於 Amazon Web Services 主控台 UI 和 文件。

Example範例名稱

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

標題

標題元素是結構描述文件所代表實體的描述性字串。它可以包含任何字元，並用於文件中。

Example範例標題

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

description

`description` 元素提供結構描述文件所代表實體的詳細說明。它可以包含任何字元，並用於文件中。

Example範例描述

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

extrinsicId

`extrinsicId` 元素代表在 Amazon Web Services IoT 系統外部管理的識別符。對於事項功能，它`fieldId`會根據內容對應至 `clusterId`、`eventId`、、 `attributeId` `commandId`或 。

`extrinsicId` 可以是字串化十進位整數 (1-10 位數） 或字串化十六進位整數 (0x 或 0X 字首，後面接著 1-8 個十六進位位數）。

> **Note**
>
> 對於 AWS，廠商 ID (VID) 為 0x1577，對於事項，則為 0。系統會確保自訂結構描述不會將這些預留 VIDs用於 功能。

Example extrinsicIds

```
0018
0x001A
0x15771002
```

extrinsicProperties

`extrinsicProperties` 元素包含一組在外部系統中定義的屬性，但在資料模型中維護。對於事項功能，它會映射到 ZCL 叢集內不同的未建模或部分建模元素、屬性、命令或事件。

外部屬性必須遵循下列限制：

- 屬性名稱必須為英數字元，不含空格或特殊字元
- 屬性值可以是任何 JSON 結構描述值
- 最多 20 個屬性

系統支援各種 `extrinsicProperties`，包括 `access`、`apiMaturity`、`cliFunctionName`、`cli`等。這些屬性有助於 ACL 轉換 AWS （反之亦然） 資料模型。

> **Note**
>
> 功能的 `action`、`property`、 `event`和 `struct` 欄位元素支援外部屬性，但功能或叢集本身不支援。

系統支援的外部屬性

系統會在轉換 ZCL `extrinsicProperties`期間追蹤下列未建模或部分建模的叢集、屬性、命令或事件屬性：

access

每個存取物件都包含下列項目：

- op - 以 建模的操作，enum其值為：write、 read或 invoke
- privilege - 以 建模的權限，enum其值為：view、manage、、 proxy_view operate或 administer
- role - 代表運算子角色的未限制字串

apiMaturity

未繫結的純字串，代表成熟度層級。這在 ZCL 中建模為 enum，其值為：stable、internal、provisional或 deprecated

side

模型化為具有值的列舉：server、 either和 client

布林值屬性

下列屬性是布林值旗標：

- isFabricScoped
- isFabricSensitive
- mustUseAtomicWrite
- mustUseTimedInvoke

字串屬性

下列屬性會以未繫結字串表示：

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

- reportableChange
- reportMinInterval
- reportMaxInterval
- restriction
- storage

轉換考量

對於 ZCL 轉換， extrinsicProperties 會存放在地圖中，無需處理。使用 探索的自訂結構描述不會進行 ZCL 轉換。不過，如果您計劃在未來為自訂結構描述實作 ZCL 轉換，則必須為所有未限制的純字串類型建立模型，extrinsicProperties並定義限制，例如列舉、模式 (regex) 和長度。此準備可確保在轉換期間正確處理這些屬性。

相反地，對於 AWS 連接器轉換，完全extrinsicProperties不包含這些詳細資訊，因為連接器格式不需要這些詳細資訊。

在功能結構描述文件中建置和使用類型定義

結構描述中的所有元素都會解析為類型定義。這些類型定義可以是基本類型定義 （例如布林值、字串、數字） 或命名空間類型定義 （為了方便起見，從基本類型定義建立的類型定義）。

當您定義自訂結構描述時，您可以使用基本定義和命名空間類型定義。

內容

- 點陣圖類型定義
- 列舉類型定義

基本類型定義

基本類型定義是受管整合中定義之所有類型定義的建置區塊。所有命名空間定義，包括自訂類型定義，都可以透過 $ref關鍵字或 type關鍵字解析為基本類型定義。

所有基本類型都可以使用 nullable關鍵字為 null，而且您可以使用 type關鍵字來識別所有基本類型。

布林值

布林值類型支援預設值。

範例定義：

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

整數類型支援

整數類型支援下列項目：

- default 值
- maximum 值
- minimum 值
- exclusiveMaximum 值
- exclusiveMinimum 值
- multipleOf 值

如果 x是正在驗證的值，則下列項目必須是 true：

- x ≥ minimum

- x > exclusiveMinimum
- x < exclusiveMaximum

> **Note**
>
> 零小數部分的數字會被視為整數，但浮點數會遭到拒絕。
>
> ```
> 1.0 // Schema-Compliant
> 3.1415926 // NOT Schema-Compliant
> ```

雖然您可以同時指定 minimum和 exclusiveMinimum 或 maximum和 exclusiveMaximum，但不建議同時使用兩者。

範例定義：

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

替代定義：

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

數字

將數字類型用於任何數字類型，包括整數和浮點數。

數字類型支援下列項目：

- default 值
- maximum 值
- minimum 值
- exclusiveMaximum 值
- exclusiveMinimum 值
- multipleOf 值。該倍數可以是浮點數。

如果 x是正在驗證的值，則下列項目必須是 true：

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

雖然您可以同時指定 minimum和 exclusiveMinimum 或 maximum和 exclusiveMaximum，但不建議同時使用兩者。

範例定義：

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

替代定義：

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

```
}
```

Strings

字串類型支援下列項目：

- `default` 值
- 長度限制（必須是非負數），包括 `maxLength`和 `minLength`值
- `pattern` 規則表達式的值

當您定義規則表達式時，如果表達式符合字串中的任何位置，則字串有效。例如，規則表達式會p比對任何包含 p 的字串，例如 "apple"，而不只是字串 "p"。為了清楚起見，建議您使用 包圍規則表達式 `^...$`（例如 ^p$)，除非您有特定理由不這樣做。

範例定義：

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

Null

Null 類型僅接受單一值：`null`。

範例定義：

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

陣列

陣列類型支援下列項目：

- `default` — 將用作預設值的清單。
- `items` — 對所有陣列元素施加的 JSON 類型定義。

- 長度限制（必須是非負數）
  - minItems
  - maxItems
- pattern Regex 的值
- uniqueItems — 布林值，指出陣列中的元素是否需要是唯一的
- prefixItems — 陣列，其中每個項目都是對應至文件陣列每個索引的結構描述。也就是說，第一個元素驗證輸入陣列的第一個元素，第二個元素驗證輸入陣列的第二個元素，以此類推。

範例定義：

```
{
   "type": "array",
   "default": ["1", "2"],
   "items" : {
        "type": "string",
        "pattern": "^([a-zA-Z0-9_ -/]+)$"
    },
    "minItems" : 1,
    "maxItems": 4,
    "uniqueItems" : true,
}
```

陣列驗證的範例：

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

使用元組驗證的替代定義：

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

```
    ]
}

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

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

物件

物件類型支援下列項目：

- 屬性限制條件
  - properties — 使用 properties關鍵字定義物件的屬性（鍵/值對）。的值properties是物件，其中每個索引鍵都是屬性的名稱，而每個值都是用來驗證該屬性的結構描述。此properties關鍵字會忽略不符合關鍵字中任何屬性名稱的任何屬性。
  - required — 預設不需要properties關鍵字定義的屬性。不過，您可以使用 required關鍵字提供必要屬性的清單。required 關鍵字接受零個或多個字串的陣列。每個字串都必須是唯一的。
  - propertyNames — 此關鍵字允許控制屬性名稱的 RegEx 模式。例如，您可能想要強制執行物件的所有屬性具有遵循特定慣例的名稱。
  - patternProperties — 這會將規則表達式映射至結構描述。如果屬性名稱符合指定的規則運算式，則屬性值必須驗證對應的結構描述。例如，使用 patternProperties指定值應符合特定結構描述，並指定特定類型的屬性名稱。
  - additionalProperties — 此關鍵字控制如何處理額外的屬性。額外屬性是名稱未列在屬性關鍵字中的屬性，或符合 中任何規則表達式的屬性patternProperties。根據預設，允許其他屬性。將此欄位設定為 false表示不允許其他屬性。
  - unevaluatedProperties — 此關鍵字類似於 ，additionalProperties但它可以識別子結構描述中宣告的屬性。 unevaluatedProperties 透過收集在處理結構描述時成功驗證的任何屬性，並使用這些屬性作為允許的屬性清單。這可讓您執行更複雜的動作，例如有條件地新增屬性。如需詳細資訊，請參閱下列範例。
- anyOf — 此關鍵字的值必須是非空白陣列。陣列的每個項目都必須是有效的 JSON 結構描述。如果執行個體成功驗證至少一個由此關鍵字值定義的結構描述，則執行個體會成功對此關鍵字進行驗證。
- oneOf — 此關鍵字的值必須是非空白陣列。陣列的每個項目都必須是有效的 JSON 結構描述。如果執行個體成功針對此關鍵字的值所定義的確切結構描述進行驗證，則執行個體會成功對此關鍵字進行驗證。

必要範例：

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

// Schema Compliant
{
  "test": 4
}

// NOT Schema Compliant
{}
```

PropertyNames 範例：

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

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

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

PatternProperties 範例：

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

```
}

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

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

AdditionalProperties 範例：

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

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

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

UnevaluatedProperties 範例：

```
{
  "type": "object",
  "properties": {
    "standard_field": { "type": "string" }
  },
  "patternProperties": {
    "^@": { "type": "integer" } // Allows properties starting with '@'
```

```
  },
  "unevaluatedProperties": false // No other properties allowed
}

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

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

AnyOf 範例：

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

// Schema Compliant
"short"
12

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

OneOf 範例：

```
{
  "oneOf": [
```

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

// Schema Compliant
10
9

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

命名空間類型定義

命名空間類型定義是從基本類型建置的類型。這些類型必須遵循 *namespace*.*typename*. 受管整合在 aws和 matter 命名空間下提供預先定義類型的格式。您可以將任何命名空間用於自訂類型，預留aws和matter命名空間除外。

若要尋找可用的命名空間類型定義，請使用 ListSchemaVersions API，並將Type篩選條件設定為 definition。

matter 類型

使用 ListSchemaVersions API 在matter命名空間下尋找資料類型，並將Namespace篩選條件設定為 matter，並將Type篩選條件設定為 definition。

aws 類型

使用 ListSchemaVersions API 在aws命名空間下尋找資料類型，並將Namespace篩選條件設定為 aws，並將Type篩選條件設定為 definition。

點陣圖類型定義

點陣圖有兩個必要的屬性：

- type 必須是 物件
- properties 必須是包含每個位元定義的物件。每個位元都是具有屬性 extrinsicId和 的物件value。每個位元的值必須是最小值為 0 且最大值至少為 1 的整數。

範例點陣圖定義：

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

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

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

列舉類型定義

列舉需要三個屬性：

- type 必須是 物件

- enum 必須是一組唯一字串，其中至少有一個項目
- extrinsicIdMap 是具有列舉值屬性的物件。每個屬性的值應該是對應至列舉值的外部識別符。

列舉定義範例：

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

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

// NOT Schema Compliant
"NotAnEnumValue"
```

管理 IoT 裝置命令和事件

除了執行重要的安全性、軟體和硬體更新之外，裝置命令還可讓您遠端管理實體裝置，確保完全控制裝置。使用大型裝置機群時，知道裝置何時執行命令可讓您監督整個裝置實作。裝置命令或自動更新會觸發裝置狀態變更，進而建立新的裝置事件。此裝置事件將觸發自動傳送至客戶受管目的地的通知。

主題

裝置命令

命令請求是傳送至裝置的命令。命令請求包含承載，指定要採取的動作，例如開啟燈泡。若要傳送裝置命令，受管整合會代表最終使用者呼叫 SendManagedThingCommand API，並將命令請求傳送至裝置。

對 的回應SendManagedThingCommand是 `traceId`，您可以盡可能使用此回應`traceId`來追蹤命令交付和任何相關的命令回應工作流程。

如需 SendManagedThingCommand API 操作的詳細資訊，請參閱 SendManagedThingCommand。

`UpdateState` 動作

若要更新裝置的狀態，例如光線開啟的時間，請在呼叫 SendManagedThingCommand API 時使用`UpdateState`動作。提供您在 中更新的資料模型屬性和新值`parameters`。以下範例說明將燈泡的更新為 `OnTime`的 SendManagedThingCommand API 請求5。

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

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

ReadState 動作

若要取得裝置的最新狀態，包括所有資料模型屬性的目前值，請在呼叫 SendManagedThingCommand API 時使用 ReadState動作。在 中propertiesToRead，您可以使用下列選項：

- 提供特定資料模型屬性以取得 的最新值，例如OnOff判斷光線是開啟還是關閉。
- 使用萬用字元運算子 (*) 讀取功能的所有裝置狀態屬性。

以下範例說明使用 ReadState動作之 SendManagedThingCommand API 請求的兩種案例：

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

裝置事件

裝置事件包含裝置的目前狀態。這可能表示裝置已變更狀態，或正在報告其狀態，即使狀態未變更。它包含資料模型中定義的屬性報告和事件。事件可能是洗水機器週期已完成，或調溫器已達到最終使用者設定的目標溫度。

裝置事件通知

最終使用者可以訂閱他們建立的特定客戶受管目的地，以更新特定裝置事件。若要建立客戶受管目的地，請呼叫 `CreateDestination` API。當裝置向受管整合報告裝置事件時，如果有客戶受管目的地，則會通知該目的地。

標記您的受管整合資源

為了協助您管理和組織資源，您可以選擇以標籤形式將自己的中繼資料指派給每個資源。本節說明標籤並示範如何建立它們。

標籤基本概念

您可以使用標籤，以不同方式（例如，依用途、擁有者或環境）分類受管整合資源。當您有許多相同類型的資源時，這會很有用，因為您可以依據先前指派的標籤快速識別資源。每個標籤皆包含由您定義的一個「索引鍵」與選擇性的「值」。例如，您可以為您的物件類型定義一組標籤，協助您以類型追蹤裝置。我們建議您為每種資源類型建立符合您需求的一組標籤金鑰。使用一致的標籤金鑰組可讓您更輕鬆管理您的 資源。

您可以根據新增或套用的標籤來搜尋與篩選資源。您也可以使用標籤來控制對您資源的存取，如 搭配 IAM 政策使用標籤 所述。

為了方便使用， AWS 管理主控台中的標籤編輯器提供集中、統一的方式來建立和管理標籤。如需詳細資訊，請參閱使用 管理主控台中的使用標籤編輯器。 AWS

您也可以使用 AWS CLI 和 受管整合 API 來使用標籤。當您使用下列命令中的 Tags 欄位建立標籤時，您可以將標籤與受管物件、佈建設定檔、登入資料儲存庫和over-the-air(OTA) 任務建立關聯：

- CreateManagedThing
- CreateProvisioningProfile
- CreateCredentialLocker
- CreateOtaTask
- CreateAccountAssociation

您可以使用下列命令新增、修改或刪除支援標記功能的現有資源標籤：

- TagResource
- ListTagsForResource
- UntagResource

您可以編輯標籤索引鍵和值，也可以隨時從資源中移除標籤。您可以將標籤的值設為空白字串，但您無法將標籤的值設為 Null。如果您新增的標籤具有與該資源上現有標籤相同的索引鍵，則新值會覆寫舊值。如果您刪除資源，也會刪除與該資源相關聯的任何標籤。

標籤的限制與上限

以下基本限制適用於 標籤：

- 每個資源的標籤數上限：50
- 金鑰長度上限：127 個 UTF-8 Unicode 字元
- 值長度上限：255 個 UTF-8 Unicode 字元
- 標籤鍵與值皆區分大小寫。
- 請勿於標籤名稱或值中使用 `aws:` 字首。它保留供 AWS 使用。您不可編輯或刪除具此字首的標籤名稱或值。具此字首的標籤，不算在受資源限制的標籤計數內。
- 如果您的標記結構描述是跨多項服務和資源使用，請記得其他服務可能會有字元使用限制。允許使用的字元包括可用 UTF-8 表示的英文字母、空格和數字，以及以下特殊字元：+ - = . _ : / @。

搭配 IAM 政策使用標籤

您可以在用於受管整合 API 動作的 IAM 政策中套用標籤型資源層級許可。這可讓您更有效地控制使用者可以建立、修改或使用哪些資源。您可以使用 `Condition` 元素 (也稱為 `Condition` 區塊)，以及 IAM 政策中的以下條件內容金鑰和值，來根據資源標籤控制使用者存取 (許可)：

- 使用 `aws:ResourceTag/`*`tag-key`*`: `*`tag-value`* 以允許或拒絕資源上具有特定標籤的使用者動作。
- 使用 `aws:RequestTag/`*`tag-key`*`: `*`tag-value`* 以在提出 API 請求時，要求使用 (或不使用) 特定標籤，以建立或修改允許標籤的資源。
- 使用 `aws:TagKeys: [`*`tag-key`*`, ...]` 以在提出 API 請求時，要求使用 (或不使用) 特定標籤金鑰集，以建立或修改允許標籤的資源。

> **Note**
>
> IAM 政策中的條件內容索引鍵和值僅適用於那些受管整合動作，其中能夠標記之資源的識別符是必要的參數。例如，根據條件內容索引鍵和值，不允許或拒絕使用 GetCustomEndpoint，因為在此請求中未參考任何可標記的資源 （ 受管物件、佈建設定檔、登入資料儲存庫、over-the-

air任務）。如需可標記的受管整合資源及其支援的條件索引鍵的詳細資訊，請參閱 的 AWS IoT 受管整合功能的動作、資源和條件索引鍵 AWS IoT Device Management。

如需使用標籤的詳細資訊，請參閱《AWS Identity and Access Management 使用者指南》中的使用標籤控制。該指南的 IAM JSON 政策參考章節有詳細的語法、說明，還有元素、變數範例，以及在 IAM 中的 JSON 政策評估邏輯。

下列範例政策會套用兩個以標籤為基礎的 `CreateManagedThing` 動作限制。受到此政策限制的 IAM 使用者：

- 無法建立標籤為 "env=prod" 的受管物件（在範例中，請參閱行 `"aws:RequestTag/env" : "prod"`）。
- 無法修改或存取具有現有標籤 "env=prod" 的受管物件（在範例中，請參閱行 `"aws:ResourceTag/env" : "prod"`）。

JSON

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

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

您也可以透過將其包含在清單中，為特定標籤金鑰指定多個標籤值，如下所示：

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

> **Note**
>
> 如果您允許或拒絕使用者根據標籤存取資源，請務必考慮明確拒絕使用者將這些標籤新增至相同資源或從中移除的能力。否則，使用者可能透過修改標籤來避開您的限制，並取得資源的存取。

受管整合通知

受管整合通知可提供來自裝置的更新和重要洞見。通知包括連接器事件、裝置命令、生命週期事件、OTA Over-the-Air) 更新和錯誤報告。這些洞見提供可行的資訊，可用來建立自動化工作流程、立即採取行動，或存放事件資料以進行故障診斷。

目前，僅支援 Amazon Kinesis 資料串流做為受管整合通知的目的地。您必須先設定 Amazon Kinesis 資料串流，並允許受管整合存取資料串流，再設定通知。

為通知設定 Amazon Kinesis

Amazon Kinesis 設定步驟

- 步驟 1：建立 Amazon Kinesis 資料串流
- 步驟 2：建立許可政策
- 步驟 3：導覽至 IAM 儀表板，然後選取角色
- 步驟 4：使用自訂信任政策
- 步驟 5：套用您的許可政策
- 步驟 6：輸入角色名稱

若要為受管整合通知設定 Amazon Kinesis，請遵循下列步驟：

步驟 1：建立 Amazon Kinesis 資料串流

Amazon Kinesis Data Stream 可以即時擷取大量資料、長期存放資料，並讓資料可供應用程式使用。

建立 Amazon Kinesis 資料串流

- 若要建立 Kinesis 資料串流，請遵循建立和管理 Kinesis 資料串流中概述的步驟。

步驟 2：建立許可政策

建立允許受管整合存取 Kinesis 資料串流的許可政策。

建立許可政策

- 若要建立許可政策，請複製以下政策，並遵循使用 JSON 編輯器建立政策中概述的步驟

JSON

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

步驟 3：導覽至 IAM 儀表板，然後選取角色

開啟 IAM 儀表板，然後按一下角色。

導覽至 IAM 儀表板

- 開啟 IAM 儀表板，然後按一下角色。

  如需詳細資訊，請參閱AWS Identity and Access Management《 使用者指南》中的建立 IAM 角色。

步驟 4：使用自訂信任政策

您可以使用自訂信任政策來授予 Kinesis 資料串流的受管整合存取權。

使用自訂信任政策

- 建立新的角色，然後選擇自訂信任政策。按一下下一步。

  下列政策允許 受管整合擔任角色，而 `Condition`陳述式有助於防止混淆代理人問題。

  JSON

```
{
    "Version":"2012-10-17",
```

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

步驟 5：套用您的許可政策

將您在步驟 2 中建立的許可政策新增至角色。

新增許可政策

- 在新增許可頁面上，搜尋並新增您在步驟 2 中建立的許可政策。按一下下一步。

步驟 6：輸入角色名稱

- 輸入角色名稱，然後按一下建立角色。

設定受管整合通知

通知設定步驟

- 步驟 1：授予使用者呼叫 CreateDestination API 的許可
- 步驟 2：呼叫 CreateDestination API
- 步驟 3：呼叫 CreateNotificationConfiguration API

若要設定受管整合通知，請遵循下列步驟：

步驟 1：授予使用者呼叫 CreateDestination API 的許可

- 授予使用者呼叫 **CreateDestination** API 的許可

  下列政策定義使用者呼叫 CreateDestination API 的需求。

  請參閱AWS Identity and Access Management《 使用者指南》中的授予使用者將角色傳遞至 AWS 服務的許可，以取得受管整合的傳遞許可。

  JSON

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

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

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

步驟 2：呼叫 CreateDestination API

- 呼叫 **CreateDestination** API

在您建立 Amazon Kinesis 資料串流和串流存取角色之後，請呼叫 CreateDestination API 來建立通知目的地，以便將通知路由到該目的地。對於 DeliveryDestinationArn 參數，請使用來自新 Amazon Kinesis 資料串流arn的 。

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

> **Note**
>
> ClientToken 是等冪符記。如果您重試最初使用相同用戶端字符和參數成功完成的請求，則重試嘗試將成功，而無需執行任何其他動作。

步驟 3：呼叫 CreateNotificationConfiguration API

- 呼叫 **CreateNotificationConfiguration** API

  最後，使用 CreateNotificationConfiguration API 建立通知組態，將所選事件類型路由到 Kinesis 資料串流所代表的目的地。在 DestinationName 參數中，使用與最初呼叫 CreateDestination API 時相同的目的地名稱。

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

受管整合監控的事件類型

以下是使用受管整合通知監控的事件類型：

- DEVICE_COMMAND

- SendManagedThingCommand API 命令的狀態。有效值是 `succeeded` 或 `failed`。

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

- `DEVICE_COMMAND_REQUEST`
  - Web 即時通訊 (WebRTC) 的命令請求。

    WebRTC 標準允許兩個對等之間的通訊。這些對等可以傳輸即時視訊、音訊和任意資料。受管整合支援 WebRTC，可在客戶行動應用程式和最終使用者的裝置之間啟用這些類型的串流。如需 WebRTC 標準的詳細資訊，請參閱 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
  - 裝置探索狀態。

```
{
      "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
  - 發生裝置事件的通知。

```
{
      "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
  - 裝置生命週期的狀態。

```
{
      "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
  - 裝置 OTA 通知。
- DEVICE_STATE
  - 更新裝置狀態時的通知。

```
{
      "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) 連接器

cloud-to-cloud連接器可讓您建立和促進第三方裝置與 之間的雙向通訊 AWS。

主題

什麼是cloud-to-cloud(C2C) 連接器？

cloud-to-cloud連接器是預先建置的軟體套件，可安全地 AWS 雲端 將 連結至第三方雲端供應商的端點。使用 C2C 連接器，解決方案供應商可以利用 AWS IoT Device Management 的受管整合來控制連接到第三方雲端的裝置。

受管整合包含連接器的目錄， AWS 客戶可以在其中檢視和選取要與之整合的連接器。如需詳細資訊，請參閱什麼是 C2C 連接器目錄？

受管整合需要將每個連接器實作為 AWS Lambda 函數。

什麼是 C2C 連接器目錄？

AWS IoT Device Management 連接器目錄的受管整合是 C2C 連接器的集合，可促進 AWS IoT Device Management 的受管整合與第三方雲端提供者之間的雙向通訊。您可以在 AWS 管理主控台 或 中檢視連接器 AWS CLI。

使用主控台檢視受管整合連接器目錄

1. 開啟受管整合主控台
2. 在左側導覽窗格中，選擇受管整合
3. 在受管整合主控台的左側導覽窗格中，選擇目錄。

AWS Lambda 做為 C2C 連接器的 函數

每個 C2C 連接器 Lambda 函數都會在受管整合和第三方平台上的對應動作之間轉譯和傳輸命令和事件。如需 Lambda 的詳細資訊，請參閱什麼是 AWS Lambda 。

例如，假設最終使用者擁有第三方 OEM 製造的智慧燈泡。使用 C2C 連接器，最終使用者可以發出命令，透過受管整合平台開啟或關閉此光源。此命令接著會轉送至連接器中託管的 Lambda 函數，這會將請求轉換為針對第三方平台的 API 呼叫，以開啟或關閉裝置。

當您呼叫 `CreateCloudConnector` API 時，需要 Lambda 函數。部署至 Lambda 函數的程式碼必須實作所有界面和功能，如 中所述建置 C2C Cloud-to-Cloud) 連接器。

受管整合連接器工作流程

開發人員必須向 的受管整合註冊 C2C 連接器 AWS IoT Device Management。此註冊程序會建立邏輯連接器資源，供客戶存取以使用連接器。

> **Note**
>
> C2C 連接器是在 AWS IoT Device Management 受管整合中建立的一組中繼資料，用於描述連接器。

下圖說明從行動應用程式將命令傳送至雲端連線裝置時C2C 連接器的角色。C2C 連接器可做為 AWS IoT Device Management 受管整合與第三方雲端平台之間的轉譯層。

使用 C2C cloud-to-cloud) 連接器的指導方針

您建立的任何 C2C 連接器都是您的內容，而您存取的其他客戶建立的任何 C2C 連接器都是第三方內容。 AWS 不會在受管整合中建立或管理任何 C2C 連接器。

您可以與其他受管整合客戶共用 C2C 連接器。如果您這麼做，您授權 AWS 做為服務供應商，在 AWS 主控台上列出這些 C2C 連接器和相關聯絡資訊，而且您了解其他 AWS 客戶可能會與您聯絡。您全權負責授予客戶存取 C2C 連接器的權限，以及管理其他 AWS 客戶存取 C2C 連接器的任何條款。

建置 C2C Cloud-to-Cloud) 連接器

下列各節涵蓋為 AWS IoT Device Management 的受管整合建置 C2C Cloud-to-Cloud) 連接器的步驟。

主題

先決條件

在建立 C2C Cloud-to-Cloud) 連接器之前，您需要下列項目：

- AWS 帳戶 託管 C2C 連接器並透過受管整合註冊的 。如需詳細資訊，請參閱建立 AWS 帳戶。
- 當您建置連接器時，您需要特定的 IAM 許可。使用
- 確保連接器要用於的第三方雲端提供者支援 OAuth 2.0 授權。如需詳細資訊，請參閱帳戶連結的 OAuth 2.0 要求。

  此外，若要測試連接器，連接器的開發人員必須具有下列項目：

  - 要與 C2C 連接器建立關聯的第三方雲端用戶端 ID
  - 來自第三方雲端的用戶端秘密，可與您的 C2C 連接器建立關聯
  - OAuth 2.0 授權 URL
  - OAuth 2.0 字符 URL
- 第三方 API 所需的任何 API 金鑰
- 第三方 API 註冊或允許列出 託管的 OAuth 回呼 URL 所需的任何 API 金鑰 AWS。有些第三方明確允許列出 OAuth 重新導向 URL，有些則具有工作流程，使用者可以登入並註冊 OAuth URL。諮詢特定第三方，以了解允許列出受管整合 OAuth 重新導向端點所需的內容

所需的許可

當您建置連接器時，您需要特定的 IAM 許可。除了 動作的`iotmanagedintegrations:`許可之外，您還需要下列許可：

- CreateAccountAssociation、CreateConnectorDestination、GetAccountAssociation 和 StartAccountAssociationRefresh，需要 `secretsmanager:GetSecretValue`
- CreateCloudConnector 需要 `lambda:Invoke`

如需`iotmanagedintegrations:`許可和動作的詳細資訊，請參閱AWS 受管整合定義的動作

C2C 連接器需求

您開發的 C2C 連接器有助於 AWS IoT Device Management 受管整合與第三方廠商雲端之間的雙向通訊。連接器必須針對 AWS IoT Device Management 的受管整合實作界面，以代表最終使用者執行動作。這些界面提供探索最終使用者裝置、啟動從 AWS IoT Device Management 受管整合傳送的裝置命令，以及根據存取字符識別使用者的功能。若要支援裝置操作，連接器必須管理 AWS IoT Device Management 受管整合與相關第三方平台之間的請求和回應訊息翻譯。

以下是 C2C 連接器的要求：

- 第三方授權伺服器必須符合 OAuth 2.0 標準以及 中列出的組態。 OAuth 組態需求
- 需要 C2C 連接器才能從事項資料模型的 AWS 實作解譯識別符，並且必須發出符合事項資料模型 AWS 實作的回應和事件。如需詳細資訊，請參閱AWS 實作事項資料模型
- C2C 連接器必須能夠使用`SigV4`身分驗證來呼叫 AWS IoT Device Management APIs的受管整合。對於使用 SendConnectorEvent API 傳送的非同步事件，必須使用用於註冊連接器的相同 AWS 帳戶登入資料來簽署相關的 SendConnectorEvent 請求。
- 連接器必須實作 `AWS.ActivateUser`、`AWS.SendCommand`、 `AWS.DiscoverDevices`和 `AWS.DeactivateUser`操作。
- 當您的 C2C 連接器收到與裝置命令回應或裝置探索相關的第三方事件時，必須將其轉送至與 `SendConnectorEvent` API 的受管整合。如需這些事件和 `SendConnectorEvent` API 的詳細資訊，請參閱 SendConnectorEvent。

> **Note**
>
> `SendConnectorEvent` API 是 受管整合 SDK 的一部分，使用 ，而不是手動建置和簽署請求。

帳戶連結的 OAuth 2.0 要求

每個 C2C 連接器都依賴 OAuth 2.0 授權伺服器來驗證最終使用者。透過此伺服器，最終使用者將其第三方帳戶與客戶的裝置平台連結。帳戶連結是最終使用者使用 C2C 連接器支援的裝置所需的第一個步驟。如需帳戶連結和 OAuth 2.0 中不同角色的詳細資訊，請參閱 帳戶連結角色。

雖然您的 C2C 連接器不需要實作特定商業邏輯來支援授權流程，但與您的 C2C 連接器相關聯的 OAuth2.0 授權伺服器必須符合 OAuth 組態需求。

> **Note**
>
> 的受管整合 AWS IoT Device Management 僅支援具有授權碼流程的 OAuth 2.0。如需詳細資訊，請參閱 RFC 6749。

帳戶連結是允許受管整合和連接器使用存取字符存取最終使用者裝置的程序。此字符提供具有最終使用者許可的 AWS IoT Device Management 受管整合，以便連接器可以透過 API 呼叫與最終使用者的資料互動。如需詳細資訊，請參閱帳戶連結工作流程。

建議您不要在任何日誌中記錄這些敏感字符。不過，如果它們存放在日誌中，我們建議您使用 CloudWatch Logs 資料保護政策來遮罩日誌中的字符。如需詳細資訊，請參閱使用遮罩功能協助保護敏感日誌資料。

的受管整合 AWS IoT Device Management 不會直接取得存取權杖；它透過授權碼授予類型這樣做。首先，AWS IoT Device Management 的受管整合必須取得授權碼。然後，它會交換存取字符和重新整理字符的程式碼。重新整理字符用於在舊的存取權杖過期時請求新的存取權杖。如果存取字符和重新整理字符都已過期，您必須再次執行帳戶連結流程。您可以使用 `StartAccountAssociationRefresh` API 操作來執行此操作。

> **Important**
>
> 發行的存取權杖必須針對每個使用者設定範圍，但不能針對 OAuth 用戶端設定範圍。字符不應提供存取權給用戶端下所有使用者的所有裝置。
> 授權伺服器必須執行下列其中一項操作：
>
> - 發出包含可擷取最終使用者（資源擁有者）ID 的存取權杖，例如 JWT 權杖。
> - 傳回每個發行存取字符的最終使用者 ID。

OAuth 組態需求

下表說明來自 OAuth 授權伺服器的必要參數，用於 AWS IoT Device Management 執行帳戶連結的受管整合：

OAuth 伺服器參數

欄位	必要	註解
clientId	是	應用程式的公有識別符。它用於啟動身分驗證流程，並且可以公開共用。
clientSecret	是	用來向授權伺服器驗證應用程式的私密金鑰，特別是在交換存取字符的授權碼時。它應該保持機密，而不是公開共用。
authorizationType	是	此授權組態支援的授權類型。目前，「OAuth 2.0」是唯一支援的值。
authUrl	是	第三方雲端供應商的授權 URL。
tokenUrl	是	第三方雲端提供者的字符 URL。
tokenEndpointAuthenticationScheme	是	「HTTP_BASIC」或「REQUEST_BODY_CREDENTIALS」的身分驗證機制。HTTP_BASIC 表示用戶端憑證包含在授權標頭中，而梯形訊號則包含在請求內文中。

您必須設定您使用的 OAuth 伺服器，以便存取權杖字串值必須使用 UTF-8 字元集進行 Base64 編碼。

帳戶連結角色

若要建立 C2C 連接器，您需要 OAuth 2.0 授權伺服器和帳戶連結。如需詳細資訊，請參閱帳戶連結工作流程。

OAuth 2.0 會在實作帳戶連結時定義下列四個角色：

1. 授權伺服器
2. 資源擁有者 （最終使用者）
3. 資源伺服器
4. 用戶端

以下定義每個 OAuth 角色：

授權伺服器

授權伺服器是識別和驗證第三方雲端中最終使用者身分的伺服器。此伺服器提供的存取字符可以連結 AWS 最終使用者的客戶平台帳戶及其第三方平台帳戶。此程序稱為帳戶連結。

授權伺服器提供下列項目，以支援帳戶連結：

- 顯示登入頁面，供最終使用者登入您的系統。這通常稱為授權端點。
- 驗證系統中的最終使用者。
- 產生識別最終使用者的授權碼。
- 將授權碼傳遞給 AWS IoT Device Management 的受管整合。
- 接受來自 AWS IoT Device Management 受管整合的授權碼，並傳回 AWS IoT Device Management 受管整合可用來存取系統中最終使用者資料的存取字符。這通常透過單獨的 URI 完成，稱為字符 URI 或端點。

> **Important**
>
> 授權伺服器必須支援 OAuth 2.0 授權碼流程，才能與 AWS IoT Device Management Connector 的 受管整合搭配使用。AWS IoT Device Management 的 受管整合也支援使用 Code Exchange 的驗證金鑰 (PKCE) 的授權碼流程。
>
> 授權伺服器必須：
>
> - 發出包含可擷取最終使用者或資源擁有者 ID 的存取權杖，例如 JWT 權杖
> - 能夠為每個發行的存取權杖傳回最終使用者 ID

否則，您的連接器將無法支援所需的AWS.ActivateUser操作。這將防止連接器使用受管整合。

如果連接器開發人員或擁有者未維護自己的授權伺服器，則所使用的授權伺服器必須為連接器開發人員第三方平台管理的資源提供授權。這表示受管整合從授權伺服器接收的任何字符都必須在裝置（資源）上提供有意義的安全界限。例如，最終使用者字符不允許在其他最終使用者裝置上使用命令；字符提供的許可會映射到平台內的資源。考慮 Lights Incorporated 範例。當最終使用者使用連接器啟動帳戶連結流程時，他們會被重新導向到 Lights Incorporated 登入頁面，該頁面位於授權伺服器的前面。一旦他們登入並授予許可給用戶端，就會提供權杖，讓連接器存取其 Lights Incorporated 帳戶中的資源。

資源擁有者（最終使用者）

身為資源擁有者，您可以允許 AWS IoT Device Management 客戶的受管整合透過執行帳戶連結來存取與您帳戶相關聯的資源。例如，假設終端使用者已加入 Lights Incorporated 行動應用程式的智慧燈泡。資源擁有者是指已購買並加入裝置的最終使用者帳戶。在我們的範例中，資源擁有者會建模為 Lights Incorporated OAuth2.0 帳戶。身為資源擁有者，此帳戶提供發出命令和管理裝置的許可。

資源伺服器

這是託管受保護資源的伺服器，需要授權才能存取（裝置資料）。 AWS 客戶需要代表最終使用者存取受保護的資源，並透過 AWS IoT Device Management 連接器的受管整合來進行帳戶連結。以之前的智慧型燈泡為例，資源伺服器是 Lights Incorporated 擁有的雲端服務，可在燈泡加入後對其進行管理。透過資源伺服器，資源擁有者可以向智慧燈泡發出命令，例如開啟和關閉它。受保護的資源只會提供許可給最終使用者的帳戶，以及他們可能已提供許可的其他帳戶/實體。

用戶端

在此內容中，用戶端是您的 C2C 連接器。用戶端定義為代表最終使用者授予資源伺服器內資源存取權的應用程式。帳戶連結程序代表連接器、用戶端，請求存取第三方雲端中的最終使用者資源。

雖然連接器是 OAuth 用戶端，但 AWS IoT Device Management 的受管整合會代表連接器執行操作。例如，AWS IoT Device Management 的受管整合會向授權伺服器發出請求，以取得存取權杖。連接器仍被視為用戶端，因為它是唯一可存取資源伺服器中受保護資源（裝置資料）的元件。

請考慮最終使用者已加入的智慧型燈泡。在客戶平台和 Lights Incorporated 授權伺服器之間完成帳戶連結之後，連接器本身會與資源伺服器通訊，以擷取最終使用者智慧型燈泡的相關資訊。連接器

接著可以從最終使用者接收命令。這包括透過 Lights Incorporated 資源伺服器代表他們開啟或關閉燈光。因此，我們將連接器指定為用戶端。

帳戶連結工作流程

若要讓客戶的 AWS IoT Device Management 平台受管整合透過 C2C 連接器與第三方平台上的最終使用者裝置互動，可透過下列工作流程取得存取權杖：

1. 當使用者透過客戶應用程式啟動第三方裝置的加入時，AWS IoT Device Management 的受管整合會傳回授權 URI 和 AssociationId。
2. 應用程式前端會存放 AssociationId，並將最終使用者重新導向至第三方平台的登入頁面。
   - 最終使用者登入。最終使用者授予用戶端存取其裝置資料的權限。
3. 第三方平台會建立授權碼。最終使用者會重新導向至 AWS IoT Device Management 平台回呼 URI 的受管整合，包括連接至重新導向請求的程式碼。
4. 受管整合會與第三方平台字符 URI 交換此程式碼。
5. 字符 URI 會驗證授權碼，並傳回與最終使用者相關聯的 OAuth2.0 存取字符和重新整理字符。
6. 受管整合會使用 `AWS.ActivateUser`操作呼叫 C2C 連接器，以完成帳戶連結流程並取得 UserId。
7. 受管整合會將成功身分驗證頁面的 OAuthRedirectUrl （從連接器政策組態） 傳回給客戶應用程式。

   > **Note**
   >
   > 如果發生故障，AWS IoT Device Management 的受管整合會將錯誤和 error_description 查詢參數附加至 URL，以提供錯誤詳細資訊給客戶應用程式。

8. 客戶應用程式會將最終使用者重新導向至 OAuthRedirectUrl。此時，應用程式前端會從第一個步驟知道關聯的 AssociationId。

   從 AWS IoT Device Management 透過 C2C 連接器到第三方雲端平台的受管整合提出的所有後續請求，例如探索裝置和傳送命令的命令，都會包含 OAuth2.0 存取字符。

下圖顯示帳戶連結的關鍵元件之間的關係：

實作 C2C 連接器界面操作

的受管整合 AWS IoT Device Management 定義了 AWS Lambda 您必須處理的四個操作，才能符合連接器的資格。您的 C2C 連接器必須實作下列每個操作：

1. AWS.ActivateUser - AWS IoT Device Management 服務的受管整合會呼叫此 API，以擷取與提供的 OAuth2.0 字符相關聯的全域唯一使用者識別符。此操作可以選擇性地用於執行帳戶連結程序的任何其他需求。
2. AWS.DiscoverDevices - AWS IoT Device Management 服務的受管整合會將此 API 呼叫您的連接器，以探索使用者的裝置
3. AWS.SendCommand - AWS IoT Device Management 服務的受管整合會將此 API 呼叫您的連接器，以傳送使用者裝置的命令
4. AWS.DeactivateUser - AWS IoT Device Management 服務的受管整合會將此 API 呼叫您的連接器，以停用使用者的存取權杖，以在授權伺服器中取消連結。

的受管整合 AWS IoT Device Management 一律會透過 動作，使用 JSON 字串承載叫用 Lambda AWS Lambda invokeFunction函數。請求操作必須在每個請求承載中包含 operationName 欄位。如需詳細資訊，請參閱《 AWS Lambda API 參考》中的叫用。

每個調用逾時都會設定為兩秒，如果調用失敗，將會重試五次。

您為連接器實作的 Lambda 會從operationName請求承載剖析 ，並實作對應至第三方雲端的對應功能：

```
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**
>
> 連接器的開發人員必須實作上述範例中列出的 `activateUserManager.activateUser(request)`、`sendCommandManager.sendCommand(re` `deviceDiscoveryManager.listDevices(request)`和 `deactivateUser.deactivateUser`操作。

下列範例詳細說明來自 受管整合的一般連接器請求，其中每個必要界面的常見欄位都存在。從範例中，您可以看到同時有請求標頭和請求承載。請求標頭在每個操作界面中都是常見的。

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

預設請求標頭

預設標頭欄位如下所示。

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

```
        }
    }
}
```

連接器託管的任何 API 必須處理下列標頭參數：

預設標頭和欄位

欄位	必要/選用	Description
`header:auth`	是	C2C 連接器建置器在連接器註冊期間提供的授權資訊。
`header:auth:token`	是	第三方雲端提供者產生並連結至 之使用者的授權字符 `connectorAssociationID` 。
`header:auth:type`	是	所需的授權類型。

> **Note**
>
> 連接器的所有請求都會連接最終使用者的存取權杖。您可以假設最終使用者與受管整合客戶之間的帳戶連結已發生。

請求承載

除了常見的標頭，每個請求都會有承載。雖然此承載的每個操作類型都有唯一的欄位，但每個承載都有一組永遠存在的預設欄位。

請求承載欄位：

- `operationName`：指定請求的操作，等於下列其中一個值：`AWS.ActivateUser`、`AWS.SendCommand`、`AWS.DiscoverDevices`、`AWS.DeactivateUser`。
- `operationVersion`：每個操作都會進行版本化，以允許其隨著時間演進，並為第三方連接器提供穩定的界面定義。 受管整合會在所有請求的承載中傳遞版本欄位。
- `connectorId`：已傳送請求的連接器 ID。

預設回應標頭

每個操作都會使用 回應 AWS IoT Device Management 的ACK受管整合，以確認您的 C2C 連接器已收到請求並開始處理。以下是所述回應的一般範例：

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

每個操作回應都必須具有下列常見標頭：

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

下表列出預設回應標頭：

預設回應標頭和欄位

欄位	必要/選用	註解
header:responseCode	是	指出請求執行狀態的值 ENUM。

在本文件所述的各種連接器界面和 API 結構描述中，都有 responseMessage或 Message 欄位。這是選用欄位，用於 C2C 連接器 Lambda 回應有關請求及其執行的任何內容。最好是，導致狀態碼 以外的任何錯誤200都應該包含描述錯誤的訊息值。

使用 SendConnectorEvent API 回應 C2C 連接器操作請求

的受管整合 AWS IoT Device Management 預期您的連接器會針對每個 AWS.SendCommand和 AWS.DiscoverDevices操作以非同步方式運作。這表示對這些操作的初始回應只是「認可」您的 C2C 連接器已收到請求。

使用 SendConnectorEvent API，預期您的連接器會將事件類型從下列清單傳送至 AWS.DiscoverDevices和 AWS.SendCommand操作，以及主動裝置事件 （例如手動開啟和關閉的光源）。若要閱讀這些事件類型及其使用案例的詳細說明，請參閱 實作 AWS.DiscoverDevices 操作、實作 AWS.SendCommand 操作和 使用 SendConnectorEvent API 傳送裝置事件。

例如，如果您的 C2C 連接器收到DiscoverDevices請求，AWS IoT Device Management 的受管整合預期它會與上述定義的回應格式同步回應。然後，您必須針對 實作 AWS.DiscoverDevices 操作DEVICE_DISCOVERY 事件，使用 中定義的請求結構叫用 SendConnectorEventAPI。SendConnectorEvent 進行 API 呼叫時，您可以在任何可存取 C2C 連接器 Lambda AWS 帳戶 登入資料的地方進行。在 AWS IoT Device Management 的受管整合收到此事件之前，裝置探索流程不會成功。

> **Note**
>
> 或者，如有必要，SendConnectorEventAPI 呼叫可以在 C2C 連接器 Lambda 調用回應之前進行。不過，此流程與軟體開發的非同步模型相衝突。

- SendConnectorEvent - 連接器會呼叫 AWS IoT Device Management API 的受管整合，將裝置事件傳送至 AWS IoT Device Management 的受管整合。只有 3 種受管整合接受的事件類型：
  - "DEVICE_DISCOVERY" – 此事件操作應用於為特定存取字符傳送第三方雲端內探索的裝置清單。
  - "DEVICE_COMMAND_RESPONSE" – 此事件操作應用於傳送特定裝置事件，作為命令執行的結果。
  - "DEVICE_EVENT" – 此事件操作應用於源自於裝置的任何事件，而非使用者型命令的直接結果。這可以做為一般事件類型，主動報告裝置狀態變更或通知。

實作 AWS.ActivateUser 操作

AWS IoT Device Management 的受管整合需要 AWS.ActivateUser操作，才能從最終使用者的 OAuth2.0 字符擷取使用者識別符。的受管整合 AWS IoT Device Management 會在請求標頭中傳遞 OAuth 字符，並預期您的連接器在回應承載中包含全域唯一使用者識別符。此操作會在帳戶連結流程成功之後發生。

下列清單概述連接器的要求，以促進成功的AWS.Activate使用者流程。

- 您的 C2C 連接器 Lambda 可以處理來自 AWS IoT Device Management 受管整合AWS.ActivateUser的操作請求訊息。

- 您的 C2C 連接器 Lambda 可以從提供的 OAuth2.0 字符中判斷唯一的使用者識別符。一般而言，如果它是 JWT 權杖，或權杖從授權伺服器請求，它可以從權杖本身擷取。

`AWS.ActivateUser` 工作流程

1. 的受管整合會使用下列承載 AWS IoT Device Management 叫用 C2C 連接器 Lambda：

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

2. C2C 連接器會從字符或透過查詢您的第三方資源伺服器，決定要包含在`AWS.ActivateUser`回應中的使用者 ID。
3. C2C 連接器會回應 Lambda 調用`AWS.ActivateUser`操作，包括預設承載以及 `userId` 欄位內對應的使用者識別符。

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

實作 AWS.DiscoverDevices 操作

裝置探索會將最終使用者擁有的實體裝置清單與 AWS IoT Device Management 受管整合中維護的那些最終使用者裝置的數位表示法保持一致。只有在使用者與 AWS IoT Device Management 的受管整合之間完成帳戶連結之後， AWS 客戶才會在最終使用者擁有的裝置上執行。裝置探索是一種非同步程序，其中 AWS IoT Device Management 的受管整合會呼叫連接器來啟動裝置探索請求。C2C 連接器會傳回由受管整合產生的與參考識別碼 （稱為 deviceDiscoveryId) 非同步的探索最終使用者裝置清單。

下圖說明最終使用者與 AWS IoT Device Management 受管整合之間的裝置探索工作流程：

AWS.DiscoverDevices 工作流程

1. 客戶代表最終使用者啟動裝置探索程序。
2. 的受管整合會deviceDiscoveryId針對客戶產生的裝置探索請求 AWS IoT Device Management 產生名為 的 AWS 參考識別符。
3. 的受管整合會使用 AWS.DiscoverDevices操作介面將裝置探索請求 AWS IoT Device Management 傳送至 C2C 連接器，包括accessToken最終使用者的有效 OAuth 以及 deviceDiscoveryId。
4. 您的連接器存放deviceDiscoveryId區要包含在DEVICE_DISCOVERY事件中。此事件也會包含探索到的最終使用者裝置清單，而且必須傳送至 AWS IoT Device Management 的受管整合，並將 SendConnectorEvent API 做為DEVICE_DISCOVERY事件。
5. 您的 C2C 連接器應呼叫資源伺服器，以擷取最終使用者擁有的所有裝置。
6. 您的 C2C 連接器 Lambda 會使用 ACK 回應來回應 Lambda 呼叫 (invokeFunction)，以傳回 AWS IoT Device Management 的受管整合，做為AWS.DiscoverDevices操作的初始回應。受管整合會通知客戶其已啟動裝置探索程序的 ACK。
7. 您的資源伺服器會將最終使用者擁有和操作的裝置清單傳送給您。
8. 連接器會將每個最終使用者裝置轉換為 AWS IoT Device Management 所需裝置格式的受管整合，包括 ConnectorDeviceIdConnectorDeviceName和每個裝置的功能報告。
9. C2C 連接器也UserId提供探索到的裝置擁有者。視您的資源伺服器實作而定，它可能會在裝置清單或個別呼叫中從您的資源伺服器擷取。
10. 接著，您的 C2C 連接器將使用 AWS 帳戶 登入資料和操作參數設定為 "DEVICE_DISCOVERY"SendConnectorEvent，透過 SigV4 呼叫 AWS IoT Device Management API 的受管整合 。傳送至 AWS IoT Device Management 受管整合之裝置清單中的

每個裝置都會以裝置特定參數表示connectorDeviceName，例如 connectorDeviceId、 和 capabilityReport。

- 根據您的資源伺服器回應，您需要相應地通知 AWS IoT Device Management 的受管整合。

  例如，如果您的資源伺服器對最終使用者探索的裝置清單具有分頁回應，則對於每個輪詢，您可以使用 statusCode 參數 傳送個別DEVICE_DISCOVERY操作事件3xx。如果您的裝置探索仍在進行中，請重複步驟 5、6 和 7。

11. 的受管整合 AWS IoT Device Management 會將發現最終使用者裝置的通知傳送給客戶。
12. 如果您的 C2C 連接器傳送DEVICE_DISCOVERY操作事件，並將 statusCode 參數更新為 200，則受管整合會通知客戶裝置探索工作流程完成。

> **Important**
>
> 如有需要，步驟 7 到 11 可以在步驟 6 之前進行。例如，如果您的第三方平台具有 API 來列出最終使用者裝置，則可以在 C2C 連接器 Lambda 回應一般 ACK SendConnectorEvent之前，使用 傳送 DEVICE_DISCOVERY 事件。

裝置探索的 C2C 連接器需求

以下清單概述 C2C 連接器的要求，以促進成功的裝置探索。

- C2C 連接器 Lambda 可以處理來自 AWS IoT Device Management 受管整合的裝置探索請求訊息，並處理AWS.DiscoverDevices操作。
- 您的 C2C 連接器可以使用 AWS 帳戶 用於註冊連接器的 憑證，透過 SigV4 呼叫 AWS IoT Device Management APIs 的受管整合。

裝置探索程序

下列步驟概述使用 C2C 連接器的裝置探索程序，以及 AWS IoT Device Management 的受管整合。

裝置探索程序

1. 受管整合會觸發裝置探索：

   - DiscoverDevices 使用下列 JSON 承載將 POST 請求傳送至 ：

     ```
     /DiscoverDevices
     ```

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

2. 連接器認可探索：

   - 連接器會傳送包含下列 JSON 回應的確認：

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

3. 連接器會傳送裝置探索事件：

   - /connector-event/*{your_connector_id}* 使用下列 JSON 承載將 POST 請求傳送至：

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

為 DISCOVER_DEVICES 事件建構CapabilityReport

如上述定義的事件結構所示，做為 AWS.DiscoverDevices操作回應的 DISCOVER_DEVICES 事件中所報告的每個裝置，都需要 CapbilityReport 來描述對應的裝置功能。`CapabilityReport` 會以符合事項的格式告知 AWS IoT Device Management 裝置功能的受管整合。下列欄位必須在 ` CapabilityReport ` 中提供：

- nodeId、字串：包含下列項目之裝置節點的識別符 endpoints
- version、字串：此裝置節點的版本，由連接器開發人員設定
- endpoints、List<Cluster>：此裝置端點支援的事項資料模型 AWS 實作清單。
  - id、字串：連接器開發人員設定的端點識別符
  - deviceTypes、List<String>：此端點擷取的裝置類型清單，即「攝影機」。
  - clusters、List<Cluster>：此端點支援的事項資料模型 AWS 實作清單。

- `id`、字串：依事項標準定義的叢集識別符。
- `revision`、整數：依事項標準定義的叢集修訂編號。
- `attributes`、Map<String、Object>：屬性識別符及其對應目前裝置狀態值的映射，具有 事件標準定義的識別符和有效值。
  - `id`、字串：Matter Data Model AWS 實作定義的屬性 ID。
  - `value`、物件：屬性 ID 所定義之屬性的目前值。'value' 的類型可能會根據 屬性而變更。每個屬性的 `value` 欄位都是選用的，只有在連接器 Lambda 可以在探索期間判斷目前狀態時才應包含。
- `commands`，List<String>：依事項標準所定義，支援此叢集的命令 IDs 清單。
- `events`、List<String>：依事項標準所定義，支援此叢集的事件 IDs 清單。

如需目前支援的功能清單及其對應的AWS 事項資料模型實作，請參閱最新版本的資料模型文件。

實作 AWS.SendCommand 操作

`AWS.SendCommand` 此操作允許 AWS IoT Device Management 的受管整合，透過 AWS 客戶將最終使用者啟動的命令傳送至您的資源伺服器。您的資源伺服器可能支援多種類型的裝置，其中每種類型都有自己的回應模型。命令執行是一種非同步程序，其中 AWS IoT Device Management 的受管整合會使用 `traceId` 傳送命令執行的請求，您的連接器會將此請求包含在透過 `SendConnectorEvent` API 傳回至受管整合的命令回應中。AWS IoT Device Management 的受管整合預期資源伺服器會傳回確認收到該命令的回應，但不一定表示該命令已執行。

下圖說明命令執行流程，其中包含最終使用者嘗試開啟其房屋照明的範例：

裝置命令執行工作流程

1. 最終使用者傳送命令，以使用 AWS 客戶的應用程式開啟燈光。
2. 客戶將命令資訊轉送至 AWS IoT Device Management 的受管整合與最終使用者的裝置資訊。
3. 受管整合會產生 "traceId"，您的連接器會在將命令回應傳回服務時使用。
4. AWS IoT Device Management 的 受管整合會使用 `AWS.SendCommand`操作界面，將命令請求傳送至您的連接器。
   - 此界面定義的承載包含裝置識別符、建構為事項endpoints/clusters/commands的裝置命令、最終使用者的存取權杖，以及其他必要的參數。
5. 連接器存放`traceId`要包含在命令回應中的 。

- 連接器會將受管整合命令請求轉譯為資源伺服器的適當格式。

6. 連接器`UserId`會從提供的最終使用者的存取權杖取得，並將其與 命令建立關聯。

   a. 在 JWT 和類似字符的情況下， `UserId` 可能會使用單獨的呼叫從您的資源伺服器擷取，或從存取字符中擷取。

   b. 實作取決於您的資源伺服器和存取權杖詳細資訊。

7. 連接器會將資源伺服器呼叫「開啟」最終使用者的光源。
8. 資源伺服器會與裝置互動。

   a. 連接器會轉送至資源伺服器已交付命令的 AWS IoT Device Management 受管整合，並以 ACK 做為初始同步命令回應。

   b. 受管整合接著會將其轉送回客戶應用程式。

9. 裝置開啟燈光後，資源伺服器會擷取該裝置事件。
10. 您的資源伺服器會將裝置事件傳送至連接器。
11. 連接器會將資源伺服器產生的裝置事件轉換為受管整合 DEVICE_COMMAND_RESPONSE 事件操作類型。
12. 連接器會呼叫操作為 "DEVICE_COMMAND_RESPONSE" 的 `SendConnectorEvent` API。

    - 它會在初始請求中連接 AWS IoT Device Management 的受管整合`traceId`所提供的 。

13. 受管整合會通知客戶有關最終使用者的裝置狀態變更。
14. 客戶通知最終使用者裝置指示燈已開啟。

> **Note**
>
> 您的資源伺服器組態會決定處理失敗裝置命令請求和回應訊息的邏輯。這包括使用命令的相同 referenceId 進行訊息重試嘗試。

裝置命令執行的 C2C 連接器需求

下列清單概述 C2C 連接器的要求，以促進成功的裝置命令執行。

- C2C 連接器 Lambda 可以處理來自 AWS IoT Device Management 受管整合`AWS.SendCommand`的操作請求訊息。
- 您的 C2C 連接器必須追蹤傳送至資源伺服器的命令，並將其映射至適當的 `traceId`。

- 您可以使用 AWS 帳戶 用於註冊 C2C 連接器的 AWS 登入資料，透過 SigV4 呼叫 AWS IoT Device Management 服務 API 的 受管整合。

1. 受管整合會將命令傳送至連接器 （ 請參閱先前圖表中的步驟 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 連接器 ACK 命令 （ 請參閱先前圖表中的步驟 7，其中連接器會將 ACK 傳送至 AWS IoT Device Management Service 的受管整合 ）。

   ```
   {
        "header":{
             "responseCode":200
        },
        "payload":{
   ```

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

3. 連接器傳送 Device Command Response 事件 （ 請參閱上圖中的步驟 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**
>
> 在透過 SendConnectorEvent API 收到對應的 DEVICE_COMMAND_RESPONSE 事件之前，由於命令執行而導致的裝置狀態變更不會反映在 AWS IoT Device Management 的受
  ```

管整合中。這表示在受管整合收到先前步驟 3 的事件之前，無論您的連接器調用回應是否表示成功，裝置狀態都不會更新。

解譯 AWS.SendCommand 請求中包含的事項「端點」

受管整合將使用裝置探索期間回報的裝置功能，來判斷裝置可接受哪些命令。每個裝置功能都透過事項資料模型的 AWS 實作進行建模；因此，所有傳入的命令都將衍生自指定叢集中的 `commands` 欄位。您的連接器必須負責剖析 `endpoints` 欄位、判斷對應的事項命令，以及將其翻譯為正確的命令到達裝置。一般而言，這表示將事項資料模型轉換為相關的 API 請求。

執行命令後，您的連接器接著會判斷事項資料模型 AWS 實作定義的 `attributes` 已變更。然後，這些變更會透過使用 API 傳送的 API DEVICE_COMMAND_RESPONSE 事件，回報給 AWS IoT Device Management 的受管整合`SendConnectorEvent`。

請考慮下列範例`AWS.SendCommand`承載中包含的 `endpoints` 欄位：

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

從此物件中，連接器可以判斷下列項目：

1. 設定端點和叢集資訊：

    a. 將端點`id`設定為 "1"。

    > **Note**
    >
    > 如果裝置定義多個端點，讓單一叢集（例如開啟/關閉）可以控制多個功能（即開啟/關閉燈光，以及開啟/關閉頻閃），則會使用此 ID 將命令路由至正確的功能。

b. 將叢集id設定為 "0x0202"（風扇控制叢集）。

2. 設定命令資訊：

   a. 將命令識別符設定為 "0xff01"（更新狀態命令由 定義 AWS)。

   b. 使用請求中提供的值更新包含的屬性識別符。

3. 更新 屬性：

   a. 將屬性識別符設定為 "0x0000" (FanMode 的 Fan Control Cluster 屬性）。

   b. 將屬性值設定為 "3"（高風扇速度）。

受管整合已定義兩種「自訂」命令類型，這些類型並非由事項資料模型的 AWS 實作嚴格定義：ReadState 和 UpdateState 命令。若要取得並設定事項定義的叢集屬性，受管整合會傳送AWS.SendCommand請求給您的連接器，其中包含與 UpdateState (id： 0xff01) 或 ReadState (id：0xff02) 相關的命令 IDs，以及必須更新或讀取的屬性對應參數。對於從Matter Data Model 的對應 AWS 實作中設定為可變（可更新）或可擷取（可讀取）的屬性，可以針對任何裝置類型叫用這些命令。

使用 SendConnectorEvent API 傳送裝置事件

裝置啟動的事件概觀

雖然 SendConnectorEvent API 用於非同步回應 AWS.SendCommand和 AWS.DiscoverDevices操作，但它也用於通知受管整合任何裝置啟動的事件。裝置啟動的事件可以定義為裝置在沒有使用者啟動命令的情況下產生的任何事件。這些裝置事件可能包括但不限於裝置狀態變更、動作偵測、電池電量等。您可以使用 SendConnectorEvent API 搭配操作 DEVICE_EVENT，將這些事件傳回至受管整合。

下節使用安裝在家中的智慧型攝影機做為範例，進一步說明這些事件的工作流程：

裝置事件工作流程

1. 您的攝影機會偵測動作，其會產生傳送至資源伺服器的 事件。
2. 您的資源伺服器會處理事件，並將其傳送至 C2C 連接器。
3. 連接器會將此事件轉譯為 AWS IoT Device Management DEVICE_EVENT 介面的受管整合。
4. 您的 C2C 連接器會使用操作設為 "DEVICE_EVENT" 的 SendConnectorEvent API，將此裝置事件傳送至受管整合。

5. 受管整合可識別相關客戶，並將此事件轉傳給客戶。
6. 客戶會收到此事件，並透過使用者識別符將其顯示給使用者。

如需 SendConnectorEvent API 操作的詳細資訊，請參閱《AWS IoT Device Management API 參考指南》SendConnectorEvent中的 受管整合。

裝置啟動的事件需求

以下是裝置啟動事件的一些需求。

- 您的 C2C 連接器資源應該能夠從資源伺服器接收非同步裝置事件
- 您的 C2C 連接器資源應該可以使用 AWS 帳戶 用於註冊 C2C 連接器的 AWS 憑證，透過 SigV4 呼叫 AWS IoT Device Management 服務 API 的受管整合。

下列範例示範透過 SendConnectorEvent API 傳送裝置來源事件的連接器：

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

從下列範例中，我們看到以下內容：

- 這是來自 ID 等於 1 的裝置端點。
- 此事件相關的裝置功能，其叢集 ID 為 0x0202，與 Fan Control 事件叢集相關。
- 已變更的 屬性具有 0x000 的 ID，與叢集內的風扇模式列舉相關。它已更新為 值 3，與 High 值相關。
- 由於 connectorId 是雲端服務在建立時傳回的參數，因此 Connectors 必須使用 GetCloudConnector 查詢並依 篩選lambdaARN。使用 Lambda.get_function_url_config API ARN 查詢 Lambda 自己的 。這可讓 在 lambda 中CloudConnectorId動態存取 ，而不是像先前一樣靜態設定。

實作 AWS.DeactivateUser 操作

使用者停用概觀

當客戶刪除其 AWS 客戶帳戶，或最終使用者想要取消其系統中帳戶 AWS 與客戶系統的連結時，需要停用提供的使用者存取權杖。在任一種使用案例受管整合中，都需要使用 C2C 連接器來促進此工作流程。

下圖說明從系統取消連結最終使用者帳戶

使用者停用工作流程

1. 使用者會在 AWS 客戶帳戶與與 C2C 連接器相關聯的第三方授權伺服器之間啟動取消連結程序。
2. 客戶透過 AWS IoT Device Management 的受管整合啟動刪除使用者的關聯。
3. 受管整合會使用AWS.DeactivateUser操作界面，透過對連接器的請求啟動停用程序。
    - /user 的存取權杖包含在請求的 標頭中。
4. 您的 C2C 連接器接受請求，並叫用您的授權伺服器來撤銷權杖及其提供的任何存取權。
    - 例如，執行 後，來自未連結使用者帳戶的事件不應再傳送至受管整合AWS.DeactivateUser。
5. 您的授權伺服器會撤銷存取權，並將回應傳回 C2C 連接器。
6. 您的 C2C 連接器會傳送 AWS IoT Device Management 的受管整合 ACK，說明使用者的存取權杖已被撤銷。
7. 受管整合會刪除最終使用者擁有且與您的資源伺服器相關聯的所有資源。
8. 受管整合會將 ACK 傳送給客戶，指出與您的系統相關的所有關聯都會遭到刪除。

9. 客戶會通知最終使用者其帳戶已從您的平台取消連結。

AWS.DeactivateUser 需求

- C2C 連接器 Lambda 函數會從受管整合接收請求訊息，以處理`AWS.DeactivateUser`操作。
- C2C 連接器必須撤銷提供的 OAuth2.0 權杖，以及授權伺服器內使用者的對應重新整理權杖。

以下是連接器將收到的範例`AWS.DeactivateUser`請求：

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

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

叫用 C2C 連接器

AWS Lambda 允許以資源為基礎的政策授權誰可以叫用 Lambda。由於 AWS IoT Device Management 的受管整合是 AWS 服務，因此您必須允許受管整合透過資源政策調用 C2C 連接器 Lambda。

將至少具有下列最低許可的資源政策連接至 C2C 連接器 Lambda。這提供與 Lambda 函數叫用權限的受管整合。此政策包含 `Condition`金鑰，可協助您將 的可用性限制`connectorId`為僅限預期使用者。

JSON

```
{
  "Version":"2012-10-17",
```

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

將許可新增至您的 IAM 角色

所有受管整合 APIs 都需要 AWS sigV4 身分驗證才能叫用。SigV4 正在簽署通訊協定，以使用您的 AWS 帳戶 登入資料來驗證 AWS API 請求。您用來叫用受管整合 APIs 的 IAM 角色必須具有下列許可，才能成功叫用 APIs：

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

如需有關新增這些許可的其他資訊，請聯絡 支援。

其他資源

若要註冊 C2C 連接器，您需要下列項目：

- 指定您要註冊之連接器的 Lambda ARN。

手動測試 C2C 連接器

若要end-to-end手動測試 C2C 連接器，您必須模擬客戶和最終使用者。

您需要下列資源：

- 指定您要測試之連接器的 AWS Lambda ARN。
- 從雲端平台測試 OAuth 2.0 使用者帳戶。
- 向 AWS IoT Device Management 受管整合註冊的連接器。如需詳細資訊，請參閱使用 C2C Cloud-to-Cloud) 連接器。

使用 C2C Cloud-to-Cloud) 連接器

C2C 連接器可管理請求和回應訊息的轉譯，並啟用受管整合與第三方廠商雲端之間的通訊。它有助於跨不同的裝置類型、平台和通訊協定進行統一控制，以便加入和管理第三方裝置。

下列程序列出使用 C2C 連接器的步驟。

使用 C2C 連接器的步驟：

1. CreateCloudConnector

   設定連接器以啟用受管整合和第三方供應商雲端之間的雙向通訊。

   設定連接器時，請提供下列詳細資訊：

   - 名稱：選擇連接器的描述性名稱。
   - 描述：提供連接器目的和功能的簡短摘要。
   - AWS Lambda ARN：指定將為連接器供電之 AWS Lambda 函數的 Amazon Resource Name (ARN)。

建置和部署 AWS Lambda 函數，以與第三方廠商 APIs通訊來建立連接器。接著，在受管整合中呼叫 CreateCloudConnector API，並提供用於註冊的 AWS Lambda 函數 ARN。確保 AWS Lambda 函數部署在您在受管整合中建立連接器的相同 AWS 帳戶中。您將獲指派一個唯一的連接器 ID 來識別整合。

CreateCloudConnector API 請求和回應範例：

```
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"
}
```

建立流程：

> **Note**
>
> 視需要使用 GetCloudConnector、UpdateCloudConnector、DeleteCloudConnector 和 ListCloudConnectors APIs。

2. CreateConnectorDestination

設定目的地以提供連接器與第三方廠商雲端建立安全連線所需的設定和身分驗證登入資料。使用目的地向受管整合註冊第三方身分驗證憑證，例如 OAuth 2.0 授權詳細資訊，包括授權 URL、身分驗證機制，以及登入資料在其中的位置 AWS Secrets Manager。

先決條件

建立 ConnectorDestination 之前，您必須：

- 呼叫 CreateCloudConnector API 來建立連接器。函數傳回的 ID 用於 CreateConnectorDestination API 呼叫。
- 擷取連接器 `tokenUrl` 3P 平台的 。（您可以將 authCode 交換為 accessToken)。
- 擷取連接器 3P 平台的 authUrl。（最終使用者可以使用其使用者名稱和密碼進行身分驗證）。
- 在帳戶的秘密管理員中使用 `clientId`和 `clientSecret`（從 3P 平台）。

CreateConnectorDestination API 請求和回應範例：

```
Request:

{
    "Name": "CreateConnectorDestination",
    "Description": "CreateConnectorDestination",
    "AuthType": "OAUTH",
    "AuthConfig": {
        "oAuth": {
            "authUrl": "https://xxxx.com/oauth2/authorize",
            "tokenUrl": "https://xxxx/oauth2/token",
            "scope": "testScope",
            "tokenEndpointAuthenticationScheme": "HTTP_BASIC",
            "oAuthCompleteRedirectUrl": "about:blank",
            "proactiveRefreshTokenRenewal": {
                "enabled": false,
                "DaysBeforeRenewal": 30
            }
        }
    },
    "CloudConnectorId": "<connectorId>", // The connectorID instance from response
 of Step 1.
    "SecretsManager": {
        "arn": "arn:aws:secretsmanager:*****:secret:*******",
        "versionId": "********"
```

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

Response:

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

雲端目的地建立流程：

> **Note**
>
> 視需要使用 GetCloudConnector、UpdateCloudConnector、DeleteCloudConnector 和 ListCloudConnectors APIs。

3. CreateAccountAssociation

關聯代表最終使用者的第三方雲端帳戶與連接器目的地之間的關係。建立關聯並將最終使用者連結至受管整合之後，即可透過唯一的關聯 ID 存取其裝置。此整合可啟用三個關鍵函數：探索裝置、傳送命令和接收事件。

先決條件

在建立 AccountAssociation 之前，您必須完成下列操作：

- 呼叫 CreateConnectorDestination API 來建立目的地。函數傳回的 ID 用於 CreateAccountAssociation API 呼叫。
- 叫用 CreateAccountAssociation API。

CreateAccountAssociation API 請求和回應範例：

```
Request:

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

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

Response:

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

> **Note**
>
> 視需要使用 GetCloudConnector、UpdateCloudConnector、DeleteCloudConnector 和 ListCloudConnectors APIs。

AccountAssociation 具有從 GetAccountAssociation 和 ListAccountAssociations APIs 查詢的狀態。這些 APIs會顯示關聯的狀態。StartAccountAssociationRefresh API 允許在其重新整理權杖過期時重新整理 AccountAssociation 狀態。

4. 裝置探索

每個受管物件都會連結到裝置特定的詳細資訊，例如其序號和資料模型。資料模型說明裝置的功能，指出裝置是燈泡、開關、調溫器或其他類型的裝置。若要探索 3P 裝置並為 3P 裝置建立 managedThing，您必須依序遵循下列步驟。

a. 呼叫 StartDeviceDiscovery API 以啟動裝置探索程序。

StartDeviceDiscovery API 請求和回應範例：

```
Request:

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

Response:

{
```

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

b. 叫用 GetDeviceDiscovery API 來檢查探索程序的狀態。

c. 叫用 ListDiscoveredDevices API 來列出探索到的裝置。

範例 ListDiscoveredDevices API 請求和回應：

```
Request:

//Empty body

Response:

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

d. 叫用 CreateManagedThing API，從探索清單中選擇要匯入受管整合的裝置。

CreateManagedThing API 請求和回應範例：

```
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. 叫用 GetManagedThing API 來檢視這個新建立的 managedThing。狀態將為 UNASSOCIATED。

f. 叫用 RegisterAccountAssociation API 將此managedThing與特定 建立關聯accountAssociation。在成功的 RegisterAccountAssociation API 結束時，狀態會managedThing變更為 ASSOCIATED 狀態。

RegisterAccountAssociation API 請求和回應範例：

```
Request:

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

Response:

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

5. 將命令傳送至 3P 裝置

若要控制新加入的裝置，請使用 SendManagedThingCommand API，搭配先前建立的關聯 ID 和以裝置支援的功能為基礎的控制動作。連接器使用來自帳戶連結程序的預存登入資料來驗證第三方雲端，並叫用 操作的相關 API 呼叫。

SendManagedThingCommand API 請求和回應範例：

```
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"
}
```

將命令傳送至 3P 裝置流程：

6. 連接器將事件傳送至受管整合

   SendConnectorEvent API 會擷取從連接器到受管整合的四種事件類型，以操作類型參數的下列列舉值表示：

   - DEVICE_COMMAND_RESPONSE：連接器為了回應命令而傳送的非同步回應。

- DEVICE_DISCOVERY：為了回應裝置探索程序，連接器會將探索的裝置清單傳送至受管整合，並使用 SendConnectorEvent API。
- DEVICE_EVENT：傳送收到的裝置事件。
- DEVICE_COMMAND_REQUEST：從裝置啟動的命令請求。例如，WebRTC 工作流程。

連接器也可以使用 SendConnectorEvent API 搭配選用`userId`參數轉送裝置事件。

- 對於具有 的裝置事件`userId`：

  SendConnectorEvent API 請求和回應範例：

```
Request:

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

Response:

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

- 對於沒有 的裝置事件`userId`：

  SendConnectorEvent API 請求和回應範例：

```
Request:
```

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

Response:

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

若要移除特定 managedThing與帳戶關聯之間的連結，請使用取消註冊機制：

DeregisterAccountAssociation API 請求和回應範例：

```
Request:

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

Response:

HTTP/1.1 200 // Empty body
```

傳送事件流程：

7. 將連接器狀態更新為「已列出」，讓其他受管整合客戶可以看到

根據預設，連接器是私有的，只有建立連接器 AWS 的帳戶才能看見。您可以選擇讓其他受管整合客戶可以看到連接器。

若要與其他使用者共用您的連接器，請使用連接器詳細資訊頁面上 AWS 管理主控台 的讓連接器 ID 可見選項，將連接器 ID 提交至 AWS 以供檢閱。核准後，連接器可供相同 中的所有受管整合使用者使用 AWS 區域。此外，您可以透過修改連接器相關聯 AWS Lambda 函數上的存取政策，限制對特定 AWS 帳戶 IDs 的存取。為了確保您的連接器可供其他客戶使用，請管理 Lambda 函數上從其他 AWS 帳戶到可見連接器的 IAM 存取許可。

在讓其他受管整合客戶看見連接器之前，請檢閱管理連接器共用和存取許可 AWS 服務 的條款和組織的政策。

受管整合中樞 SDK

使用本節中的主題，了解如何使用 受管整合中樞 SDK 來加入和控制 IoT 中樞裝置。如需受管整合終端裝置 SDK 的詳細資訊，請參閱 受管整合 終端裝置 SDK。

Hub SDK 架構

裝置加入

在您開始使用受管整合之前，請先檢閱 Hub SDK 元件如何支援裝置加入。本節涵蓋裝置加入所需的基本架構元件，包括核心佈建器和通訊協定特定的外掛程式如何一起運作，以處理裝置身分驗證、通訊和使用者設定。

用於裝置加入的中樞 SDK 元件

SDK 元件

- 核心佈建器
- 通訊協定特定的佈建器外掛程式
- 通訊協定特定的中介軟體

核心佈建器

核心佈建器是協調 IoT 中樞部署中裝置加入的中央元件。它會協調受管整合與通訊協定特定佈建器外掛程式之間的所有通訊，確保安全可靠的裝置加入。當您加入裝置時，核心佈建器會處理身分驗證流程、管理 MQTT 訊息，以及透過這些函數處理裝置請求：

MQTT 連線

建立與 MQTT 代理程式的連線，以進行雲端主題發佈和訂閱。

訊息佇列和處理常式

依序處理傳入的新增和移除裝置請求。

通訊協定外掛程式界面

透過管理身分驗證和無線電聯結模式，使用適用於裝置加入的通訊協定特定佈建器外掛程式。

Hub SDK APIs

從通訊協定特定的 CDMB 外掛程式接收裝置功能報告並將其轉送至受管整合。

通訊協定特定的佈建器外掛程式

通訊協定特定的佈建程式外掛程式是管理不同通訊協定之裝置加入的程式庫。每個外掛程式會將核心佈建器的命令轉換為 IoT 裝置的通訊協定特定動作。這些外掛程式會執行：

- 通訊協定特定的中介軟體初始化
- 根據核心佈建器請求的無線電聯結模式組態
- 透過中介軟體 API 呼叫移除裝置

通訊協定特定的中介軟體

通訊協定特定的中介軟體可做為裝置通訊協定與受管整合之間的轉譯層。此元件處理雙向通訊：從佈建器外掛程式接收命令並將其傳送至通訊協定堆疊，同時收集來自裝置的回應，並透過系統路由回這些回應。

裝置加入流程

檢閱使用 Hub SDK 加入裝置時發生的操作順序。本節顯示元件在加入過程中如何互動，並概述支援的加入方法。

加入流程

- 簡單設定 (SS)
- 零接觸設定 (ZTS)
- 使用者引導設定 (UGS)

簡單設定 (SS)

最終使用者開啟 IoT 裝置的電源，並使用裝置製造商應用程式掃描其 QR 碼。然後，裝置會註冊到受管整合雲端，並連接到 IoT 中樞。

零接觸設定 (ZTS)

零接觸設定 (ZTS) 透過預先關聯供應鏈中的上游裝置，簡化裝置加入。例如，完成此步驟之前會將裝置預先連結至客戶帳戶，而不是掃描裝置 QR 碼的最終使用者。例如，此步驟可在履行中心完成。

當最終使用者接收裝置並開啟電源時，它會自動在受管整合雲端中註冊，並連接到 IoT 中樞，而不需要任何額外的設定動作。

使用者引導設定 (UGS)

最終使用者會開啟裝置的電源，並遵循互動式步驟將其加入受管整合。這可能包括按下 IoT 中樞上的按鈕、使用裝置製造商應用程式，或同時按下中樞和裝置上的按鈕。如果簡易設定失敗，您可以使用此方法。

裝置控制

受管整合會處理裝置註冊、命令執行和控制。您可以使用其廠商和與通訊協定無關的裝置管理，在不知道裝置特定通訊協定的情況下建立最終使用者體驗。

透過裝置控制，您可以檢視和修改裝置狀態，例如燈泡亮度或門位置。此功能會針對狀態變更發出事件，您可以將其用於分析、規則和監控。

主要功能

修改或讀取裝置狀態

根據裝置類型檢視和變更裝置屬性。您可以存取：

- 裝置狀態：目前的裝置屬性值
- 連線狀態：裝置連線能力狀態
- 運作狀態：系統值，例如電池電量和訊號強度 (RSSI)

狀態變更通知

當裝置屬性或連線狀態變更時，接收事件，例如燈泡亮度調整或門鎖狀態變更。

離線模式

裝置即使沒有網際網路連線，也會與相同 IoT 中樞上的其他裝置通訊。裝置狀態會在連線恢復時與雲端同步。

狀態同步

追蹤來自多個來源、裝置製造商應用程式和手動裝置調整的狀態變更。

檢閱透過受管整合控制裝置所需的 Hub SDK 元件和程序。本主題說明 Edge Agent、通用資料模型橋接器 (CDMB) 和通訊協定特定的外掛程式如何一起運作，以處理裝置命令、管理裝置狀態，以及處理不同通訊協定的回應。

裝置控制流程

下圖說明最終使用者如何開啟 Zigbee 智慧插頭，以示範end-to-end裝置控制流程。

用於裝置控制的中樞 SDK 元件

Hub SDK 架構使用下列元件，在您的 IoT 實作中處理和路由裝置控制命令。每個元件在將雲端命令轉譯為裝置動作、管理裝置狀態和處理回應方面都扮演特定角色。下列各節詳細說明這些元件如何在您的部署中一起運作：

Hub SDK 包含下列元件，並促進 IoT 中樞上的裝置加入和控制。

主要元件：

Edge 代理程式

做為 IoT 中樞與受管整合之間的閘道。

通用資料模型橋接器 (CDMB)

在 AWS 資料模型和本機通訊協定資料模型之間進行轉譯，例如 Z-Wave 和 Zigbee。它包含核心 CDMB 和通訊協定特定的 CDMB 外掛程式。

佈建器

處理裝置探索和加入。它包含適用於通訊協定特定加入任務的核心佈建器和通訊協定特定的佈建器外掛程式。

次要元件

Hub 加入

使用用戶端憑證和金鑰佈建中樞，以安全進行雲端通訊。

MQTT 代理

提供與受管整合雲端的 MQTT 連線。

Logger

在本機或受管整合雲端寫入日誌。

安裝和驗證受管整合 Hub SDK

選擇下列部署方法，在您的裝置上安裝受管整合 Hub SDK，AWS IoT Greengrass 以進行自動化部署或手動指令碼安裝。本節說明這兩種方法的設定和驗證步驟。

部署方法

- 使用 安裝 Hub 開發套件 AWS IoT Greengrass
- 使用指令碼部署 Hub SDK
- 使用 systemd 部署 Hub SDK

使用 安裝 Hub 開發套件 AWS IoT Greengrass

使用 AWS IoT Greengrass (Java 版本 ） 部署裝置的受管整合 Hub SDK 元件。

> **Note**
>
> 您必須已設定 並了解 AWS IoT Greengrass。如需詳細資訊，請參閱AWS IoT Greengrass 開發人員指南文件中的內容 AWS IoT Greengrass。

AWS IoT Greengrass 使用者必須具有修改下列目錄的許可：

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

主題

- 在本機部署元件
- 雲端部署

- 驗證中樞佈建
- 驗證 CDMB 操作
- 驗證 LPW-Provisioner 操作

在本機部署元件

在您的裝置上使用 CreateDeployment AWS IoT Greengrass API 來部署 Hub SDK 元件。版本編號不是靜態的，可能會根據您當時使用的版本而有所不同。針對 使用下列格式**version**：com.amazon.IoTManagedIntegrationsDevice.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"
```

雲端部署

請依照AWS IoT Greengrass 開發人員指南中的指示執行下列步驟：

1. 將成品上傳至 Amazon S3。
2. 更新配方以包含 Amazon S3 成品位置。
3. 為新元件建立對裝置的雲端部署。

驗證中樞佈建

檢查您的組態檔案以確認成功佈建。開啟 `/data/aws/iotmi/config/iotmi_config.json` 檔案並確認狀態設定為 `PROVISIONED`。

驗證 CDMB 操作

檢查日誌檔案是否有 CDMB 啟動訊息和初始化成功。`####`位置可能會因 AWS IoT Greengrass 安裝的位置而有所不同。

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

範例

```
[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
```

驗證 LPW-Provisioner 操作

檢查日誌檔案是否有 LPW-Provisioner 啟動訊息並成功初始化。####位置可能會因 AWS IoT Greengrass 安裝的位置而有所不同。

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

範例

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

使用指令碼部署 Hub SDK

使用安裝指令碼手動部署受管整合 Hub SDK 元件，然後驗證部署。本節說明指令碼執行步驟和驗證程序。

主題

準備您的環境

在執行 SDK 安裝指令碼之前，請先完成以下步驟：

1. 在 資料夾middleware內建立名為 的artifacts資料夾。
2. 將您的中樞中介軟體檔案複製到 middleware 資料夾。
3. 在啟動 SDK 之前執行初始化命令。

> **Important**
>
> 每次中樞重新啟動後，重複初始化命令。

```
#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
```

執行 Hub SDK 指令碼

導覽至成品目錄並執行start_iotmi_sdk.sh指令碼。此指令碼會以正確的順序啟動中樞 SDK 元件。檢閱下列範例日誌以確認成功啟動：

> **Note**
>
> 您可以在 artifacts/logs 資料夾中找到所有執行中元件的日誌。

```
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----
```

驗證中樞佈建

檢查 中的 iot_provisioning_state 欄位/data/aws/iotmi/config/iotmi_config.json是否設定為 PROVISIONED。

驗證代理程式操作

檢查日誌檔案是否有客服人員啟動訊息和初始化成功。

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

範例

```
[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**
>
> 檢查您的 `artifacts`目錄中是否存在`iotmi.db`資料庫。

驗證 LPW-Provisioner 操作

檢查日誌檔案是否有`LPW-Provisioner`啟動訊息和初始化成功。

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

下列代碼顯示了範例。

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

使用 systemd 部署 Hub SDK

> **Important**
>
> 請遵循 release.tgz 檔案`readme.md`中的 `hubSystemdSetup`目錄以取得最新的更新。

本節說明在 Linux 型中樞裝置上部署和設定服務的指令碼和程序。

概觀

部署程序包含兩個主要指令碼：

- copy_to_hub.sh：在主機電腦上執行 ，將必要的檔案複製到中樞
- setup_hub.sh：在中樞上執行 以設定環境和部署服務

此外， 會systemd/deploy_iotshd_services_on_hub.sh處理程序引導序列和程序許可管理，並由 自動觸發setup_hub.sh。

先決條件

成功部署需要列出的先決條件。

- 中樞提供 systemd 服務
- 中樞裝置的 SSH 存取
- 中樞裝置上的 Sudo 權限
- scp 主機機器上安裝的 公用程式
- sed 主機機器上安裝的 公用程式
- 主機機器上安裝的 unzip 公用程式

檔案結構

檔案結構旨在促進其各種元件的組織和管理，從而實現內容的高效存取和導覽。

```
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 版本 tgz 檔案中，整體檔案結構為：

```
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
```

初始設定

解壓縮 SDK 套件

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

導覽至擷取的目錄並準備套件：

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

新增裝置組態檔案

請依照列出的兩個步驟建立裝置組態檔案，並將其複製到中樞。

1. 新增裝置組態檔案，以建立所需的裝置組態檔案。SDK 會將此檔案用於其 函數。
2. 複製組態檔案，將建立的組態檔案複製到中樞。

將檔案複製到中樞

從主機機器執行部署指令碼：

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

Example範例

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

這會複製：

- 套件檔案 （ 重新命名為中樞上的 package.zip)
- 組態檔案
- 憑證
- 系統化服務檔案

設定中樞

複製檔案後，SSH 會進入中樞並執行設定指令碼：

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

使用者和群組組態

根據預設，我們使用 SDK 元件的使用者中樞和群組中樞。有多種設定方式：

- 使用自訂使用者/群組：

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

- 在執行設定指令碼之前手動建立它們：

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

- 在 中新增命令`setup_hub.sh`。

管理 服務

若要重新啟動所有 服務，請從中樞執行下列指令碼：

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

設定指令碼將建立必要的目錄、設定適當的許可，以及自動部署 服務。如果您不是使用 SSH/SCP，則必須copy_to_hub.sh針對特定部署方法修改 。在部署之前，請確定所有憑證檔案和組態都已正確設定。

將您的中樞加入受管整合

透過設定所需的目錄結構、憑證和裝置組態檔案，設定您的中樞裝置以與受管整合通訊。本節說明中樞加入子系統元件如何搭配使用、存放憑證和組態檔案的位置、如何建立和修改裝置組態檔案，以及完成中樞佈建程序的步驟。

Hub 加入子系統

中樞加入子系統使用這些核心元件來管理裝置佈建和組態：

Hub 加入元件

透過協調中樞狀態、佈建方法和身分驗證資料，管理中樞加入程序。

裝置組態檔案

將基本集線器組態資料存放在裝置上，包括：

- 裝置佈建狀態 （ 佈建或非佈建 ）
- 憑證和金鑰位置
- 身分驗證資訊 其他 SDK 程序，例如 MQTT 代理，請參考此檔案來判斷中樞狀態和連線設定。

憑證處理常式界面

提供用於讀取和寫入裝置憑證和金鑰的公用程式介面。您可以實作此界面來使用：

- 檔案系統儲存
- 硬體安全模組 (HSM)
- 信任的平台模組 (TPM)
- 自訂安全儲存解決方案

MQTT 代理元件

使用 管理device-to-cloud的通訊：

- 佈建的用戶端憑證和金鑰
- 組態檔案中的裝置狀態資訊
- 受管整合的 MQTT 連線

下圖說明中樞加入子系統架構及其元件。如果您不使用 AWS IoT Greengrass，您可以忽略圖表的該元件。

Hub 加入設定

在開始機群佈建加入程序之前，請先為每個中樞裝置完成這些設定步驟。本節說明如何建立受管物件、設定目錄結構，以及設定所需的憑證。

設定步驟

- 步驟 1：註冊自訂端點
- 步驟 2：建立佈建設定檔
- 步驟 3：建立受管物件 （機群佈建）
- 步驟 4：建立目錄結構
- 步驟 5：將身分驗證資料新增至中樞裝置
- 步驟 6：建立裝置組態檔案
- 步驟 7：將組態檔案複製到您的中樞

步驟 1：註冊自訂端點

建立專用通訊端點，讓您的裝置用來與受管整合交換資料。此端點會為所有device-to-cloud訊息建立安全連線點，包括裝置命令、狀態更新和通知。

註冊端點

- 使用 RegisterCustomEndpoint API device-to-managed整合通訊的端點。

  RegisterCustomEndpoint 請求範例

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

  回應：

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

> **Note**
>
> 存放端點地址。您將需要它來進行未來的裝置通訊。

若要傳回端點資訊，請使用 `GetCustomEndpoint` API。

如需詳細資訊，請參閱《 受管整合 API 參考指南》中的 RegisterCustomEndpoint API 和 GetCustomEndpoint API。

步驟 2：建立佈建設定檔

佈建設定檔包含您的裝置連線至受管整合所需的安全登入資料和組態設定。

建立機群佈建設定檔

- 呼叫 CreateProvisioningProfile API 來產生下列項目：
  - 定義裝置連線設定的佈建範本
  - 裝置身分驗證的宣告憑證和私有金鑰

> **Important**
>
> 安全地存放宣告憑證、私有金鑰和範本 ID。您需要這些登入資料，才能將裝置加入受管整合。如果您遺失這些登入資料，則必須建立新的佈建設定檔。

`CreateProvisioningProfile` 範例請求

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

回應：

```
{
"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"
}
```

步驟 3：建立受管物件 （機群佈建）

使用 CreateManagedThing API 為您的中樞裝置建立受管物件。每個中樞都需要具有唯一身分驗證資料的自有受管物件。如需詳細資訊，請參閱 受管整合 API 參考中的 CreateManagedThing API。

當您建立受管物件時，請指定這些參數：

- Role：CONTROLLER針對不支援命令和控制的中樞，將此值設定為 ，否則設定為 DEVICE。
- AuthenticationMaterialType：將此值設定為 WIFI_SETUP_QR_BAR_CODE。
- AuthenticationMaterial：包含下列欄位。您可以使用 UPC或 EAN，但不能同時使用兩者。
  - SN：此裝置的唯一序號
  - UPC：此裝置的通用產品代碼
  - EAN：此裝置的國際文章編號

> **Important**
>
> 每個裝置在其身分驗證資料中都必須有唯一的序號 (SN)。

CreateManagedThing 請求範例：

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

如需詳細資訊，請參閱 受管整合 API 參考中的 CreateManagedThing。

（選用）取得受管物件

受管物件ProvisioningStatus的 必須是 PRE_ASSOCIATED才能繼續。如需 ProvisioningStatus 的詳細資訊，請參閱 裝置佈建。使用 GetManagedThing API 來驗證您的受管物件是否存在，並準備好進行佈建。如需詳細資訊，請參閱 受管整合 API 參考中的 GetManagedThing。

步驟 4：建立目錄結構

為您的組態檔案和憑證建立目錄。根據預設，中樞加入程序會使用 /data/aws/iotmi/config/iotmi_config.json。

您可以在組態檔案中指定憑證和私有金鑰的自訂路徑。本指南使用預設路徑 /data/aws/iotmi/certs。

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

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

步驟 5：將身分驗證資料新增至中樞裝置

將憑證和金鑰複製到您的中樞裝置，然後建立裝置特定的組態檔案。這些檔案會在佈建程序期間，在您的中樞與受管整合之間建立安全通訊。

複製宣告憑證和金鑰

- 將這些身分驗證檔案從 CreateProvisioningProfile API 回應複製到您的中樞裝置：
  - claim_cert.pem：宣告憑證（適用於所有裝置）
  - claim_pk.key：宣告憑證的私有金鑰

  將兩個檔案放在 /data/aws/iotmi/certs目錄中。

> **Important**
>
> 以 PEM 格式儲存憑證和私有金鑰時，請正確處理換行字元，以確保格式正確。對於 PEM 編碼的檔案，新行字元(\n)必須以實際的行分隔符號取代，因為僅儲存逸出的新行之後將無法正確擷取。

> **Note**
>
> 如果您使用安全儲存，請將這些登入資料存放在您的安全儲存位置，而不是檔案系統。如需詳細資訊，請參閱建立用於安全儲存的自訂憑證處理常式。

步驟 6：建立裝置組態檔案

建立包含唯一裝置識別符、憑證位置和佈建設定的組態檔案。軟體開發套件會在中樞加入期間使用此檔案來驗證您的裝置、管理佈建狀態，以及儲存連線設定。

> **Note**
>
> 每個中樞裝置都需要具有唯一裝置特定值的專屬組態檔案。

使用下列程序來建立或修改您的組態檔案，並將其複製到中樞。

- 建立或修改組態檔案 （機群佈建）。

  在裝置組態檔案中設定這些必要欄位：

  - 憑證路徑
    1. `iot_claim_cert_path`：申請憑證的位置 (`claim_cert.pem`)
    2. `iot_claim_pk_path`：私有金鑰的位置 (`claim_pk.key`)
    3. 實作安全儲存憑證處理常式時，`SECURE_STORAGE`針對這兩個欄位使用
  - 連線設定
    1. `fp_template_name`：先前 `ProvisioningProfile`的名稱。

2. endpoint_url：您的受管整合來自 RegisterCustomEndpoint API 回應的端點 URL（與區域中所有裝置相同）。

- 裝置識別符

1. SN：符合您 CreateManagedThing API 呼叫的裝置序號（每個裝置唯一）
2. UPC來自 CreateManagedThing API 呼叫的通用產品代碼（與此產品的所有裝置相同）

```
{
    "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"
    }
}
```

組態檔案的內容

檢閱 iotmi_config.json 檔案的內容。

目錄

金鑰	值	由客戶新增？	備註
iot_provisioning_method	FLEET_PROVISIONING	是	指定您要使用的佈建方法。
iot_claim_cert_path	您指定的檔案路徑 或 SECURE_STORAGE 。例如 /data/aws/	是	指定您要使用 或 的檔案路徑SECURE_STORAGE 。

金鑰	值	由客戶新增？	備註
	iotmi/certs/claim_cert.pem		
iot_claim_pk_path	您指定的檔案路徑 或 SECURE_STORAGE 。例如 /data/aws/iotmi/certs/claim_pk.pem	是	指定您要使用 或 的檔案路徑SECURE_STORAGE 。
fp_template_name	機群佈建範本名稱應等於先前使用的 ProvisioningProfile 名稱。	是	等於先前ProvisioningProfile 使用的 名稱
endpoint_url	受管整合的端點 URL。	是	您的裝置使用此 URL 連線到受管整合雲端。若要取得此資訊，請使用 RegisterCustomEndpoint API。
SN	裝置序號。例如 AIDACKCEVSQ6C2EXAMPLE 。	是	您必須為每個裝置提供此唯一資訊。
UPC	裝置通用產品程式碼。例如 841667145075 。	是	您必須為裝置提供此資訊。
managed_thing_id	受管物件的 ID。	否	此資訊稍後會由中樞佈建後的加入程序新增。
iot_provisioning_state	佈建狀態。	是	佈建狀態必須設定為 NOT_PROVISIONED 。
iot_permanent_cert_path	IoT 憑證路徑。例如 /data/aws/iotmi/iot_cert.pem 。	否	此資訊稍後會由中樞佈建後的加入程序新增。

金鑰	值	由客戶新增？	備註
iot_permanent_pk_path	IoT 私有金鑰檔案路徑。例如 /data/aws/iotmi/iot_pk.pem 。	否	此資訊稍後會由中樞佈建後的加入程序新增。
client_id	將用於 MQTT 連線的用戶端 ID。	否	此資訊稍後會由中樞佈建後的加入程序新增，以供其他元件使用。
mqtt_keep_alive_interval	範圍為 30-1200，單位以秒為單位。預設值為 300.	是	使用此設定 MQTT 連線的持續作用間隔。
event_manager_upper_bound	預設值為 500。	否	此資訊稍後會由中樞佈建後的加入程序新增，以供其他元件使用。

步驟 7：將組態檔案複製到您的中樞

將您的組態檔案複製到 /data/aws/iotmi/config或自訂目錄路徑。您將在加入程序期間提供此HubOnboarding二進位檔路徑。

對於機群佈建

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

加入裝置並在中樞中操作它們

透過建立受管物件並將其連接到您的中樞，將裝置設定為加入受管整合中樞。裝置可以透過簡單設定或使用者引導式設定加入中樞。

主題

輕鬆設定以加入和操作裝置

透過建立受管物件並將其連接到您的中樞，將裝置設定為加入受管整合中樞。本節說明使用簡單設定完成裝置加入程序的步驟。

先決條件

在嘗試加入裝置之前，請先完成以下步驟：

- 將中樞裝置加入受管整合中樞。
- AWS CLI 從 Managed Integrations AWS CLI 命令參考安裝最新版本的
- 訂閱 DEVICE_LIFE_CYCLE 事件通知。

設定步驟

步驟 1：建立登入資料儲存庫

為您的裝置建立登入資料儲存庫。

建立登入資料儲存庫

- 使用 `create-credential-locker` 命令。執行此命令將觸發建立所有製造資源，包括 Wi-Fi 設定金鑰對和裝置憑證。

  create-credential-locker 範例

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

  回應：

  ```
  {
    "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"
  }
  ```

  如需詳細資訊，請參閱 受管整合命令參考中的 create-credential-locker AWS CLI 命令。

步驟 2：將登入資料儲存庫新增至您的中樞

將登入資料儲存貯體新增至您的中樞。

將登入資料儲存貯體新增至您的中樞

- 使用下列命令將登入資料儲存庫新增至您的中樞。

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

步驟 3：使用登入資料建立受管物件。

使用裝置的登入資料建立受管物件。每個裝置都需要自己的受管物件。

建立受管物件

- 使用 `create-managed-thing`命令為您的裝置建立受管物件。

create-managed-thing 範例

```
#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 裝置有不同的命令。

回應：

```
{
  "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"
}
```

如需詳細資訊，請參閱 受管整合命令參考中的 create-managed-thing AWS CLI 命令。

步驟 4：插入裝置並檢查其狀態。

插入裝置並檢查其狀態。

- 使用 `get-managed-thing`命令來檢查裝置的狀態。受管物件的 ProvisioningStatus 必須啟用。如需 ProvisioningStatus 的詳細資訊，請參閱 裝置佈建。

  get-managed-thing 範例

```
#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"
```

回應：

```
{
  "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"
}
```

如需詳細資訊，請參閱 受管整合命令參考中的 get-managed-thing AWS CLI 命令。

步驟 5：取得裝置功能

使用 `get-managed-thing-capabilities`命令來取得您的端點 ID，並檢視裝置的可能動作清單。

取得裝置的功能

- 使用 `get-managed-thing-capabilities`命令並記下端點 ID。

  get-managed-thing-capabilties 範例

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

  回應：

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

  如需詳細資訊，請參閱 受管整合命令參考中的 get-managed-thing-capabilities AWS CLI 命令。

步驟 6：將命令傳送至受管物件

使用 send-managed-thing-command命令將切換動作命令傳送至您的受管物件。

將命令傳送至受管物件

- 使用 send-managed-thing-command命令將命令傳送至您的受管物件。

  send-managed-thing-command 範例

```
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**
>
> 此範例使用 jq cli 至 ，但您也可以傳遞整個endpointId字串

  回應：

```
{
```

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

如需詳細資訊，請參閱 受管整合 AWS CLI 命令參考中的 send-managed-thing-command 命令。

步驟 7：從您的中樞移除受管物件

移除 受管物件以清除您的中樞。

刪除受管物件

- 使用 `delete-managed-thing`命令從裝置中樞移除受管物件。

  delete-managed-thing 範例

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

  如需詳細資訊，請參閱 受管整合命令參考中的 delete-managed-thing AWS CLI 命令。

  > **Note**
  >
  > 如果裝置卡在 `DELETE_IN_PROGRESS` 狀態，請將 `--force`旗標附加到 `delete-managed-thing command`。

  > **Note**
  >
  > 對於 Z-wave 裝置，您需要在執行 命令之後，讓裝置進入配對模式。

使用者引導設定以加入和操作裝置

透過建立 受管物件並將其連接到 中樞，將您的裝置設定為加入受管整合中樞。本節說明使用使用者引導設定完成裝置加入程序的步驟。

先決條件

在嘗試加入裝置之前，請先完成以下步驟：

- 將中樞裝置加入受管整合中樞。
- AWS CLI 從 Managed Integrations AWS CLI 命令參考安裝最新版本的
- 訂閱 DEVICE_DISCOVERY-STATUS 事件通知。

使用者引導的設定步驟

先決條件：在 Z Wave 裝置上啟用配對模式

在 Z 波裝置上啟用配對模式。配對模式會因每個 Z-Wave 裝置而異，因此請參閱裝置的說明以正確設定配對模式。它通常是使用者必須按下的按鈕。

步驟 1：開始裝置探索

為您的中樞啟動裝置探索，以取得用於加入裝置的探索任務 ID。

啟動裝置探索

- 使用 start-device-discovery 命令來取得探索任務 ID。

  start-device-discovery 範例

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

  #For Zwave
  aws iot-managed-integrations start-device-discovery --discovery-type ZWAVE \
```

```
  --controller-identifier HUB_MANAGED_THING \
  --authentication-material-type ZWAVE_INSTALL_CODE \
  --authentication-material 13333

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

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

回應：

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

> **Note**
>
> Z-wave 和 Zigbee 裝置有不同的命令。

如需詳細資訊，請參閱 受管 integrations AWS CLI Command Reference 中的 start-device-discovery API。

步驟 2：查詢探索任務 ID

使用 `list-discovered-devices`命令來取得裝置的身分驗證資料。

查詢您的探索任務 ID

- 使用探索任務 ID 搭配 list-discovered-devices 命令，以取得裝置的身分驗證資料。

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

回應：

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

步驟 3：為您的裝置建立受管物件

使用 create-managed-thing命令為您的裝置建立受管物件。每個裝置都需要自己的受管物件。

建立受管物件

- 使用 create-managed-thing命令為您的裝置建立受管物件。

  create-managed-thing 範例

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

  回應：

  ```
  {
      "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"
    }
  ```

  如需詳細資訊，請參閱 受管整合命令參考中的 create-managed-thing AWS CLI 命令。

步驟 4：查詢受管物件

您可以使用 get-managed-thing命令來檢查受管物件是否已啟用。

查詢受管物件

- 使用 get-managed-thing命令來檢查受管物件的佈建狀態是否設定為 ACTIVATED。如需佈建狀態的詳細資訊，請參閱 裝置佈建。

  get-managed-thing 範例

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

  回應：

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

  如需詳細資訊，請參閱 受管整合命令參考中的 get-managed-thing AWS CLI 命令。

步驟 5：取得受管物件功能

您可以使用 檢視受管物件的可用動作清單get-managed-thing-capabiltiies。

取得裝置的功能

- 使用 get-managed-thing-capabilities命令來取得端點 ID。另請注意可能的動作清單。

  get-managed-thing-capabilities 範例

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

回應：

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

如需詳細資訊，請參閱 受管整合命令參考中的 get-managed-thing-capabilities 命令。 AWS CLI

步驟 6：將命令傳送至受管物件

您可以使用 send-managed-thing-command 命令，將切換動作命令傳送至受管物件。

使用切換動作將命令傳送至受管物件。

- 使用 send-managed-thing-command命令來傳送切換動作命令。

  send-managed-thing-command 範例

```
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**
>
> 此範例使用 jq cli 至 ，但您也可以傳遞整個endpointId字串

回應：

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

如需詳細資訊，請參閱 受管整合 AWS CLI 命令參考中的 send-managed-thing-command 命令。

步驟 7：檢查受管物件狀態

檢查受管物件的狀態，以驗證切換動作是否成功。

檢查受管物件的裝置狀態

- 使用 `get-managed-thing-state`命令來驗證切換動作是否成功。

  get-managed-thing-state 範例

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

  回應：

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

```
  }
```

如需詳細資訊，請參閱 受管整合命令參考中的 get-managed-thing-state 命令。 AWS CLI

步驟 8：從您的中樞移除受管物件

移除 受管物件以清除您的中樞。

刪除受管物件

- 使用 delete-managed-thing 命令移除受管物件。

  delete-managed-thing 範例

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

  如需詳細資訊，請參閱 受管整合命令參考中的 delete-managed-thing AWS CLI 命令。

> **Note**
>
> 如果裝置卡在 DELETE_IN_PROGRESS 狀態，請將 --force旗標附加至 delete-managed-thing命令。

> **Note**
>
> 對於 Z-wave 裝置，您需要在執行 命令之後，讓裝置進入配對模式。

建立用於安全儲存的自訂憑證處理常式

裝置憑證管理在加入受管整合中樞時至關重要。當憑證預設存放在檔案系統中時，您可以建立自訂憑證處理常式，以增強安全性和彈性的憑證管理。

受管整合 終端裝置 SDK 提供憑證處理常式來保護儲存介面，您可以將其實作為共用物件 (.so) 程式庫。建置您的安全儲存實作以讀取和寫入憑證，然後在執行時間將程式庫檔案連結至 HubOnboarding 程序。

API 定義和元件

檢閱下列secure_storage_cert_handler_interface.hpp檔案，以了解實作的 API 元件和需求

主題

API 定義

secure_storage_cert_hander_interface.hpp 的內容

```
/*
    * 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
```

關鍵元件

- CERT_TYPE_T - 中樞上不同類型的憑證。
  - 宣告 - 最初在中樞上的宣告憑證將交換為永久憑證。
  - Kubernetes - 目前未使用。
  - 永久 - 與受管整合端點連線的永久憑證。
- read_cert_and_private_key - (FUNCTION TO BE IMPLEMENTED) 將 中的憑證和金鑰值讀取至參考輸入。此函數必須能夠同時讀取宣告和永久憑證，並依上述憑證類型區分。
- write_permanent_cert_and_private_key - (FUNCTION TO BE IMPLEMENTED) 會將永久憑證和金鑰值寫入所需的位置。

建置範例

將您的內部實作標頭與公有界面 (`secure_storage_cert_handler_interface.hpp`) 分開，以維護乾淨的專案結構。透過此區隔，您可以在建置憑證處理常式時管理公有和私有元件。

> **Note**
>
> 宣告`secure_storage_cert_handler_interface.hpp`為公有。

主題

專案結構

繼承界面

建立繼承界面的具體類別。在個別目錄下隱藏此標頭檔案和其他檔案，以便在建置時輕鬆區分私有和公有標頭。

```
#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
```

實作

實作上述定義的儲存體方案 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;
  }
```

實作界面 中定義的原廠函數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
    )
```

用量

編譯之後，您會有一個`libSecureStorageCertHandler.so`共用物件程式庫檔案及其相關聯的符號連結。將程式庫檔案和符號連結同時複製到 HubOnboarding 二進位檔預期的程式庫位置。

主題

重要考量

- 確認您的使用者帳戶具有 HubOnboarding 二進位和程式`libSecureStorageCertHandler.so`庫的讀取和寫入許可。
- 保留 `secure_storage_cert_handler_interface.hpp`做為您唯一的公有標頭檔案。所有其他標頭檔案應保留在您的私有實作中。
- 驗證您的共用物件程式庫名稱。當您建置時`libSecureStorageCertHandler.so`，HubOnboarding 可能需要檔案名稱中的特定版本，例

如 libSecureStorageCertHandler.so.1.0。使用 ldd命令來檢查程式庫相依性，並視需要建立符號連結。

- 如果您的共用程式庫實作具有外部相依性，請將它們存放在 HubOnboarding 可存取的目錄中，例如 /usr/lib or the iotmi_common目錄。

使用安全儲存

將 iot_claim_cert_path和 同時設定為 iot_claim_pk_path，以更新您的 iotmi_config.json 檔案**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"
  }
}
```

自訂通訊協定外掛程式

您可以使用自訂通訊協定外掛程式，將您的專屬 IoT 通訊協定整合到 AWS IoT Device Management 生態系統的受管整合中。透過定義明確的 SDK 介面，您可以加入裝置、定義功能和處理即時控制流程，同時保持與受管整合和中樞 SDK 元件的完整相容性。

下列清單討論自訂通訊協定外掛程式的主要功能。

資料模型自訂

定義您自己的 AWS 資料模型結構描述，並在佈建流程中將其上傳至受管整合。您可以稍後在工作流程中使用這些結構描述。

彈性的外掛程式實作

- 使用 建置您自己的外掛程式元件Hub SDK 用戶端。

- 實作不同功能的個別外掛程式，例如佈建和控制，或為兩者建立統一的用戶端。
- 維持受管整合資產與中介軟體堆疊等自有資產之間的明確界限，以實現解耦且易於開發的程式碼邏輯實作。

回溯相容性

對於現有客戶，順暢地加入新的自訂通訊協定，同時保持現有無線電類型正常運作。

下圖說明自訂通訊協定外掛程式架構。

Hub SDK 用戶端

Hub SDK 用戶端程式庫可做為受管整合 Hub SDK 與您在相同中樞上執行的自有通訊協定堆疊之間的界面。它會公開一組公有 APIs，以促進通訊協定堆疊與 Device Hub SDK 元件的互動。使用案例包括自訂外掛程式控制、自訂外掛程式佈建器和本機控制器。

主題

- 取得受管整合 Hub SDK
- 關於 Hub SDK 工具組
- 使用 Hub SDK 用戶端建立自訂應用程式
- 執行您的自訂應用程式
- Hub SDK 用戶端 API 參考
- 資料類型

取得受管整合 Hub SDK

Hub SDK 用戶端隨附 受管整合 SDK。從受管整合主控台聯絡我們以存取中樞 SDK。

關於 Hub SDK 工具組

下載後，您會看到一個`IotMI-DeviceSDK-Toolkit`資料夾，其中包含可在應用程式中使用的所有公有標頭檔案和`.so`檔案。受管整合團隊也提供`main.cpp`示範用途的範例，以及您可以直接執行`bin/`的示範應用程式二進位檔。您可以選擇性地將此做為應用程式的起點。

使用 Hub SDK 用戶端建立自訂應用程式

使用下列步驟來建立您的自訂應用程式。

1. 在您的應用程式中包含標頭檔案 (.h) 和共用物件檔案 (.so)。

   您必須在應用程式中包含公有標頭檔案 (.h) 和共用物件檔案 (.so)。對於 .so 檔案，您可以將它們放在 lib 資料夾中。最終配置將與此類似：

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

2. 在主要應用程式中建立 Hub SDK 用戶端。

   a. 在主要應用程式中，您必須先初始化 Hub SDK 用戶端，才能用來處理請求。您可以直接使用建構用戶端`clientId`。

   b. 擁有用戶端之後，您可以將其連線至受管整合裝置 SDK。

      以下是建立 Hub SDK 用戶端以及如何連線的範例。

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

      auto client = std::make_unique<DeviceSDKClient>(your_own_clientId);
      iotmi_statusCode_t status = client->connect();
      ```

      > **Note**
      >
      > *your_own_clientId* 必須與您在使用者引導設定中 start-device-discovery 中指定的相同，或在簡單設定佈建流程中用於 create-managed-thing。

3. 執行下列步驟來發佈和訂閱 。

a. 建立連線後，您現在可以訂閱來自受管整合 Hub SDK 的傳入任務。傳入的任務可以是控制任務或佈建任務。您也需要在收到任務時定義自己的回呼函數，以及您自己的自訂內容，以用於您自己的追蹤目的。

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

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

b. 建立連線之後，您現在可以從應用程式將請求發佈到受管整合中樞 SDK。您可以定義自己的任務訊息類型，針對不同的業務用途使用不同的承載。請求可以同時包含控制請求和佈建請求，類似於訂閱流程。最後，您可以為 指派地址`rspPayload`，以同步方式從受管整合中樞開發套件取得回應。

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

  status = client->iotmi_control_publish_request(&api_payload);

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

4. 建置您自己的 `CMakeLists.txt`，並從那裡建置您的應用程式。最終輸出可以是可執行的二進位檔，例如 `MyFirstApplication`

執行您的自訂應用程式

執行自訂應用程式之前，請完成下列步驟來設定中樞，並啟動受管整合中樞 SDK：

- 遵循 的加入指示將您的中樞加入受管整合。
- 完成 中記錄的安裝程序安裝和驗證受管整合 Hub SDK。

一旦符合先決條件，您就可以執行自訂應用程式。例如：

```
./MyFirstApplication
```

> **Important**
>
> 您必須使用指令碼部署 Hub SDK使用指令碼手動更新 中列出的啟動指令碼，以啟動您自己的應用程式。順序很重要，請勿變更順序。
> 更新以下內容。變更
>
> ```
> ./IotMI-DeviceSDK-Toolkit/bin/DeviceSDKClientDemo >> $LOGS_DIR/
> logDeviceSDKClientDemo_logs.txt &
> ```
>
> 至
>
> ```
> ./MyFirstApplication >> $LOGS_DIR/MyFirstApplication_logs.txtt &
> ```

Hub SDK 用戶端 API 參考

Hub SDK 用戶端 (DeviceSDKClient 類別） 為您的自訂應用程式提供與受管整合 Device SDK 互動的介面。使用此用戶端，您可以執行下列動作：

- 從 受管整合元件訂閱佈建相關和控制相關的任務。
- 將佈建相關和控制相關請求發佈至受管整合元件。

如需 AWS IoT Device Management APIs 受管整合的資訊，請參閱什麼是 AWS Lambda 。

主題

- 用戶端初始化

- 佈建任務訂閱
- 佈建任務發佈
- 控制任務訂閱
- 控制任務發佈
- 記錄功能
- 其他 APIs

用戶端初始化

若要開始使用 DeviceSDKClient，請使用用戶端 ID 初始化它。

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

這會建立具有指定 的新DeviceSDKClient執行個體clientId。clientId 必須符合您向受管整合註冊的 。

參數

clientId （字串） - 此執行個體的用戶端 ID。

```
connect()
```

將DeviceSDKClient執行個體連線至受管整合。

傳回值

- IOTMI_STATUS_OK - 連線成功。
- IOTMI_STATUS_CUSTOM_PLUGIN_CONNECTION_ERROR - 連線至受管整合時發生錯誤。

佈建任務訂閱

使用這些方法來訂閱受管整合元件中的佈建相關任務。

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

從 受管整合元件訂閱佈建相關任務，例如裝置加入和取消佈建。

參數

- `callback` (DeviceSDKClient_SubscriberCallback) - 在收到任務時執行的回呼函數。
- `context` (char*) - 傳遞至回呼函數的自訂內容。

傳回值

- `IOTMI_STATUS_OK` - 訂閱成功。
- `IOTMI_STATUS_CUSTOM_PLUGIN_CLIENT_NOT_CONNECTED` - DeviceSDKClient 執行個體未連線至受管整合。
- `IOTMI_STATUS_CUSTOM_PLUGIN_SUBSCRIBE_ERROR` - 訂閱任務時發生錯誤。

佈建任務發佈

使用這些方法來將與佈建相關的請求發佈至 受管整合元件。

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

將與佈建相關的請求發佈至受管整合元件。例如，裝置加入的事件或取消佈建狀態

參數

`request` (DataModel：：iotmi_client_request_t) - 包含詳細資訊之請求結構的指標。

傳回值

- `IOTMI_STATUS_OK` - 已成功發佈請求。
- `IOTMI_STATUS_CUSTOM_PLUGIN_CLIENT_NOT_CONNECTED` - `DeviceSDKClient`執行個體未連線至受管整合。
- `IOTMI_STATUS_INVALID_PARAMETER` - 請求中的一或多個參數無效。
- `IOTMI_STATUS_INVALID_JSON_OBJECT` - 請求承載不是有效的 JSON 物件。
- `IOTMI_STATUS_NO_MEMORY` - 發生記憶體配置錯誤。

控制任務訂閱

使用這些方法來訂閱受管整合元件中的控制相關任務。

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

訂閱受管整合元件中的控制相關任務（例如，裝置控制請求）。

參數

- `callback` (DeviceSDKClient_SubscriberCallback) - 在收到任務時執行的回呼函數。
- `context` (char) - 傳遞至回呼函數的自訂內容。

傳回值

- `IOTMI_STATUS_OK` - 訂閱成功。
- `IOTMI_STATUS_CUSTOM_PLUGIN_CLIENT_NOT_CONNECTED` - `DeviceSDKClient`執行個體未連線至受管整合。
- `IOTMI_STATUS_CUSTOM_PLUGIN_SUBSCRIBE_ERROR` - 訂閱任務時發生錯誤。

控制任務發佈

使用這些方法來將控制相關請求發佈至 受管整合元件。

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

將控制相關請求發佈至受管整合元件。例如，未經請求的事件、命令請求或裝置狀態查詢。

參數

`request` (DataModel：：iotmi_client_request_t) - 包含詳細資訊之請求結構的指標。

傳回值

- `IOTMI_STATUS_OK` - 已成功發佈請求。
- `IOTMI_STATUS_CUSTOM_PLUGIN_CLIENT_NOT_CONNECTED` - DeviceSDKClient 執行個體未連線至受管整合。
- `IOTMI_STATUS_INVALID_PARAMETER` - 請求中的一或多個參數無效。
- `IOTMI_STATUS_INVALID_JSON_OBJECT` - 請求承載不是有效的 JSON 物件。
- `IOTMI_STATUS_NO_MEMORY` - 發生記憶體配置錯誤。

記錄功能

使用這些方法來實作 受管整合提供的記錄功能。

記錄器初始化

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

您必須先初始化記錄器，才能使用任何記錄功能。

參數

`logger_name` - 您指定的記錄器名稱。預設值為： `MyApplication`

記錄巨集

`LOGGER_LOGD(...)`

在應用程式中使用此巨集進行 DEBUG 層級記錄。

`LOGGER_LOGI(...)`

在應用程式中使用此巨集進行 INFO 層級記錄。

`LOGGER_LOGW(...)`

在應用程式中使用此巨集進行 WARN 層級記錄。

`LOGGER_LOGE(...)`

在應用程式中使用此巨集記錄錯誤層級。

> **Note**
>
> 如需記錄功能的詳細資訊，請參閱 Hub 記錄文件。自訂通訊協定外掛程式完全支援受管整合提供的所有記錄功能。

其他 APIs

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

傳回與 DeviceSDKClient 執行個體相關聯的用戶端 ID。

傳回值

用戶端 ID。

資料類型

本節定義用於自訂通訊協定外掛程式的資料類型。

iotmi_client_request_t

代表要發佈至受管整合元件的請求。

messageType

訊息的類型 (CommonTypes：：C2MIMessageType)。下列清單顯示有效值。

- C2MI_DEVICE_ONBOARDED：指出具有相關承載的裝置加入訊息。
- C2MI_DE_PROVISIONING_PRE_ASSOCIATED_COMPLETE：表示預先關聯裝置的取消佈建任務完成通知。
- C2MI_DE_PROVISIONING_ACTIVATED_COMPLETE：表示已啟動裝置的取消佈建任務完成通知。
- C2MI_DE_PROVISIONING_COMPLETE_RESPONSE：表示取消佈建任務完成回應。
- C2MI_CONTROL_EVENT：表示具有潛在裝置狀態變更的控制事件。
- C2MI_CONTROL_SEND_COMMAND：表示來自本機控制器的控制命令。
- C2MI_CONTROL_SEND_DEVICE_STATE_QUERY：表示來自本機控制器的控制裝置狀態查詢。

reqPayload

請求承載，通常是 JSON 格式的字串。

rspPayload

由受管整合元件填入的回應承載。

iotmi_client_event_t

代表從受管整合元件接收的事件。

event_id

事件的唯一識別符。

長度

事件資料的長度。

data

事件資料的指標，包括 `messageType`。下列清單顯示可能的值。

- `C2MI_PROVISION_UGS_TASK`：表示 UGS 流程的佈建任務。
- `C2MI_PROVISION_SS_TASK`：表示 SimpleSetup 流程的佈建任務。
- `C2MI_DE_PROVISION_PRE_ASSOCIATED_TASK`：表示預先關聯裝置的取消佈建任務。
- `C2MI_DE_PROVISION_ACTIVATED_TASK`：表示已啟動裝置的取消佈建任務。
- `C2MI_DEVICE_ONBOARDED_RESPONSE`：表示裝置加入回應。
- `C2MI_CONTROL_TASK`：表示控制任務。
- `C2MI_CONTROL_EVENT_NOTIFICATION`：表示本機控制器的控制事件通知。

ctx

與事件相關聯的自訂內容。

Hub 控制

Hub 控制是受管整合終端裝置 SDK 的延伸，允許它與 Hub SDK 中的 `MQTTProxy` 元件連接。透過中樞控制，您可以使用結束裝置 SDK 實作程式碼，並透過受管整合雲端將中樞控制為個別裝置。中樞控制 SDK 會在中樞 SDK 中以單獨的套件提供，標記為 `iot-managed-integrations-hub-control-x.x.x`。

主題

先決條件

若要設定中樞控制，您需要下列項目：

- 已加入 Hub SDK 0.4.0 版或更新版本的中樞。
- 從 下載最新版本的結束裝置 SDK AWS 管理主控台。
- 在中樞上執行的 MQTT 代理元件，0.5.0 版或更新版本。

終端裝置 SDK 元件

從結束裝置 SDK 使用下列元件：

- 資料模型的程式碼產生器
- 資料模型處理常式

由於 Hub SDK 已有加入程序和雲端連線，因此您不需要下列元件：

- 佈建者
- PKCS 介面
- 任務處理常式
- MQTT 代理程式

與終端裝置 SDK 整合

1. 遵循資料模型的程式碼產生器中的指示來產生低階 C 程式碼。
2. 遵循整合終端裝置 SDK 中的指示，以：

   a. 設定建置環境

      在 Amazon Linux 2023/x86_64 上建置程式碼做為您的開發主機。安裝必要的建置相依性：

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

   b. 開發硬體回呼函數

      實作硬體回呼函數之前，請先了解 API 的運作方式。此範例使用 On/Off 叢集和 OnOff 屬性來控制裝置函數。如需 API 詳細資訊，請參閱 低階 C-Function APIs。

      ```
      struct DeviceState
      {
        struct iotmiDev_Agent *agent;
      ```

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

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

c. 設定端點和掛接硬體回呼函數

實作函數之後，請建立端點並註冊您的回呼。完成這些任務：

i. 建立裝置代理程式

ii. 為您要支援的每個叢集結構填入回呼函數點

iii. 設定端點並註冊支援的叢集

```
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);
}
```

範例：建置中樞控制

Hub 控制項是 Hub SDK 套件的一部分。中樞控制子套件會標記 ，`iot-managed-integrations-hub-control-x.x.x`並包含與未修改的裝置 SDK 不同的程式庫。

1. 將程式碼產生的檔案移至 `example` 資料夾：

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

2. 若要建置中樞控制，請執行下列命令：

```
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. 使用中樞上的 `MQTTProxy` 元件執行範例，並執行 `HubOnboarding`和 `MQTTProxy`元件。

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

如需資料模型受管整合資料模型，請參閱 。遵循 中的步驟 5 開始使用結束裝置 SDK 設定端點，並管理最終使用者與 iot-managed-integrations 之間的通訊。

支援的範例

下列範例已經過建置和測試：

- iotmi_device_dm_air_purifier_demo
- iotmi_device_basic_diagnostics
- iotmi_device_dm_camera_demo

支援平台

下表顯示支援用於中樞控制的平台。

架構	作業系統	GCC 版本	Binutils 版本
X86_64	Linux	10.5.0	2.37
aarch64	Linux	10.5.0	2.37

啟用 CloudWatch Logs

Hub SDK 提供完整的記錄功能。根據預設，Hub SDK 會將日誌寫入本機檔案系統。不過，您可以利用雲端 API 來設定日誌串流至 CloudWatch Logs，其提供：

- 監控裝置效能：擷取詳細的執行期日誌以進行主動式裝置管理。在整個裝置機群中啟用進階日誌分析和監控
- 故障診斷問題：產生精細的日誌項目以進行快速診斷分析。記錄系統和應用程式層級事件以進行深入調查。
- 彈性且集中式的記錄：遠端日誌管理，無需直接存取裝置。在單一可搜尋的儲存庫中彙總來自多個裝置的日誌。

先決條件

- 將受管裝置加入雲端。如需詳細資訊，請參閱 Hub 加入設定。
- 驗證 Hub 代理程式啟動和成功初始化。如需詳細資訊，請參閱 安裝和驗證受管整合 Hub SDK。

> **Note**
>
> 若要建立記錄組態，請參閱 PutRuntimeLogConfiguration API 以取得詳細資訊。

> **Warning**
>
> 啟用日誌會計入分層配額計量。增加日誌層級將產生更高的訊息量和額外的成本。

設定 Hub SDK 日誌組態

呼叫 API 來設定執行時間日誌組態，以設定中樞 SDK 日誌設定。

Example範例 API 請求

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

RuntimeLogConfigurations 屬性

下列屬性是選用的，可以在 RuntimeLogConfigurations API 中設定。

LogLevel

設定執行時間追蹤的最低嚴重性等級。數值: DEBUG, ERROR, INFO, WARN

預設： WARN （發行的組建）

LogFlushLevel

決定立即資料排清至本機儲存體的嚴重性等級。數值: DEBUG, ERROR, INFO, WARN

預設：DISABLED

LocalStoreLocation

指定執行時間追蹤的儲存位置。預設：/var/log/awsiotmi

- 作用中日誌： /var/log/awsiotmi/ManagedIntegrationsDeviceSdkHub.log
- 輪換日誌：/var/log/awsiotmi/ManagedIntegrationsDeviceSdkHub.N.log(N 表示輪換順序）

LocalStoreFileRotationMaxBytes

當目前的檔案超過指定的大小時， 會觸發檔案輪換。

> **Important**
>
> 為了獲得最佳效率，請將檔案大小保持在 125 KB 以下。超過 125 KB 的值將自動限制。

LocalStoreFileRotationMaxFiles、

設定日誌協助程式允許的輪換檔案數目上限。

UploadLog

控制執行時間追蹤轉移至雲端。日誌存放在 `/aws/iotmanagedintegration` CloudWatch Logs 群組中。

預設：`false`。

UploadPeriodMinutes

定義執行時間追蹤上傳的頻率。預設：5

DeleteLocalStoreAfterUpload

控制上傳後的檔案刪除。預設：`true`

> **Note**
>
> 如果設定為 false，上傳的檔案會重新命名為： `/var/log/awsiotmi/ManagedIntegrationsDeviceSdkHub.uploaded.{uploaded_timestamp}`

日誌檔案範例

請參閱以下 CloudWatch Logs 檔案的範例：

支援的 Zigbee 和 Z-Wave 裝置類型

此頁面列出已經過受管整合測試且受支援的中樞連線裝置類型。受管整合使用者引導設定 (UGS)支援這些裝置的 簡單設定 (SS)和 。

此資料表列出支援的 Zigbee 裝置。

Zigbee 裝置類型	支援的功能
智慧型燈泡/可調光燈/RGB 燈	OnOff、LevelControl、ColorControl
智慧插頭	OnOff
智慧切換	OnOff
LED 長條圖	OnOff、LevelControl、ColorControl
水閥	OnOff
輻射器閥	節溫器、OnOff、計時器
節溫器	節溫器、FanControl、OnOff、計時器
車庫門開啟器	WindowCovering、OnOff、LevelControl
煙霧警示	BooleanState、OnOff、TemperatureMeasurement、Timer、SmokeCOAlarm
動作感應器	BooleanState
佔用/人存在感應器	BooleanState、 OccupancySensing
門窗感應器	BooleanState
漏水感應器	BooleanState
震動感應器	BooleanState
溫度和濕度感應器	TemperatureMeasurement、RelativeHumidityMeasurement

此資料表列出支援的 Z-Wave 裝置。

Z-Wave 裝置類型	支援的功能
智慧燈泡/可調光燈	OnOff、LevelControl
智慧插頭	OnOff
車庫門控制器	OnOff、LevelControl
電表	ElectricalEnergyMeasurement、ElectricalPowerMeasurement
電池	LevelControl
警報器	LevelControl
動作感應器	BooleanState
門窗感應器	BooleanState
漏水感應器	BooleanState
溫度感應器	TemperatureMeasurement
CO 感應器	SmokeCOAlarm
煙霧感應器	SmokeCOAlarm

在 Raspberry Pi 上執行受管整合

> **Note**
>
> 在 Raspberry Pi 上實作 AWS IoT Hub SDK 是一個示範專案，僅用於學習和測試目的，不適用於生產環境。基於本示範的目的，請設定下列組態以利開發：
> AWS 登入資料儲存：僅供示範使用，登入資料和憑證會存放在可存取的位置，以便於測試和開發。生產環境必須使用安全儲存解決方案 AWS Secrets Manager，例如 或 Systems Manager 參數存放區。他們必須實作靜態加密，並遵循 AWS IoT 安全準則。
> 容器權限： 示範會以更高的權限執行，以允許不受限制地存取主機資源並簡化開發工作流程。在生產環境中，容器應以最低的必要權限運作。

網路橋接組態：示範使用網路橋接組態來公開內部網路流量，以便於偵錯和監控。在生產環境中，實作適當的網路隔離和分割，以防止未經授權存取內部網路流量。
USB 裝置許可：啟用不受限制的 USB 裝置存取，以便輕鬆連接開發周邊裝置和測試裝置。針對生產環境，實作嚴格的 USB 裝置控制和驗證，以防止裝置詐騙攻擊。
這些組態可讓您直接進行測試，且不得用於生產環境。部署至生產環境時，請遵循安全最佳實務，以防止主機系統入侵和未經授權存取登入資料。

作為先決條件，您必須先設定 Sonoff Zigbee USB 硬體鎖，才能設定 Raspberry Pi。

Sonoff Zigbee USB 硬體鎖的快閃記憶體韌體

先決條件

- Sonoff Zigbee USB Dongle
- Windows：安裝 CP210x Universal Windows 驅動程式

刷新韌體

1. 下載 Zigbee Dongle Firmware Build 7.4.1.0。
2. 開啟 Silabs Firmware Flasher。
3. 將 Sonoff Zigbee USB Dongle 連接至您的電腦。
4. 捲動並尋找 ZBDongle-E。
5. 選擇連線。
6. 等待裝置連線。
7. 選擇變更韌體。
8. 選取上傳您自己的韌體。
9. 尋找 Zigbee Dongle Firmware Build 7.4.1.0 下載的位置，然後選取它。
10. 按一下 Install (安裝)。
11. 等待韌體安裝。
12. 安裝完成時，選擇繼續。

硬體鎖現在已準備好可供使用。

選擇下列選項，在您的 Raspberry Pi 上執行受管整合中樞 SDK。這兩種方法的設定和驗證步驟如下所示。

主題

Raspberry Pi 上的受管整合中樞 SDK 映像

> **Note**
>
> 在 Raspberry Pi 上實作 AWS IoT Hub SDK 是一個示範專案，僅用於學習和測試目的，不適用於生產環境。基於本示範的目的，請設定下列組態以利開發：
> AWS 登入資料儲存：僅供示範使用，登入資料和憑證會存放在可存取的位置，以便於測試和開發。生產環境必須使用安全儲存解決方案 AWS Secrets Manager，例如 或 Systems Manager 參數存放區。他們必須實作靜態加密，並遵循 AWS IoT 安全準則。
> 容器權限： 示範會以更高的權限執行，以允許不受限制地存取主機資源並簡化開發工作流程。在生產環境中，容器應以最低的必要權限運作。
> 網路橋接組態： 示範使用網路橋接組態來公開內部網路流量，以便於偵錯和監控。在生產環境中，實作適當的網路隔離和分割，以防止未經授權存取內部網路流量。
> USB 裝置許可：啟用不受限制的 USB 裝置存取，以便輕鬆連接開發周邊裝置和測試裝置。針對生產環境，實作嚴格的 USB 裝置控制和驗證，以防止裝置詐騙攻擊。
> 這些組態可讓您直接進行測試，且不得用於生產環境。部署至生產環境時，請遵循安全最佳實務，以防止主機系統入侵和未經授權存取登入資料。

先決條件

部署 Raspberry Pi 映像之前，請先完成這些要求：

- 下載並安裝 Raspberry Pi 成像器。

- 取得 SD 卡。
- 使用 2.4Ghz 64 位元四核心 CPU (8GB RAM) 設定 Raspberry Pi 5。
- 連接 Sonoff Zigbee USB Dongle。
- Sonoff Zigbee USB 硬體鎖的快閃記憶體韌體.
- 連接 Silicon Labs SLUSB001A Dongle。
- 註冊 AWS 帳戶。
- AWS CLI 從 受管整合 AWS CLI 命令參考安裝最新版本的 。

在新的 SD 卡上刷新 Raspberry Pi 映像

使用下列步驟，將受管整合映像快閃至 SD 卡：

1. 下載受管整合 Raspberry Pi Hub SDK 映像。
2. 在桌面上啟動 Raspberry Pi Imager。
3. 將 SD 卡插入電腦的內建 SD 讀卡機或外部 USB 讀卡機。
4. 選取選擇裝置 → Raspberry Pi 5。
5. 選取選擇作業系統 → 使用自訂 → 尋找 IotMI-HubSDK-RPi-Image-v1.0.0.img.gz 檔案 → Open。
6. 選取選擇儲存體 → 選取 SD 讀卡機。
7. 確認您的組態符合下列畫面：

8. 按一下 Next (下一步)。
9. 設定作業系統自訂設定：
   - 主機名稱：選取 raspberrypi。
   - 使用者名稱和密碼：
     - 啟用設定使用者名稱和密碼：
     - 針對使用者名稱：，輸入 hub123456。
     - 對於密碼：，輸入 sh123456。
   - 無線 LAN：
     - 啟用設定無線 LAN。
     - 輸入路由器的 SSID 和密碼。

     範例設定：

    - SSID：iotmi-tplink
    - 密碼：********（至少 8 個字元）
  - 將國家/地區： 設定為 US。
- 設定地區設定：
  - 將時區： 設定為 America/Los Angeles。
  - 將鍵盤配置： 設定為 US。
- SSH：
  - 選擇服務索引標籤。
  - 勾選啟用 SSH。
  - 選擇使用密碼身分驗證。

10.確認作業系統自訂和資料清除的所有快顯視窗。

11.等待寫入程序完成。

12.使用以下畫面確認成功完成：

13.按一下 Continue (繼續)。

14.移除 SD 卡並將其插入 Raspberry Pi。

在 Raspberry Pi 上執行 Hub SDK

在設定的 Raspberry Pi 上啟動 Hub SDK 服務：

1. 將備妥的 SD 卡插入 Raspberry Pi 5 裝置。
2. 將 Sonoff Zigbee USB 加密狗和 Silicon Labs SLUSB001A 加密狗連接到 Raspberry Pi。
3. 開啟 Raspberry Pi 的電源。
4. 確保 Raspberry Pi 和您的電腦（您 SSH 的來源）位於相同的網路上。
5. 使用您在映像部署期間設定的登入資料，將 SSH 傳送至 Raspberry Pi。

```
ssh username@hostname
```

6. 導覽至中樞 SDK 目錄：

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

7. 完成 Hub 加入設定以新增身分驗證和組態資料。

> **Note**
>
> 您必須位於 YUL或 DUB 區域才能執行此步驟。

8. 執行 Hub SDK：

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

系統會顯示下列成功啟動 Hub SDK 的回應：

```
-------Stopping SDK running processes---
-------Starting Hub SDK-------------
-------Creating logs directory----------
Logs directory created.
-------Verifying Middleware paths-------
All middleware libraries exist
-------Verifying Middleware pre reqs---
AIPC and KVstroage directories exist
-------Starting HubOnboarding-----------
-------Starting MQTT Proxy-----------
-------Staring Log Daemon---
-------Starting Event Manager-----------
-------Starting Zigbee Service----------
--Checking Zigbee network information--
-------Starting Zwave Service----------
/data/aws/iotmi/middleware/AceZwave/bin /data/aws/iotmi
/data/aws/iotmi
-------Starting CDMB--------------------
-------Starting Agent--------------------
-------Starting Provisioner--------------
-------Checking SDK status---------------
hub1234+    1780  0.2  0.1 1093936 16368 pts/1   Sl+  16:34   0:00 ./iotmi_mqtt_proxy -
C /data/aws/iotmi/config/iotmi_config.json
Process 'iotmi_mqtt_proxy' is running.
hub1234+    1884  0.0  0.0 236272  2624 pts/1    Sl+  16:34   0:00 ./middleware/
AceCommon/bin/ace_eventmgr
Process 'ace_eventmgr' is running.
hub1234+    1892  9.1  0.1 393040  8352 pts/1    Sl+  16:34   0:04 ./middleware/
AceZigbee/bin/ace_zigbee_service
Process 'ace_zigbee_service' is running.
```

```
hub1234+    1923  0.0  0.1 1570736 12736 pts/1   Sl+  16:34   0:00 ./zwave_svc
Process 'zwave_svc' is running.
hub1234+    1958  0.0  0.0 1067632 5776 pts/1    Sl+  16:34   0:00 ./iotmi_cdmb
Process 'iotmi_cdmb' is running.
hub1234+    2001  0.2  0.2 2017712 21264 pts/1   Sl+  16:35   0:00 ./iotmi_device_agent
Process 'iotmi_device_agent' is running.
hub1234+    2045  0.0  0.1 1457824 12624 pts/1   Sl+  16:35   0:00 ./
iotmi_lpw_provisioner
Process 'iotmi_lpw_provisioner' is running.
hub1234+    1813  0.0  0.0 875152  6848 pts/1    Sl+  16:34   0:00 ./iotmi_log_daemon
Process 'iotmi_log_daemon' is running.
------Successfully Started Hub SDK----
```

後續步驟

成功啟動 Hub SDK 後，請前往 繼續進行裝置加入和管理使用者引導設定以加入和操作裝置。

Raspberry Pi 上的受管整合 Hub SDK Docker 容器

> **Note**
>
> 在 Raspberry Pi 上實作 AWS IoT Hub SDK 是一個示範專案，僅用於學習和測試目的，不適用於生產環境。基於本示範的目的，請設定下列組態以利開發：
> AWS 登入資料儲存：僅供示範使用，登入資料和憑證會存放在可存取的位置，以便於測試和開發。生產環境必須使用安全儲存解決方案 AWS Secrets Manager，例如 或 Systems Manager 參數存放區。他們必須實作靜態加密，並遵循 AWS IoT 安全準則。
> 容器權限： 示範會以更高的權限執行，以允許不受限制地存取主機資源並簡化開發工作流程。在生產環境中，容器應以最低的必要權限運作。
> 網路橋接組態： 示範使用網路橋接組態來公開內部網路流量，以便於偵錯和監控。在生產環境中，實作適當的網路隔離和分割，以防止未經授權存取內部網路流量。
> USB 裝置許可：啟用不受限制的 USB 裝置存取，以便輕鬆連接開發周邊裝置和測試裝置。針對生產環境，實作嚴格的 USB 裝置控制和驗證，以防止裝置詐騙攻擊。
> 這些組態可讓您直接進行測試，且不得用於生產環境。部署至生產環境時，請遵循安全最佳實務，以防止主機系統入侵和未經授權存取登入資料。

先決條件

Docker 容器需要下列先決條件。

- 下載並安裝 Raspberry Pi 成像器。
- 取得 SD 卡。
- 使用 2.4Ghz 64 位元四核心 CPU (8GB RAM) 設定 Raspberry Pi 5。
- 連接 Sonoff Zigbee USB Dongle。
- Sonoff Zigbee USB 硬體鎖的快閃記憶體韌體.
- 連接 Silicon Labs SLUSB001A Dongle。
- 註冊 AWS 帳戶。
- AWS CLI 從 受管整合 AWS CLI 命令參考安裝最新版本的 。
- 使用 IP 地址或主機名稱對 Raspberry Pi 的 SSH 存取。

在 Raspberry Pi 上使用受管整合 Hub SDK Docker 容器

1. 下載受管整合 Raspberry Pi Hub SDK Docker。
2. 使用 SCP 將檔案複製到 Raspberry Pi：

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

3. 透過 SSH 連線至 Raspberry Pi：

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

4. 如果不存在，請安裝 Docker：

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

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

# Log in again
```

5. 如果不存在，請安裝 Docker Compose：

```
# Install Docker Compose
sudo apt-get update
```

```
sudo apt-get install -y docker-compose-plugin
```

6. 解壓縮 Hub SDK 檔案：

```
# Navigate to the home directory
cd

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

7. 導覽至 hub-docker 目錄：

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

8. 完成 Hub 加入設定以設定身分驗證和設定。

> **Note**
>
> 您必須位於 YUL或 DUB 區域才能執行此步驟。

9. 啟動 Docker 容器：

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

預期的輸出：

```
[+] Running 1/1
 # Container iotmi-hubsdk-docker-hubsdk-1  Started
hubsdk-1  | -\-\-\-\-\-\-Checking USB dongles-\-\-\-\-\-\-\-\-\-\-\-\-\-
hubsdk-1  | -\-\-\-\-\-\-Stopping SDK running processes-\-\-
hubsdk-1  | iotmi_mqtt_proxy: no process found
hubsdk-1  | ace_eventmgr: no process found
hubsdk-1  | ace_zigbee_service: no process found
hubsdk-1  | zwave_svc: no process found
hubsdk-1  | iotmi_cdmb: no process found
hubsdk-1  | iotmi_device_agent: no process found
hubsdk-1  | iotmi_lpw_provisioner: no process found
hubsdk-1  | iotmi_log_daemon: no process found
hubsdk-1  | -\-\-\-\-\-\-Starting Hub SDK-\-\-\-\-\-\-\-\-\-\-\-\-\-
```

```
hubsdk-1  | -\-\-\-\-\-\-Creating logs directory-\-\-\-\-\-\-\-\-\-
hubsdk-1  | Logs directory created.
hubsdk-1  | -\-\-\-\-\-\-Verifying Middleware paths-\-\-\-\-\-\-
hubsdk-1  | All middleware libraries exist
hubsdk-1  | -\-\-\-\-\-\-Verifying Middleware pre reqs-\-\-
hubsdk-1  | AIPC and KVstroage directories exist
hubsdk-1  | -\-\-\-\-\-\-Starting HubOnboarding-\-\-\-\-\-\-\-\-\-\-
hubsdk-1  | -\-\-\-\-\-\-Starting MQTT Proxy-\-\-\-\-\-\-\-\-\-\-
hubsdk-1  | -\-\-\-\-\-\-Staring Log Daemon-\-\-
hubsdk-1  | -\-\-\-\-\-\-Starting Event Manager-\-\-\-\-\-\-\-\-\-\-
hubsdk-1  | -\-\-\-\-\-\-Starting Zigbee Service-\-\-\-\-\-\-\-\-\-\-
hubsdk-1  | -\-\-\-\-\-\-Starting Zwave Service-\-\-\-\-\-\-\-\-\-\-
hubsdk-1  | /data/aws/iotmi/middleware/AceZwave/bin /data/aws/iotmi
hubsdk-1  | /data/aws/iotmi
hubsdk-1  | -\-\-\-\-\-\-Starting CDMB-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-
hubsdk-1  | -\-\-\-\-\-\-Starting Agent-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-
hubsdk-1  | -\-\-\-\-\-\-Starting Provisioner-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-
hubsdk-1  | -\-\-\-\-\-\-Checking SDK status-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-\-
hubsdk-1  | root         105  0.2  0.2 1093488 16608 ?       Sl   20:51   0:00 ./
iotmi_mqtt_proxy -C /data/aws/iotmi/config/iotmi_config.json
hubsdk-1  | Process 'iotmi_mqtt_proxy' is running.
hubsdk-1  | root         183  0.0  0.0 236272  3152 ?        Sl   20:51   0:00 ./
middleware/AceCommon/bin/ace_eventmgr
hubsdk-1  | Process 'ace_eventmgr' is running.
hubsdk-1  | root         190 12.0  0.1 319264  8352 ?        Sl   20:51   0:04 ./
middleware/AceZigbee/bin/ace_zigbee_service
hubsdk-1  | Process 'ace_zigbee_service' is running.
hubsdk-1  | root         200  0.0  0.1 1365792 12480 ?       Sl   20:51   0:00 ./
zwave_svc
hubsdk-1  | Process 'zwave_svc' is running.
hubsdk-1  | root         233  0.0  0.0 1198704 5760 ?        Sl   20:51   0:00 ./
iotmi_cdmb
hubsdk-1  | Process 'iotmi_cdmb' is running.
hubsdk-1  | root         268  0.2  0.2 2017424 21968 ?       Sl   20:51   0:00 ./
iotmi_device_agent
hubsdk-1  | Process 'iotmi_device_agent' is running.
hubsdk-1  | root         311  0.1  0.1 1523072 13008 ?       Sl   20:51   0:00 ./
iotmi_lpw_provisioner
hubsdk-1  | Process 'iotmi_lpw_provisioner' is running.
hubsdk-1  | root         132  0.0  0.0 875024  7232 ?        Sl   20:51   0:00 ./
iotmi_log_daemon
hubsdk-1  | Process 'iotmi_log_daemon' is running.
hubsdk-1  | -\-\-\-\-\-\-Successfully Started Hub SDK-\-\-\-
```

成功啟動 Hub SDK 後，請前往 繼續進行裝置加入和管理使用者引導設定以加入和操作裝置。

Note

- 若要存取 Docker 容器 bash shell，請執行下列命令：

```
docker compose exec hubsdk bash
```

- 若要在重新啟動後重新啟動容器，請執行下列命令：

```
docker compose up -d
```

- 若要更新 Hub SDK，請取代下列資料夾中的二進位檔：

```
hub-docker/iotmi
```

- 若要在保留資料時安全地重新啟動容器，請執行下列動作：

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

受管整合示範應用程式

Note

在 Raspberry Pi 上實作 AWS IoT Hub SDK 是一個示範專案，僅用於學習和測試目的，不適用於生產環境。基於本示範的目的，請設定下列組態以利開發：
AWS 登入資料儲存：僅供示範使用，登入資料和憑證會存放在可存取的位置，以便於測試和開發。生產環境必須使用安全儲存解決方案 AWS Secrets Manager，例如 或 Systems Manager 參數存放區。他們必須實作靜態加密，並遵循 AWS IoT 安全準則。
容器權限： 示範會以更高的權限執行，以允許不受限制地存取主機資源並簡化開發工作流程。在生產環境中，容器應以最低的必要權限運作。
網路橋接組態：示範使用網路橋接組態來公開內部網路流量，以便於偵錯和監控。在生產環境中，實作適當的網路隔離和分割，以防止未經授權存取內部網路流量。
USB 裝置許可：啟用不受限制的 USB 裝置存取，以便輕鬆連接開發周邊裝置和測試裝置。針對生產環境，實作嚴格的 USB 裝置控制和驗證，以防止裝置詐騙攻擊。

這些組態可讓您直接進行測試，且不得用於生產環境。部署至生產環境時，請遵循安全最佳實務，以防止主機系統入侵和未經授權存取登入資料。

示範應用程式是以 React 為基礎的示範應用程式，顯示智慧型家庭裝置管理的受管整合功能。此應用程式透過現代 Web 界面示範 Z-Wave 和 Zigbee 裝置的裝置加入、控制和監控。

先決條件

- 註冊 AWS 帳戶。
- 建立登入資料儲存貯體，並將登入資料儲存貯體新增至您的中樞 。
- 完成 Hub 加入設定。
- Node.js 18+ 和 npm。
- AWS CLI 從 受管整合 AWS CLI 命令參考安裝最新版本的 。
- 現代 Web 瀏覽器 (Chrome、Firefox、Safari、Edge)

安裝和設定 應用程式

1. 下載受管整合示範應用程式。
2. 解壓縮套件：

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

3. 安裝依存項目：

```
npm install
```

4. 在根目錄中建立.env檔案：

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

# IoT Managed Integrations Endpoint
```

```
REACT_APP_IOT_ENDPOINT=https://your-iot-endpoint.amazonaws.com

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

5. 建置和啟動應用程式：

```
npm start
```

6. 存取應用程式，網址為：

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

如需定價資訊，請參閱 AWS IoT Device Management 定價頁面上的受管整合一節。

離線受管整合中樞

Hub SDK 離職程序概觀

中樞移出程序會從 AWS 雲端 管理系統中移除中樞。當雲端傳送 DeleteManagedThing 請求時，程序會完成兩個主要目標：

裝置端動作：

- 重設中樞的內部狀態
- 刪除所有本機儲存的資料
- 準備裝置以供未來重新加入

雲端動作：

- 移除與中樞相關聯的所有雲端資源
- 與上一個帳戶完全中斷連線

客戶通常會在下列情況下啟動中樞離職：

- 變更中樞的關聯帳戶
- 使用新裝置取代現有的中樞

此程序可確保中樞組態之間的乾淨、安全轉換，實現無縫的裝置管理和帳戶彈性。

先決條件

- 您必須擁有已加入的中樞。如需說明，請參閱 Hub 加入設定。
- 在位於 /data/aws/iotmi/config/ 的 `iotmi_config.json`檔案中，確認 `iot_provisioning_state`顯示 `PROVISIONED`。
- 確認 中參考的永久憑證和金鑰`iotmi_config.json`存在於其指定的路徑中。
- 確定 HubOnboarding、Agent、Provisioner 和 MQTT 代理已正確設定並執行。
- 確認集線器沒有子裝置。使用 DeleteManagedThing API 移除所有子裝置再繼續。

Hub SDK 離職程序

請依照下列步驟將中樞移出：

擷取 hub_managed_thing ID

`iotmi_config.json` 檔案用於存放受管整合中樞的受管物件 ID。此識別符是重要資訊，可讓中樞與 AWS IoT 受管整合服務通訊。受管物件 ID 會存放在 JSON 檔案的 rw （讀寫） 區段的 `managed_thing_id` 欄位下。這在下列範例組態中可見：

```
{
      "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
```

```
        }
    }
}
```

將命令傳送至離職中樞

使用您的帳戶登入資料，並使用上一節managed_thing_id擷取的 執行 命令：

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

確認中樞已離職

使用您的帳戶登入資料，並使用上一節managed_thing_id擷取的 執行 命令：

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

成功和失敗案例

成功案例

如果移出中樞的命令成功，預期會有下列範例回應：

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

此外，如果中樞離機命令成功，iotmi_config.json則會觀察下列範例。確認 rw 區段僅包含 iot_provisioning_state 和選用的中繼資料。中繼資料不存在是可接受的。iot_provisioning_state 必須是 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
        }
    }
}
```

失敗案例

如果移出中樞的命令失敗，預期會有下列範例回應：

```
{
    "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 為 DELETE_IN_PROGRESS，請遵循 Hub 復原中的指示。
- 如果 ProvisioningStatus 不是 DELETE_IN_PROGRESS，則將中樞移出的命令在受管整合雲端中失敗，或未由受管整合雲端接收。遵循 Hub 復原中的指示。
- 如果離職失敗，您的iotmi_config.json檔案看起來會像下面的範例檔案。

```
{
    "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
            }
        }
}
```

（選用）卸任 Hub SDK 之後

> **Important**
>
> 下列案例列出在卸任 Hub SDK 失敗後要採取的選用動作，或者如果您想要在卸任後重新加入您的中樞。

重新加入

如果離職成功，請依照步驟 3：建立受管物件（機群佈建）和其餘的加入程序來加入 Hub SDK。

Hub 復原

裝置中樞離職成功且雲端離職失敗

如果 GetManagedThing API 呼叫未傳回`Managed Thing resource not found`訊息，但檔案已`iotmi_config.json`離線。請參閱範例 json 檔案的成功案例。

若要從此案例中復原，請參閱強制刪除。

裝置中樞離職失敗

當檔案`iotmi_config.json`未正確離職時，就會發生這種情況。請參閱範例 json 檔案的失敗案例。

若要從此案例中復原，請參閱強制刪除。如果 `iotmi_config.json` 仍未停止運作，則必須重設集線器。

裝置中樞離職和雲端離職失敗

在此案例中， `iotmi_config.json` 仍未停止運作，且中樞狀態為 `ACTIVATED`、 或 `DISCOVERED`。

若要從此案例復原，請參閱強制刪除。如果強制刪除失敗，或iotmi_config.json仍未停止運作，則必須重設集線器。

中樞離線，中樞狀態為 DELETE_IN_PROGRESS

在此案例中，中樞離線，雲端會收到離職命令。

若要從此案例中復原，請參閱強制刪除。

強制刪除

若要刪除沒有成功裝置中樞離職的雲端資源，請遵循下列步驟。此操作可能會導致雲端和裝置狀態不一致，進而可能導致未來操作發生問題。

使用中樞的 managed_thing_id和 強制參數呼叫 DeleteManagedThing API：

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

接著，呼叫 GetManagedThing API 並確認其傳回 Managed Thing resource not found。這會確認雲端資源已刪除。

> **Note**
>
> 不建議使用此方法，因為它可能會導致雲端和裝置狀態之間的不一致。一般而言，在嘗試刪除雲端資源之前，最好確保裝置中樞順利移出。

通訊協定特定的中介軟體

> **Important**
>
> 此處提供的文件和程式碼說明中介軟體的參考實作。它不會作為 SDK 的一部分提供給您。

通訊協定特定的中介軟體具有與基礎通訊協定堆疊互動的關鍵角色。受管整合的裝置加入和裝置控制元件 Hub SDK 都會使用它來與終端裝置互動。

中介軟體會執行下列函數。

- 透過提供一組常見的 APIs，從不同廠商的裝置通訊協定堆疊中抽象APIs。
- 提供執行緒排程器、事件佇列管理和資料快取等軟體執行管理。

中介軟體架構

下面的區塊圖代表 Zigbee 中介軟體的架構。Z-Wave 等其他通訊協定的中介軟體架構也類似。

通訊協定特定的中介軟體有三個主要元件。

- ACS Zigbee DPK：Zigbee Device Porting Kit (DPK) 用於提供基礎硬體和作業系統的抽象，從而實現可攜性。基本上，這可以視為硬體抽象層 (HAL)，它提供常用APIs，以控制來自不同廠商的 Zigbee 無線電並與之通訊。Zigbee 中介軟體包含適用於 Silicon Labs Zigbee 應用程式架構的 DPK API 實作。
- ACS Zigbee 服務：Zigbee 服務做為專用協助程式執行。它包含 API 處理常式，可透過 IPC 通道從用戶端應用程式提供 API 呼叫。AIPC 用作 Zigbee 轉接器和 Zigbee 服務之間的 IPC 通道。它提供其他功能，例如同時處理非同步/同步命令、從 HAL 處理事件，以及使用 ACS Event Manager 進行事件註冊/發佈。
- ACS Zigbee 轉接器：Zigbee 轉接器是應用程式程序內執行的程式庫（在此情況下，應用程式是 CDMB 外掛程式）。Zigbee 轉接器提供一組 APIs，供 CDMB/Provisioner 通訊協定外掛程式等用戶端應用程式使用，以控制終端裝置並與之通訊。

End-to-end中介軟體命令流程範例

以下是通過 Zigbee 中介軟體的命令流程範例。

以下是通過 Z-Wave 中介軟體的命令流程範例。

通訊協定特定的中介軟體程式碼組織

本節包含儲存`IotManagedIntegrationsDeviceSDK-Middleware`庫中每個元件程式碼位置的相關資訊。以下是此儲存庫中資料夾結構的範例。

```
./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
```

主題

- Zigbee 中介軟體程式碼組織
- Z-Wave 中介軟體程式碼組織

Zigbee 中介軟體程式碼組織

以下顯示 Zigbee 參考中介軟體程式碼組織。

主題

- ACS Zigbee DPK
- Silicon Labs Zigbee SDK
- ACS Zigbee 服務
- ACS Zigbee 轉接器

ACS Zigbee DPK

Zigbee DPK 的程式碼位於下列範例中列出的目錄內：

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

```
|—   |— src
|—   |— zigbeed
|—       |— ember
|—       |— include
|— zwave
|—   |— include
|—   |— src
|—   |— zwaved
|—       |— fxnZwaveClient
|—       |— include
|—       |— zware
```

Silicon Labs Zigbee SDK

Silicon Labs SDK 會顯示在 IotManagedIntegrationsDeviceSDK-Middleware/*example*-iot-ace-z3-gateway 資料夾內。此 ACS Zigbee DPK 層已針對此 Silicon Labs 開發套件實作。

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

ACS Zigbee 服務

Zigbee Service 的程式碼位於 IotManagedIntegrationsDeviceSDK-Middleware/*example*-iot-ace-general/middleware/zigbee/ 資料夾內。此位置的 src和 include子資料夾包含與 ACS Zigbee 服務相關的所有檔案。

```
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 轉接器

ACS Zigbee 轉接器的程式碼位於 `IotManagedIntegrationsDeviceSDK-Middleware/example-iot-ace-general/middleware/zigbee/api` 資料夾內。此位置的 `src`和 `include`子資料夾包含與 ACS Zigbee 轉接器程式庫相關的所有檔案。

```
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 中介軟體程式碼組織

以下顯示 Z-wave 參考中介軟體程式碼組織。

主題

- ACS Z-Wave DPK
- Silicon Labs ZWare 和 Zip Gateway
- ACS Z-Wave 服務
- ACS Z-Wave 轉接器

ACS Z-Wave DPK

Z-Wave DPK 的程式碼位於 `IotManagedIntegrationsDeviceSDK-Middleware/example-iot-ace-dpk/example/dpk/ace_hal/zwave` 資料夾內。

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

```
|—   |— include
|—   |— src
|—   |— zigbeed
|—       |— ember
|—       |— include
|— zwave
|—   |— include
|—   |— src
|—   |— zwaved
|—       |— fxnZwaveClient
|—       |— include
|—       |— zware
```

Silicon Labs ZWare 和 Zip Gateway

Silicon 實驗室 ZWare 和 Zip Gateway 的程式碼位於 `IotManagedIntegrationsDeviceSDK-Middleware/example-iot-ace-z3-gateway` 資料夾內。此 ACS Z-Wave DPK 層已針對 Z-Wave C-APIs和 Zip 閘道實作。

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

ACS Z-Wave 服務

Z-Wave Service 的程式碼位於 資料夾內。`IoTmanagedintegrationsMiddlewares/exampleiot-ace-zwave-mw/`此位置的 `src`和 `include` 資料夾包含與 ACS Z-Wave 服務相關的所有檔案。

```
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 轉接器

ACS Zigbee 轉接器的程式碼位於 IotManagedIntegrationsDeviceSDK-Middleware/*example*-iot-ace-zwave-mw/cli/ 資料夾內。此位置的 src和 include 資料夾包含與 ACS Z-Wave 轉接器程式庫相關的所有檔案。

```
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
```

整合中介軟體與 SDK

下列各節會討論新中樞上的中介軟體整合。

主題

裝置移植套件 (DPK) API 整合

若要整合任何晶片組廠商 SDK 與中介軟體，中間的 DPK （裝置移植套件） 層會提供標準 API 介面。受管整合服務供應商或 ODMs 需要根據其 IoT Hub 上使用的 Zigbee/Z-wave/Wi-Fi 晶片組支援的廠商 SDK 實作這些 APIs。

參考實作和程式碼組織

除了中介軟體之外，所有其他 Device SDK 元件，例如受管整合 Device Agent 和通用資料模型橋接器 (CDMB) 都可以使用，無需進行任何修改，而且只需要跨編譯。

中介軟體的實作是以適用於 Zigbee 和 Z-Wave 的 Silicon Labs 開發套件為基礎。如果中介軟體中存在的 Silicon Labs SDK 支援在新中樞中使用的 Z-Wave 和 Zigbee 晶片組，則可以在不進行任何修改的情況下使用參考中介軟體。您只需要交叉編譯中介軟體，然後就可以在新的中樞上執行。

您可以在 中找到 Zigbee 的 DPK （裝置移植套件） APIs`acehal_zigbee.c`，且 DPK APIs的參考實作存在於 `zigbee` 資料夾內。

```
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
```

適用於 Z-Wave APIs DPK API 可在 中找到，acehal_zwave.c且 DPK APIs 的參考實作存在於 zwaved 資料夾內。

```
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
```

做為為不同廠商 SDK 實作 DPK 層的起點，可以使用和修改參考實作。需要下列兩個修改，才能支援不同的廠商 SDK：

1. 將目前的廠商 SDK 取代為儲存庫中的新廠商 SDK。
2. 根據新廠商 SDK 實作中介軟體 DPK （ 裝置移植套件 ） APIs。

受管整合 終端裝置 SDK

建置 IoT 平台，將智慧型裝置連線至受管整合，並透過統一的控制界面處理命令。終端裝置 SDK 會與您的裝置韌體整合，並提供開發套件邊緣元件的簡化設定，以及與 AWS IoT Core 和 AWS IoT 裝置管理的安全連線。從 下載最新版本的結束裝置 SDK AWS 管理主控台

本指南說明如何在韌體中實作結束裝置 SDK。檢閱架構、元件和整合步驟，以開始建置您的實作。

主題

什麼是結束裝置 SDK？

什麼是結束裝置 SDK？

終端裝置 SDK 是 提供的來源碼、程式庫和工具的集合 AWS IoT。開發套件專為資源受限的環境而打造，可支援具有最低 512 KB RAM 和 4 MB 快閃記憶體的裝置，例如在內嵌 Linux 和即時作業系統 (RTOS) 上執行的攝影機和空氣淨化器。從 AWS IoT 管理主控台下載最新版本的結束裝置 SDK。

核心元件

SDK 結合了用於雲端通訊的 MQTT 代理程式、用於任務管理的任務處理常式，以及受管整合 Data Model Handler。這些元件可共同運作，在裝置與受管整合之間提供安全的連線能力和自動化資料轉譯。

如需詳細的技術需求，請參閱 技術參考。

終端裝置 SDK 架構和元件

本節說明結束裝置 SDK 架構及其元件如何與您的低階 C-Functions 互動。下圖說明 SDK 架構中的核心元件及其關係。

終端裝置 SDK 元件

終端裝置 SDK 架構包含這些元件，用於受管整合功能整合：

佈建者

在受管整合雲端中建立裝置資源，包括用於安全 MQTT 通訊的裝置憑證和私有金鑰。這些登入資料會在您的裝置與受管整合之間建立信任的連線。

MQTT 代理程式

透過安全執行緒的 C 用戶端程式庫管理 MQTT 連線。此背景程序會處理多執行緒環境中的命令佇列，以及記憶體受限裝置的可設定佇列大小。訊息會透過受管整合路由進行處理。

任務處理常式

處理裝置韌體、安全修補程式和檔案交付的over-the-air(OTA) 更新。此內建服務會管理所有已註冊裝置的軟體更新。

資料模型處理常式

使用 的事項資料模型實作，在受管整合與低階 C-Functions AWS之間翻譯操作。如需詳細資訊，請參閱 GitHub 上的事項文件。

金鑰和憑證

透過 PKCS #11 API 管理密碼編譯操作，同時支援硬體安全模組和軟體實作，例如 corePKCS11。此 API 會在 TLS 連線期間處理 Provisionee 和 MQTT Agent 等元件的憑證操作。

佈建者

佈建者是受管整合的元件，可依宣告啟用機群佈建。使用佈建者，您可以安全地佈建裝置。開發套件會建立裝置佈建所需的資源，其中包含從受管整合雲端取得的裝置憑證和私有金鑰。當您想要佈建裝置，或有任何變更可能需要您重新佈建裝置時，您可以使用佈建者。

主題

佈建者工作流程

此程序需要在雲端和裝置端進行設定。客戶設定雲端需求，例如自訂端點、佈建設定檔和受管物件。在第一次開啟裝置電源時，佈建者：

1. 使用宣告憑證連線至受管整合端點
2. 透過機群佈建掛鉤驗證裝置參數
3. 在裝置上取得並存放永久憑證和私有金鑰
4. 裝置使用永久憑證重新連線
5. 探索裝置功能並將其上傳至受管整合

成功佈建後，裝置會直接與受管整合通訊。佈建者只會針對重新佈建任務啟用 。

設定環境變數

在雲端環境中設定下列 AWS 登入資料：

```
$ 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
```

註冊自訂端點

在雲端環境中使用 RegisterCustomEndpoint API 命令，為device-to-cloud通訊建立自訂端點。

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

回應範例

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

> **Note**
>
> 存放端點地址以設定佈建參數。 使用 GetCustomEndpoint API 傳回端點資訊。
> 如需詳細資訊，請參閱《 受管整合 API 參考指南》中的 GetCustomEndpoint API 和 RegisterCustomEndpoint API。

建立佈建設定檔

建立佈建設定檔，以定義您的機群佈建方法。在雲端環境中執行 CreateProvisioningProfile API，以傳回宣告憑證和私有金鑰進行裝置身分驗證：

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

回應範例

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

您可以實作 corePKCS11 平台抽象程式庫 (PAL)，讓 corePKCS11 程式庫可與您的裝置搭配使用。corePKCS11 PAL 連接埠必須提供存放宣告憑證和私有金鑰的位置。使用此功能，您可以安全地存放裝置的私有金鑰和憑證。您可以將私有金鑰和憑證存放在硬體安全模組 (HSM) 或信任的平台模組 (TPM) 上。

建立受管物件

使用 CreateManagedThing API 向受管整合雲端註冊您的裝置。包含裝置的序號 (SN) 和通用產品代碼 (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;"
```

以下顯示範例 API 回應。

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

API 會傳回可用於佈建驗證的受管物件 ID。您需要提供裝置序號 (SN) 和通用產品代碼 (UPC)，這些代碼在佈建交易期間與核准的受管物件相符。交易會傳回類似下列的結果：

```
/**
 * @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 使用者 Wi-Fi 佈建

裝置製造商和解決方案供應商擁有自己的專屬 Wi-Fi 佈建服務，用於接收和設定 Wi-Fi 登入資料。Wi-Fi 佈建服務涉及使用專用行動應用程式、低功耗藍牙 (BLE) 連線和其他專屬通訊協定，以安全地傳輸初始設定程序的 Wi-Fi 登入資料。

終端裝置 SDK 的取用者必須實作 Wi-Fi 佈建服務，而且裝置可以連線至 Wi-Fi 網路。

依宣告佈建機群

使用佈建者，最終使用者可以使用依宣告佈建來佈建唯一憑證，並使用受管整合進行註冊。

可從佈建範本回應或裝置憑證取得用戶端 ID <common name>“_”<serial number>

受管物件功能

佈建者探索受管物件功能，然後將功能上傳至受管整合。它可讓應用程式和其他 服務存取功能。裝置、其他 Web 用戶端和服務可以使用 MQTT 和預留 MQTT 主題來更新功能，或使用 REST API 來更新 HTTP。

Over-the-Air更新

OTA 架構概觀

Over-the-Air(OTA) 更新程序涉及數個元件一起運作，以將韌體更新交付至您的裝置。下圖說明如何透過結束裝置 SDK、中樞 SDK 和 功能之間的互動來處理 OTA 更新請求。

OTA 更新架構包含下列元件：

- 客戶：將任務文件上傳到 S3 儲存貯體，並透過 API 啟動更新
- OTA 服務：處理任務建立、驗證和管理
- AWS IoT 任務：管理任務執行和交付至裝置
- 裝置：使用 Harmony SDK 接收和套用更新

先決條件

在建立 OTA 任務之前，您必須設定下列先決條件：

設定 Amazon S3 存取

若要啟用 OTA 更新，您必須將任務文件上傳至 Amazon S3 儲存貯體，並設定適當的存取許可：

1. 將 OTA 任務文件上傳至 S3 儲存貯體
2. 新增 Amazon S3 儲存貯體政策，授予受管整合對任務文件的存取權：

JSON

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

```
    {
      "Sid": "PolicyForS3JobDocument",
      "Effect": "Allow",
      "Principal": {
        "Service": "iotmanagedintegrations.amazonaws.com"
      },
      "Action": "s3:GetObject",
      "Resource": [
        "arn:aws:s3:::YOUR_BUCKET/*",
        "arn:aws:s3:::YOUR_BUCKET/ota_job_document.json",
        "arn:aws:s3:::YOUR_BUCKET"
      ]
    }
  ]
}
```

實作Over-the-Air(OTA) 任務

視您的更新需求和裝置目標策略而定，您可以透過兩種方式建立 OTA 任務：

一次性 OTA 任務更新

一次性 OTA 任務包含執行 OTA 更新的目標靜態清單 (ManagedThings)。您一次最多可以新增 100 個目標。工作流程使用 AWS IoT 任務與機群索引，同時維護受管整合抽象層。

使用下列範例來建立一次性 OTA 任務：

```
aws iotmanagedintegrations create-ota-task \
  --description "One-time OTA update" \
  --s3-url "s3://test-job-document-bucket/ota-job-document.json" \
  --protocol HTTP \
  --target ["arn:aws:iotmanagedintegrations:region:account id:managed-thing/managed
 thing id"] \
  --ota-mechanism PUSH \
  --ota-type ONE_TIME \
  --client-token "foo" \
  --tags '{"key1":"foo","key2":"foo"}'
```

持續 OTA 任務更新

OTA (Over-the-Air) 分組工作流程可讓您根據特定屬性，使用具有機群索引 AWS IoT 的工作，同時維護受管整合抽象層，將韌體更新部署至裝置群組。連續 OTA 任務使用查詢字串，而非特定目標。符合

查詢條件的所有裝置都會進行 OTA 更新，並且會持續重新評估查詢條件。相符的目標將具有任務部署。

設定先決條件

在建立持續 OTA 任務之前，請先完成下列先決條件：

1. 呼叫 CreateManagedThing API 並執行機群佈建來建立受管物件。
2. 將中繼資料屬性新增至受管物件，以進行查詢目標鎖定。

ManagedThing 使用 UpdateManagedThing API 將屬性和中繼資料新增至 ：

```
aws iotmanagedintegrations update-managed-thing \
  --managed-thing-id "YOUR_MANAGED_THING_ID" \
  --meta-data '{"owner":"managedintegrations","version":"1.0"}'
```

使用下列範例來建立連續 OTA 任務：

```
aws iotmanagedintegrations create-ota-task \
  --description "Continuous OTA update" \
  --s3-url "s3://test-job-document-bucket/ota-job-document.json" \
  --protocol HTTP \
  --ota-mechanism PUSH \
  --ota-type CONTINUOUS \
  --client-token "foo" \
  --ota-target-query-string "attributes.owner=managedintegrations" \
  --tags '{"key1":"foo","key2":"foo"}'
```

了解持續 OTA 工作流程

持續 OTA 更新工作流程遵循下列步驟：

1. 您可以使用 UpdateManagedThing API 更新具有屬性的受管物件。
2. 使用以特定裝置屬性為目標的查詢字串建立 OTA 任務。
3. OTA 服務 AWS IoT Core 會根據查詢屬性在 中建立動態物件群組
4. IoT 任務會在相符的裝置上執行更新
5. 您可以透過 ListOtaTaskExecutions API 監控進度，或透過 Kinesis 串流監控 OTA 通知 （如果啟用）。

受管整合 OTA 和 IoT 任務之間的差異

受管整合 OTA 和 IoT 任務之間的基本區別在於服務協調和自動化。受管整合 OTA 提供單一服務解決方案，可消除多服務協調的複雜性。

受管整合 OTA 會自動執行的操作：

- 動態物件群組建立：根據您的查詢條件自動產生 AWS IoT Core 物件群組。
- 目標解析：將查詢字串（範例：`attributes.owner=managedintegrations`) 轉換為實際的裝置目標。
- 服務整合：在 AWS IoT Core IoT Jobs 和 Fleet Indexing 服務之間無縫協調。
- 生命週期管理：處理從建立到執行監控的整個 OTA 工作流程。

MI OTA 消除了什麼：

- 在 中建立物件群組 AWS IoT Core。
- 將物件新增至群組。
- 建立 IoT 任務。

受管整合 OTA 會根據您的查詢字串在內部處理這三個操作、自動探索符合您條件的裝置、在幕後建立 IoT 任務，以及協調完整的 OTA 工作流程，而不需要您直接與多個 AWS 服務互動。

OTA 任務組態設定

您可以建立 OTA 更新的組態，以控制如何將更新推展至裝置、設定中止條件，以及設定逾時。

範例：CreateOtaTaskConfiguration

使用下列範例來建立 OTA 任務組態：

```
aws iotmanagedintegrations create-ota-task-configuration \
  --description "OTA configuration" \
  --name "MyOtaConfig" \
  --push-config '{
    "AbortConfig": {
      "AbortConfigCriteriaList": [
        {
          "Action": "CANCEL",
          "FailureType": "FAILED",
```

```
          "MinNumberOfExecutedThings": 1,
          "ThresholdPercentage": 90.0
        }
      ]
    },
    "RolloutConfig": {
      "ExponentialRolloutRate": {
        "BaseRatePerMinute": 1,
        "IncrementFactor": 3.0,
        "RateIncreaseCriteria": {
          "numberOfNotifiedThings": 1
        }
      },
      "MaximumPerMinute": 1
    },
    "TimeoutConfig": {
      "InProgressTimeoutInMinutes": 100
    }
  }' \
  --client-token "foo"
```

將組態設定套用至 OTA 任務

建立組態後，您會收到新增至CreateOtaTask請求taskConfigurationId的 以及其他組態：

```
aws iotmanagedintegrations create-ota-task \
  --description "OTA with configuration" \
  --s3-url "s3://test-job-document-bucket/ota-job-document.json" \
  --protocol HTTP \
  --target ["arn:aws:iotmanagedintegrations:region:account id:managed-thing/managed
 thing id"] \
  --ota-mechanism PUSH \
  --ota-type ONE_TIME \
  --client-token "foo" \
  --task-configuration-id "ae4f49352c5443369f43ad6c3a7f1580" \
  --ota-scheduling-config '{
    "EndBehavior": "STOP_ROLLOUT",
    "EndTime": "2024-10-23T17:00",
    "StartTime": "2024-10-20T17:00"
  }' \
  --ota-task-execution-retry-config '{
    "RetryConfigCriteria": [
      {
```

```
        "FailureType": "FAILED",
        "MinNumberOfRetries": 1
      }
    ]
  }' \
  --tags '{"key1":"foo","key2":"foo"}'
```

監控 OTA 通知

您可以使用兩種不同的方法來監控 OTA 更新：

透過 Kinesis Data Streams 傳送通知

啟用 OTA 通知時，更新狀態事件會自動推送到您的 Kinesis 串流。這可提供跨裝置韌體更新進度的即時可見性。

使用 ListOtaTaskExecutions API 監控

您可以使用 ListOtaTaskExecutions API 來手動檢查受管物件的 OTA 更新狀態：

```
aws iotmanagedintegrations list-ota-task-executions \
  --task-id "task-123456789" \
  --max-results 25
```

回應提供每個受管物件的詳細執行狀態：

```
{
  "taskExecutionSummaries": [
    {
      "taskExecutionSummary": {
        "executionNumber": 1,
        "lastUpdatedAt": 1634567890,
        "queuedAt": 1634567800,
        "startedAt": 1634567830,
        "status": "SUCCEEDED",
        "retryAttempt": 0
      },
      "managedThingId": "device-001"
    },
    {
      "taskExecutionSummary": {
        "executionNumber": 1,
        "lastUpdatedAt": 1634567920,
```

```
        "queuedAt": 1634567800,
        "startedAt": 1634567840,
        "status": "IN_PROGRESS",
        "retryAttempt": 0
      },
      "managedThingId": "device-002"
    }
  ],
  "nextToken": "NEXT_TOKEN"
}
```

此 API 可讓您擷取特定 OTA 任務針對的每個受管物件的詳細執行狀態，包括時間戳記和目前狀態。

處理任務文件

當您建立 OTA 任務時，任務處理常式會在您的裝置上執行下列步驟。當更新可用時，它會透過 MQTT 請求任務文件。

1. 訂閱 MQTT 通知主題。
2. 針對待定任務呼叫 StartNextPendingJobExecution API。
3. 接收可用的任務文件。
4. 根據指定的逾時處理更新。

使用任務處理常式，應用程式可以判斷要立即採取動作，還是等到指定的逾時期間。

實作 OTA 代理程式

當您從受管整合收到任務文件時，您必須實作自己的 OTA 代理程式，以處理任務文件、下載更新並執行任何安裝操作。OTA 代理程式需要執行下列步驟：

1. 韌體 Amazon S3 URLs剖析任務文件。
2. 透過 HTTP 下載韌體更新。
3. 驗證數位簽章。
4. 安裝已驗證的更新。
5. `iotmi\_JobsHandler\_updateJobStatus` 使用 `SUCCESS`或 `FAILED` 狀態呼叫 。

當您的裝置成功完成 OTA 操作時，必須呼叫狀態為 的 `iotmi\_JobsHandler \_updateJobStatus` API，`JobSucceeded`才能報告成功的任務。

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

資料模型程式碼產生器

了解如何將程式碼產生器用於資料模型。產生的程式碼可用來序列化和還原序列化雲端和裝置之間交換的資料模型。

專案儲存庫包含用於建立 C 程式碼資料模型處理常式的程式碼產生工具。下列主題說明程式碼產生器和工作流程。

主題

- 程式碼產生程序
- 環境設定
- 產生裝置的程式碼

程式碼產生程序

程式碼產生器會從三個主要輸入建立 C 來源檔案：Zigbee Cluster Library (ZCL) 進階平台的事項資料模型 (.matter 檔案） AWS實作、處理預先處理的 Python 外掛程式，以及定義程式碼結構的 Jinja2 範本。在產生期間，Python 外掛程式會透過新增全域類型定義、根據其相依性組織資料類型，以及格式化範本轉譯的資訊，來處理您的 .matter 檔案。

下圖說明建立 C 來源檔案的程式碼產生器。

終端裝置 SDK 包含可在connectedhomeip專案codegen.py中使用 的 Python 外掛程式和 Jinja2 範本。此組合會根據您的 .matter 檔案輸入，為每個叢集產生多個 C 檔案。

下列子主題說明這些檔案。

- Python 外掛程式
- Jinja2 範本
- （選用） 自訂結構描述

Python 外掛程式

程式碼產生器 會`codegen.py`剖析 .matter 檔案，並將資訊做為 Python 物件傳送至外掛程式。外掛程式檔案會`iotmi_data_model.py`預先處理此資料，並使用提供的範本轉譯來源。預先處理包括：

1. 新增無法從 取得的資訊`codegen.py`，例如全域類型
2. 對資料類型執行拓撲排序，以建立正確的定義順序

> **Note**
>
> 拓撲排序可確保相依類型在其相依性之後定義，無論其原始順序為何。

Jinja2 範本

終端裝置 SDK 提供專為資料模型處理常式和低階 C-Functions 量身打造的 Jinja2 範本。

Jinja2 範本

範本	產生的來源	備註
cluster.h.jinja	iotmi_device_<cluster>.h	建立低階 C 函數標頭檔案。
cluster.c.jinja	iotmi_device_<cluster>.c	使用資料模型處理常式實作和註冊回呼函數指標。
cluster_type_helpers.h.jinja	iotmi_device_type_helpers_<cluster>.h	定義資料類型的函數原型。
cluster_type_helpers.c.jinja	iotmi_device_type_helpers_<cluster>.c	產生叢集特定列舉、點陣圖、清單和結構的資料類型函數原型。
iot_device_dm_types.h.jinja	iotmi_device_dm_types.h	定義全域資料類型的 C 資料類型。
iot_device_type_helpers_global.h.jinja	iotmi_device_type_helpers_global.h	定義全域操作的 C 資料類型。
iot_device_type_helpers_global.c.jinja	iotmi_device_type_helpers_global.c	宣告標準資料類型，包括布林值、整數、浮點數、字串、點陣圖、清單和結構。

（選用）自訂結構描述

終端裝置 SDK 結合了標準化程式碼產生程序與自訂結構描述。這可讓您的裝置和裝置軟體延伸事項資料模型。自訂結構描述可協助描述裝置device-to-cloud功能。

如需受管整合資料模型的詳細資訊，包括格式、結構和需求，請參閱 受管整合資料模型。

使用 codegen.py工具產生自訂結構描述的 C 來源檔案，如下所示：

> **Note**
>
> 每個自訂叢集需要下列三個檔案的相同叢集 ID。

- 建立JSON格式為 的自訂結構描述，提供叢集的表示，以供功能報告在雲端中建立新的自訂叢集。範例檔案位於 `codegen/custom_schemas/custom.SimpleLighting@1.0`。
- 以包含與自訂結構描述相同資訊的XML格式建立 ZCL (Zigbee 叢集程式庫） 定義檔案。使用 ZAP 工具從 ZCL XML 產生您的事項 IDL 檔案。範例檔案位於 `codegen/zcl/custom.SimpleLighting.xml`。
- ZAP 工具的輸出為 ，`Matter IDL File (.matter)`並定義與您的自訂結構描述對應的事項叢集。這是`codegen.py`工具為終端裝置 SDK 產生 C 來源檔案的輸入。範例檔案位於 `codegen/matter_files/custom-light.matter`。

如需如何將自訂受管整合資料模型整合至程式碼產生工作流程的詳細說明，請參閱 產生裝置的程式碼。

環境設定

了解如何設定您的環境以使用`codegen.py`程式碼產生器。

主題

- 先決條件
- 設定您的環境

先決條件

在設定環境之前，請安裝下列項目：

- Git
- Python 3.10 或更新版本
- Poetry 1.2.0 或更新版本

設定您的環境

使用下列程序將環境設定為使用 codegen.py 程式碼產生器。

1. 從 下載最新版本的結束裝置 SDK AWS 管理主控台。
2. 設定 Python 環境。Codegen 專案以 python 為基礎，並使用 Poetry 進行相依性管理。
   - 在 codegen目錄中使用 poetry 安裝專案相依性：

     ```
     poetry run poetry install --no-root
     ```
3. 設定您的儲存庫。
   a. 複製儲存connectedhomeip庫。它使用位於 connectedhomeip/scripts/ 資料夾中的codegen.py指令碼來產生程式碼。如需詳細資訊，請參閱 GitHub 上的連線homeip。GitHub

      ```
      git clone -b v1.4.0.0 https://github.com/project-chip/connectedhomeip.git
      ```
   b. 在與IoT-managed-integrations-End-Device-SDK 根資料夾相同的層級複製它。您的資料夾結構應符合下列項目：

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

> **Note**
>
> 您不需要遞迴複製子模組。

產生裝置的程式碼

使用 受管整合程式碼產生工具，為您的裝置建立自訂 C 程式碼。本節說明如何從 SDK 隨附的範例檔案或您自己的規格產生程式碼。了解如何使用產生指令碼、了解工作流程程序，以及建立符合您裝置需求的程式碼。

主題

- 先決條件
- 產生自訂 .matter 檔案的程式碼
- 程式碼產生工作流程

先決條件

1. Python 3.10 或更新版本。

2. 從產生程式碼的 .matter 檔案開始。終端裝置 SDK 在 中提供兩個範例檔案codgen/matter_files folder：

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

> **Note**
>
> 這些範例檔案會產生示範應用程式叢集的程式碼。

產生程式碼

執行此命令以在輸出資料夾中產生程式碼：

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

產生自訂 .matter 檔案的程式碼

若要產生特定.matter檔案的程式碼或提供您自己的.matter檔案，請執行下列任務。

產生自訂 .matter 檔案的程式碼

1. 準備您的 .matter 檔案
2. 執行產生命令：

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

（選用）使用自訂結構描述產生程式碼

1. 以 JSON 格式準備您的自訂結構描述
2. 執行產生命令：

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

上述命令使用數個元件將您的.matter檔案轉換為C程式碼：

- codegen.py 從 ConnectedHomeIP 專案
- Python 外掛程式位於 codegen/py_scripts/iotmi_data_model.py
- 資料夾中的 Jinja2 範本 codegen/py_scripts/templates

外掛程式會定義要傳遞至 Jinja2 範本的變數，然後用來產生最終 C 程式碼輸出。新增 --format旗標會將 Clang 格式套用至產生的程式碼。

程式碼產生工作流程

程式碼產生程序會使用公用程式函數和透過 的拓撲排序來組織 .matter 檔案資料結構topsort.py。這可確保資料類型及其相依性的正確排序。

然後，指令碼將 .matter 檔案規格與 Python 外掛程式處理結合，以擷取和格式化必要的資訊。最後，它會套用 Jinja2 範本格式來建立最終 C 程式碼輸出。

此工作流程可確保來自 .matter 檔案的裝置特定需求準確轉換為與受管整合系統整合的功能性 C 程式碼。

低階 C-Function APIs

使用提供的低階 C-Function APIs，將您的裝置特定程式碼與受管整合整合整合。本節說明 AWS 資料模型中每個叢集可用的 API 操作，以便有效率的裝置與雲端互動。了解如何實作回呼函數、發出事件、通知屬性變更，以及為裝置端點註冊叢集。

關鍵 API 元件包括：

1. 屬性和命令的回呼函數指標結構
2. 事件發射函數
3. 屬性變更通知函數
4. 叢集註冊函數

透過實作這些 APIs，您可以在裝置的實體操作與 受管整合雲端功能之間建立橋接，確保無縫的通訊和控制。

下一節說明OnOff叢集 API。

OnOff 叢集 API

OnOff.xml 叢集支援這些屬性和命令：。

- 屬性：
  - `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`

  對於每個命令，我們提供 1：1 映射的函數指標，您可以用來掛鉤實作。

屬性和命令的所有回呼都定義在以叢集命名的 C 結構中。

範例 C 結構

```
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 結構之外，也會為所有屬性定義屬性變更報告函數。

```
/* 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);
```

事件報告函數是為所有叢集特定的事件定義。由於OnOff叢集未定義任何事件，以下是來自CameraAvStreamManagement叢集的範例。

```
/* 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);
```

每個叢集也都有一個註冊函數。

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

傳遞至註冊函數的使用者指標將傳遞至回呼函數。

受管整合中的功能和裝置互動

本節說明 C-Function 實作的角色，以及裝置與受管整合裝置功能之間的互動。

主題

- 處理遠端命令
- 處理未經要求的事件

處理遠端命令

遠端命令是由結束裝置 SDK 與 功能之間的互動所處理。下列動作描述了如何使用此互動開啟燈泡的範例。

MQTT 用戶端接收承載並傳遞至 Data Model Handler

當您傳送遠端命令時，MQTT 用戶端會收到 JSON 格式的受管整合訊息。然後，它會將承載傳遞給資料模型處理常式。例如，假設您想要使用受管整合來開啟燈泡。燈泡具有支援OnOff叢集的端點 #1。在此情況下，當您傳送命令以開啟燈泡時，受管整合會透過 MQTT 將請求傳送至裝置，這表示它想要在端點 #1 上叫用開啟命令。

資料模型處理常式會檢查回呼函數並叫用它們

資料模型處理常式會剖析 JSON 請求。如果請求包含屬性或動作，Data Model Handler 會尋找端點，並依序叫用對應的回呼函數。例如，在燈泡的情況下，當 Data Model Handler 收到 MQTT 訊息時，它會檢查與OnOff叢集中定義之 On 命令對應的回呼函數是否已在 endpoint#1 上註冊。

處理常式和 C-Function 實作執行 命令

Data Model Handler 會呼叫找到並叫用的適當回呼函數。C-Function 實作接著會呼叫對應的硬體函數來控制實體硬體，並傳回執行結果。例如，在燈泡的情況下，Data Model Handler 會呼叫回呼函數並存放執行結果。回呼函數接著會開啟燈泡。

Data Model Handler 傳回執行結果

呼叫所有回呼函數後，資料模型處理常式會合併所有結果。然後，它會以 JSON 格式封裝回應，並使用 MQTT 用戶端將結果發佈至受管整合雲端。在燈泡的情況下，回應中的 MQTT 訊息將包含回呼函數開啟燈泡的結果。

處理未經要求的事件

未經請求的事件也會由結束裝置 SDK 與 功能之間的互動處理。下列動作說明如何進行。

裝置傳送通知至資料模型處理常式

發生屬性變更或事件時，例如在裝置上推送實體按鈕時，C-Function 實作會產生未經要求的事件通知，並呼叫對應的通知函數，將通知傳送至 Data Model Handler。

資料模型處理常式翻譯通知

資料模型處理常式會處理收到的通知，並將其轉譯為 AWS 資料模型。

資料模型處理常式會將通知發佈至雲端

資料模型處理常式接著會使用 MQTT 用戶端，將未經要求的事件發佈至受管整合雲端。

開始使用結束裝置 SDK

請依照下列步驟，在 Linux 裝置上執行結束裝置 SDK。本節會引導您完成環境設定、網路組態、硬體函數實作和端點組態。

> **Important**
>
> `examples` 目錄中的示範應用程式及其 中的平台抽象層 (PAL) `platform/posix` 實作僅供參考。請勿在生產環境中使用這些項目。

請仔細檢閱下列程序的每個步驟，以確保適當的裝置與受管整合整合。

整合結束裝置 SDK

1. 設定 Amazon EC2 執行個體

   登入 AWS 管理主控台 並使用 Amazon Linux AMI 啟動 Amazon EC2 執行個體。請參閱《Amazon Elastic Container Registry 使用者指南》中的 Amazon EC2 入門。 https://docs.aws.amazon.com/AmazonECR/latest/userguide/

2. 設定建置環境

   在 Amazon Linux 2023/x86_64 上建置程式碼做為您的開發主機。安裝必要的建置相依性：

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

3. （選用）設定網路

   終端裝置 SDK 最適合與實體硬體搭配使用。如果使用 Amazon EC2，請勿遵循此步驟。

   如果您在使用範例應用程式之前未使用 Amazon EC2，請初始化網路並將您的裝置連線至可用的 Wi-Fi 網路。在裝置佈建之前完成網路設定：

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

4. 設定佈建參數

   > **Note**
   >
   > 遵循佈建者取得宣告憑證和私有金鑰，然後再繼續。

   `example/project_name/device_config.sh` 使用下列佈建參數修改組態檔案：

   佈建參數

巨集參數	Description	如何取得此資訊
`IOTMI_ROOT_CA_PATH`	根 CA 憑證檔案。	您可以從AWS IoT Core 開發人員指南中的下載 Amazon 根 CA 憑證區段下載此檔案。

巨集參數	Description	如何取得此資訊
IOTMI_CLAIM_CERTIFICATE_PATH	宣告憑證檔案的路徑。	若要取得宣告憑證和私有金鑰，請使用 CreateProvisioningProfile API 建立佈建設定檔。如需說明，請參閱建立佈建設定檔。
IOTMI_CLAIM_PRIVATE_KEY_PATH	宣告私有金鑰檔案的路徑。	
IOTMI_MANAGEDINTEGRATIONS_ENDPOINT	受管整合的端點 URL。	若要取得受管整合端點，請使用 RegisterCustomEndpoint API。如需說明，請參閱註冊自訂端點。
IOTMI_MANAGEDINTEGRATIONS_ENDPOINT_PORT	受管整合端點的連接埠號碼	根據預設，連接埠 8883 用於 MQTT 發佈和訂閱操作。連接埠 443 設定為裝置使用的 Application Layer Protocol Negotiation (ALPN) TLS 延伸。

5. 建置並執行示範應用程式

本節示範兩個 Linux 示範應用程式：簡單的安全攝影機和空氣淨化器，兩者都使用 CMake 做為建置系統。

a. 簡單安全攝影機應用程式

若要建置和執行應用程式，請執行下列命令：

```
>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
```

此示範為具有 RTC 工作階段控制器和錄製叢集的模擬攝影機實作低階 C-Functions。在執行佈建者工作流程之前完成 中提到的流程。

示範應用程式的範例輸出：

```
[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. 簡單空氣淨化器應用程式

若要建置和執行應用程式，請執行下列命令：

```
>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
```

此示範為具有 2 個端點和下列支援叢集的模擬空氣淨化器實作低階 C-Functions：

空氣淨化器端點支援的叢集

Endpoint	叢集
端點 #1：空氣淨化器	OnOff
	風扇控制
	HEPA 篩選條件監控
	啟用的碳過濾器監控
端點 #2：空氣品質感應器	空氣品質
	二氧化碳濃度測量
	Formaldehyde 集中度測量
	Pm25 集中度測量
	Pm1 集中度測量
	揮發性有機化合物總集中度測量

輸出類似於攝影機示範應用程式，具有不同的支援叢集。

6. 後續步驟：

受管整合 終端裝置 SDK 和示範應用程式現在正在您的 Amazon EC2 執行個體上執行。這可讓您在自己的實體硬體上開發和測試應用程式。透過此設定，您可以利用 受管整合服務來控制 AWS IoT 您的裝置。

a. 開發硬體回呼函數

實作硬體回呼函數之前，請先了解 API 的運作方式。此範例使用 On/Off OnOff 叢集和 屬性來控制裝置函數。如需 API 詳細資訊，請參閱 低階 C-Function APIs。

```
struct DeviceState
{
```

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

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

b. 設定端點和掛接硬體回呼函數

實作函數之後，請建立端點並註冊您的回呼。完成下列任務：

i. 建立裝置代理程式。

A. 在叫用任何其他 SDK 函數`iotmiDev_Agent_new()`之前，使用 建立裝置代理程式。

B. 您的組態至少必須包含 thingId 和 clientId 參數。

C. 使用 `iotmiDev_Agent_initDefaultConfig()`函數為佇列大小和最大端點等參數設定合理的預設值。

D. 使用完資源後，請使用 `iotmiDev_Agent_free()`函數釋放資源。這可防止記憶體洩漏，並確保應用程式中有適當的資源管理。

ii. 為您要支援的每個叢集結構填入回呼函數指標。

iii. 設定端點並註冊支援的叢集。

使用 建立端點`iotmiDev_Agent_addEndpoint()`，這需要：

A. 唯一的端點 ID。

B. 描述性端點名稱

C. 符合 AWS 資料模型定義的一或多個裝置類型。

D. 建立端點之後，請使用適當的叢集特定註冊函數來註冊叢集。

E. 每個叢集註冊都需要屬性和命令的回呼函數。系統會將使用者內容指標傳遞至回呼，以在通話之間維持狀態。

```
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. 使用任務處理常式來取得任務文件

i. 啟動對 OTA 應用程式的呼叫：

```
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. 呼叫 iotmi_JobsHandler_start 以初始化任務處理常式。

iii. 呼叫 從受管整合iotmi_JobsHandler_getJobDocument擷取任務文件。

iv. 成功取得任務文件時，請在 processOTA函數中撰寫您的自訂 OTA 操作並傳回 JobSucceeded 狀態。

```
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." ) );
```

```
}
```

將結束裝置 SDK 移植到您的裝置

將結束裝置 SDK 移植到您的裝置平台。請依照下列步驟，將您的裝置連線至 AWS IoT Device Management。

下載並驗證結束裝置 SDK

1. 從受管整合主控台下載最新版本的終端裝置 SDK。
2. 確認您的平台位於 中支援的平台清單中參考：支援的平台。

> **Note**
>
> 終端裝置 SDK 已在指定的平台上進行測試。其他平台可能可以運作，但尚未經過測試。

3. 將 SDK 檔案解壓縮 （ 解壓縮 ） 到您的工作區。
4. 使用下列設定來設定您的建置環境：
   - 來源檔案路徑
   - 標頭檔案目錄
   - 必要程式庫
   - 編譯器和連結器旗標
5. 在您移植平台抽象層 (PAL) 之前，請確定平台的基本功能已初始化。功能包括：
   - 作業系統任務
   - 週邊設備
   - 網路介面
   - 平台特定需求

將 PAL 移植到您的裝置

1. 在現有平台目錄中為您的平台特定實作建立新的目錄。例如，如果您使用 FreeRTOS，請在 建立目錄`platform/freertos`。

Example SDK 目錄結構

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

2. 將 POSIX 參考實作檔案 (.c 和 .h) 從 posix 資料夾複製到新的平台目錄。這些檔案提供您需要實作的函數範本。

   - 憑證儲存的快閃記憶體管理
   - PKCS#11 實作
   - 網路傳輸界面
   - 時間同步
   - 系統重新啟動和重設函數
   - 記錄機制
   - 裝置特定的組態

3. 使用 MBedTLS 設定 Transport Layer Security (TLS) 身分驗證。

   - 如果您已經有與平台上 SDK 版本相符的 MBedTLS 版本，請使用提供的 POSIX 實作。
   - 使用不同的 TLS 版本，您可以使用 TCP/IP 堆疊實作 TLS 堆疊的傳輸掛鉤。

4. 將平台的 MbedTLS 組態與 中的 SDK `platform/posix/mbedtls/mbedtls_config.h` 需求進行比較。確定已啟用所有必要選項。

5. SDK 倚賴 coreMQTT 與雲端互動。因此，您必須實作使用下列結構的網路傳輸層：

```
typedef struct TransportInterface
{
```

```
    TransportRecv_t recv;
    TransportSend_t send;
    NetworkContext_t * pNetworkContext;
} TransportInterface_t;
```

如需詳細資訊，請參閱 FreeRTOS 網站上的 Transport Interface 文件。

6. （選用）開發套件使用 PCKS#11 API 來處理憑證操作。 corePKCS 是非硬體特定的 PKCS#11 實作，用於原型設計。我們建議您在生產環境中使用安全加密處理器，例如信任平台模組 (TPM)、硬體安全模組 (HSM) 或安全元素：

   - 檢閱在 使用 Linux 檔案系統進行登入資料管理的範例 PKCS#11 實作`platform/posix/corePKCS11-mbedtls`。
   - 在 實作 PKCS#11 PAL layer`commonDependencies/core_pkcs11/corePKCS11/source/include/core_pkcs11.h`。
   - 在 實作 Linux 檔案系統`platform/posix/corePKCS11-mbedtls/source/iotmi_pal_Pkcs11Operations.c`。
   - 在 實作儲存類型的儲存和載入函數`platform/include/iotmi_pal_Nvm.h`。
   - 在 實作標準檔案存取`platform/posix/source/iotmi_pal_Nvm.c`。

   如需詳細的移植說明，請參閱 FreeRTOS 使用者指南中的移植 corePKCS11 程式庫。

7. 將 SDK 靜態程式庫新增至您的建置環境：

   - 設定程式庫路徑以解決任何連結器問題或符號衝突
   - 確認所有相依性都已正確連結

測試您的連接埠

您可以使用現有的範例應用程式來測試您的連接埠。編譯必須在沒有任何錯誤或警告的情況下完成。

> **Note**
>
> 我們建議您從最簡單的多工作業應用程式開始。範例應用程式提供多工作業對等項目。

1. 在 中尋找範例應用程式`examples/[device_type_sample]`。
2. 將 `main.c` 檔案轉換為您的專案，並新增 項目來呼叫現有的 main() 函數。

3. 確認您可以成功編譯示範應用程式。

技術參考

主題

參考：支援的平台

下表顯示 SDK 支援的平台。

支援平台

平台	Architecture	作業系統
Linux x86_64	x86_64	Linux
安貝拉文	Armv8 (AArch64)	Linux
AmebaD	Armv8-M 32 位元	FreeRTOS
ESP32S3	Xtensa LX7 32 位元	FreeRTOS

參考：技術需求

下表顯示 SDK 的技術需求，包括 RAM 空間。使用相同組態時，終端裝置 SDK 本身需要約 5 到 10 MB 的 ROM 空間。

RAM 空間

SDK 和元件	空間需求（使用的位元組數）
結束裝置 SDK 本身	180 KB
預設 MQTT Agent 命令佇列	480 位元組（可設定）
預設 MQTT Agent 傳入佇列	320 位元組（可設定）

參考：通用 API

本節是不屬於叢集的 API 操作清單。

```
/* 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);
```

的受管整合中的安全性 AWS IoT Device Management

的雲端安全 AWS 是最高優先順序。身為 AWS 客戶，您可以受益於資料中心和網路架構，這些架構專為符合最安全敏感組織的需求而建置。

安全性是 AWS 與您之間的共同責任。共同責任模式將其描述為雲端的安全性和雲端中的安全性：

- 雲端的安全性 – AWS 負責保護在 中執行 AWS 服務的基礎設施 AWS 雲端。 AWS 也為您提供可安全使用的服務。作為AWS 合規計畫的一部分，第三方稽核人員會定期測試和驗證我們安全的有效性。若要了解適用於受管整合的合規計劃，請參閱AWS 合規計劃的 服務範圍。
- 雲端的安全性 – 您的責任取決於您使用 AWS 的服務。您也必須對其他因素負責，包括資料的機密性、您公司的要求和適用法律和法規。

本文件可協助您了解如何在使用 受管整合時套用共同責任模型。下列主題說明如何設定受管整合以符合您的安全與合規目標。您也會了解如何使用其他 AWS 服務來協助您監控和保護受管整合資源。

主題

受管整合中的資料保護

AWS 共同責任模型適用於 受管整合中的資料保護 AWS IoT Device Management。如此模型所述，AWS 負責保護執行所有 的 全球基礎設施 AWS 雲端。您負責維護在此基礎設施上託管內容的控制權。您也同時負責所使用 AWS 服務 的安全組態和管理任務。如需資料隱私權的詳細資訊，請參閱資料隱私權常見問答集。如需有關歐洲資料保護的相關資訊，請參閱AWS 安全性部落格上的AWS 共同責任模型和 GDPR 部落格文章。

基於資料保護目的，我們建議您保護 AWS 帳戶 登入資料，並使用 AWS IAM Identity Center 或 AWS Identity and Access Management (IAM) 設定個別使用者。如此一來，每個使用者都只會獲得授與完成其任務所必須的許可。我們也建議您採用下列方式保護資料：

- 每個帳戶均要使用多重要素驗證 (MFA)。
- 使用 SSL/TLS 與 AWS 資源通訊。我們需要 TLS 1.2 並建議使用 TLS 1.3。
- 使用 設定 API 和使用者活動記錄 AWS CloudTrail。如需有關使用 CloudTrail 追蹤擷取 AWS 活動的資訊，請參閱AWS CloudTrail 《 使用者指南》中的使用 CloudTrail 追蹤。
- 使用 AWS 加密解決方案，以及其中的所有預設安全控制 AWS 服務。
- 使用進階的受管安全服務 (例如 Amazon Macie)，協助探索和保護儲存在 Amazon S3 的敏感資料。
- 如果您在 AWS 透過命令列界面或 API 存取 時需要 FIPS 140-3 驗證的密碼編譯模組，請使用 FIPS 端點。如需有關 FIPS 和 FIPS 端點的更多相關資訊，請參閱聯邦資訊處理標準 (FIPS) 140-3。

我們強烈建議您絕對不要將客戶的電子郵件地址等機密或敏感資訊，放在標籤或自由格式的文字欄位中，例如名稱欄位。這包括當您使用 的受管整合 AWS IoT Device Management ，或使用主控台、API AWS CLI或 AWS SDKs的其他 AWS 服務 時。您在標籤或自由格式文字欄位中輸入的任何資料都可能用於計費或診斷日誌。如果您提供外部伺服器的 URL，我們強烈建議請勿在驗證您對該伺服器請求的 URL 中包含憑證資訊。

受管整合的靜態資料加密

根據預設， 的受管整合會使用 AWS IoT Device Management 加密金鑰來加密靜態敏感客戶資料。

加密金鑰有兩種類型，可用於保護受管整合客戶的敏感資料：

客戶受管金鑰 (CMK)

受管整合支援使用對稱客戶受管金鑰，您可以建立、擁有和管理這些金鑰。您可以完全控制這些 KMS 金鑰，包括建立和維護其金鑰政策、IAM 政策和授予、啟用和停用這些項目、輪換其密碼編譯材料、新增標籤、建立參考 KMS 金鑰的別名，以及排程 KMS 金鑰供刪除。

AWS 擁有的金鑰

根據預設，受管整合會使用這些金鑰自動加密敏感客戶資料。您無法檢視、管理或稽核其使用方式。您不需要採取任何動作或變更任何程式來保護加密資料的金鑰。依預設加密靜態資料，有助於降低保護敏感資料所涉及的營運開銷和複雜性。同時，其可讓您建置符合嚴格加密合規性和法規要求的安全應用程式。

使用的預設加密金鑰是 AWS 擁有的金鑰。或者，更新加密金鑰的選用 API 是 `PutDefaultEncryptionConfiguration`。

如需 AWS KMS 加密金鑰類型的詳細資訊，請參閱 AWS KMS 金鑰。

AWS KMS 受管整合的 用量

受管整合會使用信封加密來加密和解密所有客戶資料。這種類型的加密會取得您的純文字資料，並使用資料金鑰對其進行加密。接著，稱為包裝金鑰的加密金鑰會加密用於加密純文字資料的原始資料金鑰。在信封加密中，可以使用額外的包裝金鑰來加密現有包裝金鑰，這些金鑰與原始資料金鑰的分隔程度更接近。由於原始資料金鑰是由個別存放的包裝金鑰加密，因此您可以將原始資料金鑰和加密的純文字資料存放在相同的位置。除了用來加密和解密資料金鑰的包裝金鑰之外， keyring 也會用來產生、加密和解密資料金鑰。

> **Note**
>
> AWS Database Encryption SDK 為您的用戶端加密實作提供信封加密。如需 AWS 資料庫加密 SDK 的詳細資訊，請參閱什麼是 AWS 資料庫加密 SDK？

如需信封加密、資料金鑰、包裝金鑰和 keyring 的詳細資訊，請參閱信封加密、資料金鑰、包裝金鑰和 Keyring。

受管整合要求 服務將客戶受管金鑰用於下列內部操作：

- 傳送`DescribeKey`請求至 AWS KMS ，以驗證在輪換資料金鑰時提供的對稱客戶受管金鑰 ID。
- 將`GenerateDataKeyWithoutPlaintext`請求傳送至 AWS KMS ，以產生由客戶受管金鑰加密的資料金鑰。
- 將`ReEncrypt*`請求傳送至 AWS KMS ，以透過客戶受管金鑰重新加密資料金鑰。
- 將`Decrypt`請求傳送至 AWS KMS ，以透過客戶受管金鑰解密資料。

使用加密金鑰加密的資料類型

受管整合使用加密金鑰來加密靜態存放的多種資料類型。下列清單概述了使用加密金鑰進行靜態加密的資料類型：

- 雲端對雲端 (C2C) 連接器事件，例如裝置探索和裝置狀態更新。
- 建立代表實體裝置的受管物件，以及包含特定裝置類型功能的裝置描述檔。如需裝置和裝置設定檔的詳細資訊，請參閱 裝置和 裝置。
- 受管整合會通知您裝置實作的各個層面。如需受管整合通知的詳細資訊，請參閱 設定受管整合通知。

- 最終使用者的個人身分識別資訊 (PII)，例如裝置身分驗證資料、裝置序號、最終使用者名稱、裝置識別符和裝置 Amazon Resource Name (arn)。

受管整合如何在 中使用金鑰政策 AWS KMS

對於分支金鑰輪換和非同步呼叫，受管整合需要金鑰政策才能使用您的加密金鑰。金鑰政策的使用原因如下：

- 以程式設計方式授權其他 AWS 主體使用加密金鑰。

如需用於管理受管整合中加密金鑰存取權的金鑰政策範例，請參閱 建立加密金鑰

> **Note**
>
> 對於 AWS 擁有的金鑰，不需要金鑰政策，因為 AWS 擁有的金鑰是由 擁有 AWS ，您無法檢視、管理或使用它。根據預設，受管整合會使用 AWS 擁有的金鑰自動加密您的敏感客戶資料。

除了使用金鑰政策來使用 AWS KMS 金鑰管理加密組態之外， 受管整合也會使用 IAM 政策。如需 IAM 政策的詳細資訊，請參閱 中的政策和許可 AWS Identity and Access Management。

建立加密金鑰

您可以使用 AWS 管理主控台 或 AWS KMS APIs建立加密金鑰。

建立加密金鑰

請遵循《 AWS Key Management Service 開發人員指南》中建立 KMS 金鑰的步驟。

金鑰政策

金鑰政策陳述式控制對 AWS KMS 金鑰的存取。每個 AWS KMS 金鑰只會包含一個金鑰政策。該金鑰政策會決定哪些 AWS 主體可以使用金鑰，以及他們可以如何使用金鑰。如需使用 AWS KMS 金鑰政策陳述式管理金鑰存取和使用的詳細資訊，請參閱使用政策管理存取。

以下是金鑰政策陳述式的範例，可用於管理 中存放之 AWS KMS 金鑰的存取和用量 AWS 帳戶 ，以進行受管整合：

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

如需金鑰存放區的詳細資訊，請參閱金鑰存放區。

更新加密組態

無縫更新加密組態的功能對於管理受管整合的資料加密實作至關重要。當您最初使用 受管整合加入時，系統會提示您選取加密組態。您的選項將是預設 AWS 擁有的金鑰或建立您自己的 AWS KMS 金鑰。

AWS 管理主控台

若要在 中更新您的加密組態 AWS 管理主控台，請開啟 AWS IoT 服務首頁，然後導覽至 Managed Integration for Unified Control>Settings>Encryption。在加密設定視窗中，您可以選取新 AWS KMS 金鑰來更新加密組態，以提供額外的加密保護。選擇自訂加密設定 （進階） 以選取現有的 AWS KMS 金鑰，或者您可以選擇建立 AWS KMS 金鑰來建立自己的客戶受管金鑰。

API 命令

有兩個 APIs用於管理受管整合中 AWS KMS 金鑰的加密組態：`PutDefaultEncryptionConfiuration`和 `GetDefaultEncryptionConfiguration`。

若要更新預設加密組態，請呼叫 `PutDefaultEncryptionConfiuration`。如需 的詳細資訊`PutDefaultEncryptionConfiuration`，請參閱 PutDefaultEncryptionConfiuration。

若要檢視預設加密組態，請呼叫 `GetDefaultEncryptionConfiguration`。如需 的詳細資訊`GetDefaultEncryptionConfiguration`，請參閱 GetDefaultEncryptionConfiguration。

受管整合的身分和存取管理

AWS Identity and Access Management (IAM) 是 AWS 服務 ，可協助管理員安全地控制對 AWS 資源的存取。IAM 管理員可控制誰可以進行身分驗證 （登入） 和授權 （具有許可），以使用受管整合資源。IAM 是 AWS 服務 您可以免費使用的 。

主題

目標對象

使用方式 AWS Identity and Access Management (IAM) 會根據您的角色而有所不同：

- 服務使用者 — 若無法存取某些功能，請向管理員申請所需許可 (請參閱 對受管整合的身分和存取權進行故障診斷)
- 服務管理員 — 負責設定使用者存取權並提交相關許可請求 (請參閱 受管整合如何與 IAM 搭配使用)
- IAM 管理員 — 撰寫政策以管理存取控制 (請參閱 受管整合的身分型政策範例)

使用身分驗證

身分驗證是您 AWS 使用身分憑證登入 的方式。您必須以 AWS 帳戶根使用者、IAM 使用者或擔任 IAM 角色身分進行身分驗證。

您可以使用身分來源的登入資料，例如 AWS IAM Identity Center (IAM Identity Center)、單一登入身分驗證或 Google/Facebook 登入資料，以聯合身分的形式登入。如需有關登入的詳細資訊，請參閱《AWS 登入 使用者指南》中的如何登入您的 AWS 帳戶。

對於程式設計存取， AWS 提供 SDK 和 CLI 以密碼編譯方式簽署請求。如需詳細資訊，請參閱《IAM 使用者指南》中的 API 請求的AWS 第 4 版簽署程序。

AWS 帳戶 根使用者

當您建立 時 AWS 帳戶，您會從一個名為 AWS 帳戶 theroot 使用者的登入身分開始，該身分可完整存取所有 AWS 服務 和 資源。強烈建議不要使用根使用者來執行日常任務。有關需要根使用者憑證的任務，請參閱《IAM 使用者指南》中的需要根使用者憑證的任務。

聯合身分

最佳實務是要求人類使用者使用聯合身分提供者，以 AWS 服務 使用臨時憑證存取 。

聯合身分是您企業目錄、Web 身分提供者的使用者，或是 AWS 服務 使用身分來源的憑證 Directory Service 存取的使用者。聯合身分會擔任角色，而該角色會提供臨時憑證。

若需集中化管理存取權限，建議使用 AWS IAM Identity Center。如需詳細資訊，請參閱 AWS IAM Identity Center 使用者指南中的什麼是 IAM Identity Center？。

IAM 使用者和群組

IAM 使用者是一種身分，具備單一人員或應用程式的特定許可。建議以臨時憑證取代具備長期憑證的 IAM 使用者。如需詳細資訊，請參閱《IAM 使用者指南》中的要求人類使用者使用聯合身分提供者，以 AWS 使用臨時憑證存取 。

IAM 群組會指定 IAM 使用者集合，使管理大量使用者的許可更加輕鬆。如需詳細資訊，請參閱《IAM 使用者指南》中的 IAM 使用者的使用案例。

IAM 角色

IAM 角色是一種具特定許可的身分，可提供臨時憑證。您可以透過從使用者切換到 IAM 角色（主控台）或呼叫 AWS CLI 或 AWS API 操作來擔任角色。如需詳細資訊，請參閱《IAM 使用者指南》中的擔任角色的方法。

IAM 角色適用於聯合身分使用者存取、臨時 IAM 使用者許可、跨帳戶存取權與跨服務存取，以及在 Amazon EC2 執行的應用程式。如需詳細資訊，請參閱《IAM 使用者指南》中的 IAM 中的快帳戶資源存取。

使用政策管理存取權

您可以透過建立政策並將其連接到身分或資源 AWS 來控制 AWS 中的存取。政策定義與身分或資源相關聯的許可。當委託人提出請求時 AWS ， 會評估這些政策。大多數政策會以 JSON 文件 AWS 的形式存放在 中。如需進一步了解 JSON 政策文件，請參閱《IAM 使用者指南》中的 JSON 政策概觀。

管理員會使用政策，透過定義哪些主體可在哪些條件下對哪些資源執行動作，以指定可存取的範圍。

預設情況下，使用者和角色沒有許可。IAM 管理員會建立 IAM 政策並將其新增至角色，供使用者後續擔任。IAM 政策定義動作的許可，無論採用何種方式執行。

身分型政策

身分型政策是附加至身分 (使用者、使用者群組或角色) 的 JSON 許可政策文件。這類政策控制身分可對哪些資源執行哪些動作，以及適用的條件。如需了解如何建立身分型政策，請參閱《IAM 使用者指南》中的透過客戶管理政策定義自訂 IAM 許可。

身分型政策可分為內嵌政策 (直接內嵌於單一身分) 與受管政策 (可附加至多個身分的獨立政策)。如需了解如何在受管政策與內嵌政策之間選擇，請參閱《IAM 使用者指南》中的在受管政策和內嵌政策間選擇。

資源型政策

資源型政策是附加到資源的 JSON 政策文件。範例包括 IAM 角色信任政策與 Amazon S3 儲存貯體政策。在支援資源型政策的服務中，服務管理員可以使用它們來控制對特定資源的存取權限。您必須在資源型政策中指定主體。

資源型政策是位於該服務中的內嵌政策。您無法在資源型政策中使用來自 IAM 的 AWS 受管政策。

其他政策類型

AWS 支援其他政策類型，可設定更多常見政策類型授予的最大許可：

- 許可界限 — 設定身分型政策可授與 IAM 實體的最大許可。如需詳細資訊，請參閱《 IAM 使用者指南》中的 IAM 實體許可界限。
- 服務控制政策 (SCP) — 為 AWS Organizations中的組織或組織單位指定最大許可。如需詳細資訊，請參閱《AWS Organizations 使用者指南》中的服務控制政策。
- 資源控制政策 (RCP) — 設定您帳戶中資源可用許可的上限。如需詳細資訊，請參閱《AWS Organizations 使用者指南》中的資源控制政策 (RCP)。
- 工作階段政策 — 在以程式設計方式為角色或聯合身分使用者建立臨時工作階段時，以參數形式傳遞的進階政策。如需詳細資訊，請參《IAM 使用者指南》中的工作階段政策。

多種政策類型

當多種類型的政策適用於請求時，產生的許可會更複雜而無法理解。若要了解如何 AWS 在涉及多種政策類型時決定是否允許請求，請參閱《IAM 使用者指南》中的政策評估邏輯。

AWS 受管整合的 受管政策

若要新增許可給使用者、群組和角色，使用 AWS 受管政策比自行撰寫政策更容易。建立 IAM 客戶受管政策需要時間和專業知識，而受管政策可為您的團隊提供其所需的許可。若要快速開始使用，您可以使用我們的 AWS 受管政策。這些政策涵蓋常見的使用案例，並可在您的 AWS 帳戶中使用。如需 AWS 受管政策的詳細資訊，請參閱《IAM 使用者指南》中的 AWS 受管政策。

AWS 服務會維護和更新 AWS 受管政策。您無法變更 AWS 受管政策中的許可。服務偶爾會在 AWS 受管政策中新增其他許可以支援新功能。此類型的更新會影響已連接政策的所有身分識別 (使用者、群組和角色)。當新功能啟動或新操作可用時，服務很可能會更新 AWS 受管政策。服務不會從 AWS 受管政策中移除許可，因此政策更新不會破壞您現有的許可。

此外， AWS 支援跨多個 服務之任務函數的受管政策。例如，ReadOnlyAccess AWS 受管政策提供所有 AWS 服務和資源的唯讀存取權。當服務啟動新功能時， 會為新操作和資源 AWS 新增唯讀許可。如需任務職能政策的清單和說明，請參閱 IAM 使用者指南中有關任務職能的AWS 受管政策。

AWS 受管政策：AWSIoTManagedIntegrationsFullAccess

您可將 `AWSIoTManagedIntegrationsFullAccess` 政策連接到 IAM 身分。

此政策會授予受管整合和相關服務的完整存取許可。若要在 中檢視此政策 AWS 管理主控台，請參閱 AWSIoTManagedIntegrationsFullAccess。

許可詳細資訊

此政策包含以下許可：

- `iotmanagedintegrations` – 為您新增此政策的 IAM 使用者、群組和角色提供受管整合和相關服務的完整存取權。
- `iam` – 允許指派的 IAM 使用者、群組和角色在 中建立服務連結角色 AWS 帳戶。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iotmanagedintegrations:*",
      "Resource": "*"
```

```
    },
    {
      "Effect": "Allow",
      "Action": "iam:CreateServiceLinkedRole",
      "Resource": "arn:aws:iam::*:role/aws-service-role/
iotmanagedintegrations.amazonaws.com/AWSServiceRoleForIoTManagedIntegrations",
      "Condition": {
        "StringEquals": {
          "iam:AWSServiceName": "iotmanagedintegrations.amazonaws.com"
        }
      }
    }
  ]
}
```

AWS 受管政策： AWS IoTManagedIntegrationsRolePolicy

您可將 `AWS IoTManagedIntegrationsRolePolicy` 政策連接到 IAM 身分。

此政策授予受管整合代表您發佈 Amazon CloudWatch logs和指標的許可。

若要在 中檢視此政策 AWS 管理主控台，請參閱 AWSIoTManagedIntegrationsRolePolicy。

許可詳細資訊

此政策包含以下許可。

- `logs` – 提供建立 Amazon CloudWatch 日誌群組並將日誌串流至群組的功能。
- `cloudwatch` – 提供發佈 Amazon CloudWatch 指標的功能。如需 Amazon CloudWatch 指標的詳細資訊，請參閱 Amazon CloudWatch 中的指標。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "CloudWatchLogs",
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup"
```

```
      ],
      "Resource": [
        "arn:aws:logs:*:*:log-group:/aws/iotmanagedintegrations/*"
      ],
      "Condition": {
        "StringEquals": {
          "aws:PrincipalAccount": "${aws:ResourceAccount}"
        }
      }
    },
    {
      "Sid": "CloudWatchStreams",
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogStream",
        "logs:PutLogEvents"
      ],
      "Resource": [
        "arn:aws:logs:*:*:log-group:/aws/iotmanagedintegrations/*:log-stream:*"
      ],
      "Condition": {
        "StringEquals": {
          "aws:PrincipalAccount": "${aws:ResourceAccount}"
        }
      }
    },
    {
      "Sid": "CloudWatchMetrics",
      "Effect": "Allow",
      "Action": [
        "cloudwatch:PutMetricData"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "cloudwatch:namespace": [
            "AWS/IoTManagedIntegrations",
            "AWS/Usage"
          ]
        }
      }
    }
  ]
```

```
}
```

受管整合更新 AWS 受管政策

檢視自此服務開始追蹤這些變更以來，受管整合的 AWS 受管政策更新詳細資訊。如需此頁面變更的自動提醒，請訂閱受管整合文件歷史記錄頁面上的 RSS 摘要。

變更	描述	Date
受管整合已開始追蹤變更	受管整合開始追蹤其 AWS 受管政策的變更。	2025 年 3 月 3 日

受管整合如何與 IAM 搭配使用

在您使用 IAM 管理受管整合的存取權之前，請先了解哪些 IAM 功能可與受管整合搭配使用。

您可以搭配 受管整合使用的 IAM 功能

IAM 功能	受管整合支援
身分型政策	是
資源型政策	否
政策動作	是
政策資源	是
政策條件索引鍵	是
ACL	否
ABAC(政策中的標籤)	否
臨時憑證	是
主體許可	是
服務角色	是

IAM 功能	受管整合支援
服務連結角色	是

若要全面了解受管整合和其他 AWS 服務如何與大多數 IAM 功能搭配使用，請參閱《IAM 使用者指南》中的AWS 與 IAM 搭配使用的 服務。

受管整合的身分型政策

支援身分型政策：是

身分型政策是可以附加到身分 (例如 IAM 使用者、使用者群組或角色) 的 JSON 許可政策文件。這些政策可控制身分在何種條件下能對哪些資源執行哪些動作。如需了解如何建立身分型政策，請參閱《IAM 使用者指南》中的透過客戶管理政策定義自訂 IAM 許可。

使用 IAM 身分型政策，您可以指定允許或拒絕的動作和資源，以及在何種條件下允許或拒絕動作。如要了解您在 JSON 政策中使用的所有元素，請參閱《IAM 使用者指南》中的 IAM JSON 政策元素參考。

受管整合的身分型政策範例

若要檢視受管整合身分型政策的範例，請參閱 受管整合的身分型政策範例。

受管整合中的資源型政策

支援資源型政策：否

資源型政策是附加到資源的 JSON 政策文件。資源型政策的最常見範例是 IAM 角色信任政策和 Amazon S3 儲存貯體政策。在支援資源型政策的服務中，服務管理員可以使用它們來控制對特定資源的存取權限。對於附加政策的資源，政策會定義指定的主體可以對該資源執行的動作以及在何種條件下執行的動作。您必須在資源型政策中指定主體。委託人可以包含帳戶、使用者、角色、聯合身分使用者或 AWS 服務。

如需啟用跨帳戶存取權，您可以在其他帳戶內指定所有帳戶或 IAM 實體作為資源型政策的主體。如需詳細資訊，請參閱《IAM 使用者指南》中的 IAM 中的快帳戶資源存取。

受管整合的政策動作

支援政策動作：是

管理員可以使用 AWS JSON 政策來指定誰可以存取內容。也就是說，哪個主體在什麼條件下可以對什麼資源執行哪些動作。

JSON 政策的 `Action` 元素描述您可以用來允許或拒絕政策中存取的動作。政策會使用動作來授予執行相關聯動作的許可。

若要查看受管整合動作的清單，請參閱《服務授權參考》中的受管整合定義的動作。

受管整合中的政策動作在動作之前使用下列字首：

```
iot-mi
```

若要在單一陳述式中指定多個動作，請用逗號分隔。

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

若要檢視受管整合身分型政策的範例，請參閱 受管整合的身分型政策範例。

受管整合的政策資源

支援政策資源：是

管理員可以使用 AWS JSON 政策來指定誰可以存取內容。也就是說，哪個主體在什麼條件下可以對什麼資源執行哪些動作。

`Resource` JSON 政策元素可指定要套用動作的物件。最佳實務是使用其 Amazon Resource Name (ARN) 來指定資源。若動作不支援資源層級許可，使用萬用字元 (*) 表示該陳述式適用於所有資源。

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

若要查看受管整合資源類型及其 ARNs，請參閱《服務授權參考》中的受管整合定義的資源。若要了解您可以使用哪些動作指定每個資源的 ARN，請參閱受管整合定義的動作。

若要檢視受管整合身分型政策的範例，請參閱 受管整合的身分型政策範例。

受管整合的政策條件索引鍵

支援服務特定政策條件金鑰：是

管理員可以使用 AWS JSON 政策來指定誰可以存取內容。也就是說，哪個主體在什麼條件下可以對什麼資源執行哪些動作。

Condition 元素會根據定義的條件，指定陳述式的執行時機。您可以建立使用條件運算子的條件運算式 (例如等於或小於)，來比對政策中的條件和請求中的值。若要查看所有 AWS 全域條件索引鍵，請參閱《IAM 使用者指南》中的AWS 全域條件內容索引鍵。

若要查看受管整合條件金鑰的清單，請參閱《服務授權參考》中的受管整合的條件金鑰。若要了解您可以使用條件金鑰的動作和資源，請參閱 受管整合定義的動作。

若要檢視受管整合身分型政策的範例，請參閱 受管整合的身分型政策範例。

受管整合中的 ACLs

支援 ACL：否

存取控制清單 (ACL) 可控制哪些主體 (帳戶成員、使用者或角色) 擁有存取某資源的許可。ACL 類似於資源型政策，但它們不使用 JSON 政策文件格式。

ABAC 與受管整合

支援 ABAC (政策中的標籤)：部分

屬性型存取控制 (ABAC) 是一種授權策略，根據稱為標籤的屬性定義許可權。您可以將標籤連接至 IAM 實體 AWS 和資源，然後設計 ABAC 政策，以便在委託人的標籤符合資源上的標籤時允許操作。

如需根據標籤控制存取，請使用 aws:ResourceTag/*key-name*、aws:RequestTag/*key-name* 或 aws:TagKeys 條件索引鍵，在政策的條件元素中，提供標籤資訊。

如果服務支援每個資源類型的全部三個條件金鑰，則對該服務而言，值為 Yes。如果服務僅支援某些資源類型的全部三個條件金鑰，則值為 Partial。

如需 ABAC 的詳細資訊，請參閱《IAM 使用者指南》中的使用 ABAC 授權定義許可。如要查看含有設定 ABAC 步驟的教學課程，請參閱《IAM 使用者指南》中的使用屬性型存取控制 (ABAC)。

搭配 受管整合使用臨時登入資料

支援臨時憑證：是

臨時登入資料提供對 AWS 資源的短期存取，並在您使用聯合或切換角色時自動建立。 AWS 建議您動態產生臨時登入資料，而不是使用長期存取金鑰。如需詳細資訊，請參閱《IAM 使用者指南》中的IAM 中的臨時安全憑證與可與 IAM 搭配運作的AWS 服務。

受管整合的跨服務主體許可

支援轉寄存取工作階段 (FAS)：是

轉送存取工作階段 (FAS) 使用呼叫 的委託人許可 AWS 服務，並結合 請求 AWS 服務 向下游服務提出請求。如需提出 FAS 請求時的政策詳細資訊，請參閱轉發存取工作階段。

受管整合的服務角色

支援服務角色：是

服務角色是服務擔任的 IAM 角色，可代您執行動作。IAM 管理員可以從 IAM 內建立、修改和刪除服務角色。如需詳細資訊，請參閱《IAM 使用者指南》中的建立角色以委派許可給 AWS 服務。

> **Warning**
>
> 變更服務角色的許可可能會中斷受管整合功能。只有在受管整合提供指引時，才能編輯服務角色。

受管整合的服務連結角色

支援服務連結角色：是

服務連結角色是連結至 的一種服務角色 AWS 服務。服務可以擔任代表您執行動作的角色。服務連結角色會出現在您的 中 AWS 帳戶 ，並由服務擁有。IAM 管理員可以檢視，但不能編輯服務連結角色的許可。

如需建立或管理服務連結角色的詳細資訊，請參閱可搭配 IAM 運作的AWS 服務。在資料表中尋找服務，其中包含服務連結角色欄中的 Yes。選擇是連結，以檢視該服務的服務連結角色文件。

受管整合的身分型政策範例

根據預設，使用者和角色沒有建立或修改受管整合資源的許可。若要授予使用者對其所需資源執行動作的許可，IAM 管理員可以建立 IAM 政策。

如需了解如何使用這些範例 JSON 政策文件建立 IAM 身分型政策，請參閱《IAM 使用者指南》中的建立 IAM 政策 (主控台)。

如需受管整合定義之動作和資源類型的詳細資訊，包括每種資源類型的 ARNs 格式，請參閱《服務授權參考》中的受管整合的動作、資源和條件金鑰。

主題

- 政策最佳實務
- 使用 受管整合主控台
- 允許使用者檢視他們自己的許可

政策最佳實務

身分型政策會判斷您帳戶中的某個人員是否可以建立、存取或刪除受管整合資源。這些動作可能會讓您的 AWS 帳戶產生費用。當您建立或編輯身分型政策時，請遵循下列準則及建議事項：

- 開始使用 AWS 受管政策並邁向最低權限許可 – 若要開始將許可授予您的使用者和工作負載，請使用將許可授予許多常見使用案例的 AWS 受管政策。它們可在您的 中使用 AWS 帳戶。我們建議您定義特定於使用案例 AWS 的客戶受管政策，以進一步減少許可。如需更多資訊，請參閱《IAM 使用者指南》中的 AWS 受管政策或任務職能的AWS 受管政策。
- 套用最低權限許可 – 設定 IAM 政策的許可時，請僅授予執行任務所需的許可。為實現此目的，您可以定義在特定條件下可以對特定資源採取的動作，這也稱為最低權限許可。如需使用 IAM 套用許可的更多相關資訊，請參閱《IAM 使用者指南》中的 IAM 中的政策和許可。
- 使用 IAM 政策中的條件進一步限制存取權 – 您可以將條件新增至政策，以限制動作和資源的存取。例如，您可以撰寫政策條件，指定必須使用 SSL 傳送所有請求。如果透過特定 例如 使用服務動作 AWS 服務，您也可以使用條件來授予其存取權 CloudFormation。如需詳細資訊，請參閱《IAM 使用者指南》中的 IAM JSON 政策元素：條件。
- 使用 IAM Access Analyzer 驗證 IAM 政策，確保許可安全且可正常運作 – IAM Access Analyzer 驗證新政策和現有政策，確保這些政策遵從 IAM 政策語言 (JSON) 和 IAM 最佳實務。IAM Access Analyzer 提供 100 多項政策檢查及切實可行的建議，可協助您撰寫安全且實用的政策。如需詳細資訊，請參閱《IAM 使用者指南》中的使用 IAM Access Analyzer 驗證政策。
- 需要多重要素驗證 (MFA) – 如果您的案例需要 IAM 使用者或 中的根使用者 AWS 帳戶，請開啟 MFA 以提高安全性。如需在呼叫 API 操作時請求 MFA，請將 MFA 條件新增至您的政策。如需詳細資訊，請參閱《IAM 使用者指南》中的透過 MFA 的安全 API 存取。

如需 IAM 中最佳實務的相關資訊，請參閱《IAM 使用者指南》中的 IAM 安全最佳實務。

使用 受管整合主控台

若要存取受管整合主控台，您必須擁有一組最低許可。這些許可必須允許您列出和檢視 中受管整合資源的詳細資訊 AWS 帳戶。如果您建立比最基本必要許可更嚴格的身分型政策，則對於具有該政策的實體 (使用者或角色) 而言，主控台就無法如預期運作。

對於僅呼叫 AWS CLI 或 AWS API 的使用者，您不需要允許最低主控台許可。反之，只需允許存取符合他們嘗試執行之 API 操作的動作就可以了。

為了確保使用者和角色仍然可以使用受管整合主控台，也請將受管整合*ConsoleAccess*或*ReadOnly* AWS 受管政策連接到實體。如需詳細資訊，請參閱《IAM 使用者指南》中的新增許可到使用者。

允許使用者檢視他們自己的許可

此範例會示範如何建立政策，允許 IAM 使用者檢視附加到他們使用者身分的內嵌及受管政策。此政策包含在主控台或使用 或 AWS CLI AWS API 以程式設計方式完成此動作的許可。

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

對受管整合的身分和存取權進行故障診斷

使用以下資訊來協助您診斷和修正使用受管整合和 IAM 時可能遇到的常見問題。

主題

- 我無權在受管整合中執行 動作
- 我未獲得執行 iam:PassRole 的授權
- 我想要允許 以外的人員 AWS 帳戶 存取我的受管整合資源

我無權在受管整合中執行 動作

如果您收到錯誤，告知您未獲授權執行動作，您的政策必須更新，允許您執行動作。

下列範例錯誤會在mateojackson IAM 使用者嘗試使用主控台檢視一個虛構 *my-example-widget* 資源的詳細資訊，但卻無虛構 iot-mi:*GetWidget* 許可時發生。

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

在此情況下，必須更新 mateojackson 使用者的政策，允許使用 iot-mi:*GetWidget* 動作存取 *my-example-widget* 資源。

如果您需要協助，請聯絡您的 AWS 管理員。您的管理員提供您的簽署憑證。

我未獲得執行 iam:PassRole 的授權

如果您收到錯誤，告知您無權執行 iam:PassRole動作，您的政策必須更新，以允許您將角色傳遞至受管整合。

有些 AWS 服務 可讓您將現有角色傳遞給該服務，而不是建立新的服務角色或服務連結角色。如需執行此作業，您必須擁有將角色傳遞至該服務的許可。

當名為 的 IAM marymajor 使用者嘗試使用主控台在受管整合中執行動作時，會發生下列範例錯誤。但是，動作請求服務具備服務角色授予的許可。Mary 沒有將角色傳遞給服務的許可。

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

在這種情況下，Mary 的政策必須更新，允許她執行 iam:PassRole 動作。

如果您需要協助，請聯絡您的 AWS 管理員。您的管理員提供您的簽署憑證。

我想要允許 以外的人員 AWS 帳戶 存取我的受管整合資源

您可以建立一個角色，讓其他帳戶中的使用者或您組織外部的人員存取您的資源。您可以指定要允許哪些信任物件取得該角色。針對支援基於資源的政策或存取控制清單 (ACL) 的服務，您可以使用那些政策來授予人員存取您的資源的許可。

如需進一步了解，請參閱以下內容：

- 若要了解受管整合是否支援這些功能，請參閱 受管整合如何與 IAM 搭配使用。
- 若要了解如何提供您擁有 AWS 帳戶 的資源存取權，請參閱《IAM 使用者指南》中的在您擁有 AWS 帳戶 的另一個 中為 IAM 使用者提供存取權。
- 若要了解如何將資源的存取權提供給第三方 AWS 帳戶，請參閱《IAM 使用者指南》中的將存取權提供給第三方 AWS 帳戶 擁有。
- 如需了解如何透過聯合身分提供存取權，請參閱《IAM 使用者指南》中的將存取權提供給在外部進行身分驗證的使用者 (聯合身分)。
- 如需了解使用角色和資源型政策進行跨帳戶存取之間的差異，請參閱《IAM 使用者指南》中的 IAM 中的跨帳戶資源存取。

使用服務連結角色進行受管整合

AWS IoT Device Management 的受管整合使用 AWS Identity and Access Management (IAM) 服務連結角色。服務連結角色是直接連結至受管整合的唯一 IAM 角色類型。服務連結角色由 受管整合預先定義，並包含服務代表您呼叫其他 AWS 服務所需的所有許可。

服務連結角色可讓您更輕鬆地設定受管整合，因為您不必手動新增必要的許可。 AWS IoT Device Management 的受管整合會定義其服務連結角色的許可，除非另有定義，否則只有受管整合才能擔任其角色。定義的許可包括信任政策和許可政策，且該許可政策無法附加至其他 IAM 實體。

您必須先刪除服務連結角色的相關資源，才能將其刪除。這可保護您的受管整合資源，因為您不會不小心移除存取資源的許可。

如需有關支援服務連結角色的其他 服務的資訊，請參閱AWS 使用 IAM 的服務，並在服務連結角色欄中尋找具有是的服務。選擇具有連結的是，以檢視該服務的服務連結角色文件。

受管整合的服務連結角色許可

AWS IoT Device Management 的受管整合使用名為 AWSServiceRoleForIoTManagedIntegrations 的服務連結角色 – 為 AWS IoT Device Management 提供受管整合，以代表您發佈日誌和指標。

AWSServiceRoleForIoTManagedIntegrations 服務連結角色信任下列服務擔任該角色：

- `iotmanagedintegrations.amazonaws.com`

名為 AWSIoTManagedIntegrationsServiceRolePolicy 的角色許可政策允許受管整合對指定的資源完成下列動作：

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

JSON

```
{
  "Version":"2012-10-17",
  "Statement" : [
    {
      "Sid" : "CloudWatchLogs",
      "Effect" : "Allow",
      "Action" : [
        "logs:CreateLogGroup",
        "logs:DescribeLogGroups"
      ],
      "Resource" : [
        "arn:aws:logs:*:*:log-group:/aws/iotmanagedintegrations/*"
      ]
    },
    {
      "Sid" : "CloudWatchStreams",
```

```
      "Effect" : "Allow",
      "Action" : [
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams"
      ],
      "Resource" : [
        "arn:aws:logs:*:*:log-group:/aws/iotmanagedintegrations/*:log-stream:*"
      ]
    },
    {
      "Sid" : "CloudWatchMetrics",
      "Effect" : "Allow",
      "Action" : [
        "cloudwatch:PutMetricData"
      ],
      "Resource" : "*",
      "Condition" : {
        "StringEquals" : {
          "cloudwatch:namespace" : [
            "AWS/IoTManagedIntegrations",
            "AWS/Usage"
          ]
        }
      }
    }
  ]
}
```

您必須設定許可，以允許您的使用者、群組或角色建立、編輯或刪除服務連結角色。如需詳細資訊，請參閱 IAM 使用者指南中的服務連結角色許可。

為受管整合建立服務連結角色

您不需要手動建立服務連結角色，當您在 CreateEventLogConfiguration、或 RegisterCustomEndpoint API 中呼叫 PutRuntimeLogConfiguration、AWS 管理主控台 AWS CLI或 AWS API 命令等事件類型時， 受管整合會為您建立服務連結角色。如需 PutRuntimeLogConfiguration、CreateEventLogConfiguration或 的詳細資訊RegisterCustomEndpoint，請參閱 PutRuntimeLogConfiguration、CreateEventLogConfiguration 或 RegisterCustomEndpoint。

若您刪除此服務連結角色，之後需要再次建立，您可以在帳戶中使用相同程序重新建立角色。當您呼叫 PutRuntimeLogConfiguration、 CreateEventLogConfiguration或 RegisterCustomEndpoint API 命令等事件類型時，受管整合會再次為您建立服務連結角色。或者，您也可以透過 聯絡 AWS 客戶支援 AWS Support Center Console。如需 AWS Support Plans 的詳細資訊，請參閱比較 AWS Support Plans。

您也可以使用 IAM 主控台，透過 IoT ManagedIntegrations - Managed Role 使用案例來建立服務連結角色。在 AWS CLI 或 AWS API 中，使用服務名稱建立iotmanagedintegrations.amazonaws.com服務連結角色。如需詳細資訊，請參閱《IAM 使用者指南》中的「建立服務連結角色」。如果您刪除此服務連結角色，您可以使用此相同的程序以再次建立該角色。

編輯受管整合的服務連結角色

受管整合不允許您編輯 AWSServiceRoleForIoTManagedIntegrations 服務連結角色。因為有各種實體可能會參考服務連結角色，所以您無法在建立角色之後變更角色名稱。然而，您可使用 IAM 來編輯角色描述。如需詳細資訊，請參閱《IAM 使用者指南》中的編輯服務連結角色。

刪除受管整合的服務連結角色

若您不再使用需要服務連結角色的功能或服務，我們建議您刪除該角色。如此一來，您就沒有未主動監控或維護的未使用實體。然而，在手動刪除服務連結角色之前，您必須先清除資源。

> **Note**
>
> 如果您嘗試刪除資源時，受管整合正在使用 角色，則刪除可能會失敗。若此情況發生，請等待數分鐘後並再次嘗試操作。

使用 IAM 手動刪除服務連結角色

使用 IAM 主控台 AWS CLI、 或 AWS API 來刪除 AWSServiceRoleForIoTManagedIntegrations 服務連結角色。如需詳細資訊，請參閱《IAM 使用者指南》中的刪除服務連結角色。

受管整合服務連結角色支援的 區域

AWS IoT Device Management 的受管整合支援在提供服務的所有區域中使用服務連結角色。如需詳細資訊，請參閱 AWS Regions and endpoints。

AWS Secrets Manager 用於 C2C 工作流程的資料保護

AWS Secrets Manager 是一項秘密儲存服務，可用來保護資料庫登入資料、API 金鑰和其他秘密資訊。然後，在您的程式碼中，您可以將硬式編碼登入資料取代為對 Secrets Manager 的 API 呼叫。這有助於確保檢查程式碼的人員不會洩露秘密，因為秘密不存在。如需概觀，若要取得概述，請參閱《AWS Secrets Manager 使用指南》。

Secrets Manager 會使用 AWS Key Management Service 金鑰加密秘密。如需詳細資訊，請參閱 AWS Key Management Service中的機密加密和解密。

的受管整合與 AWS IoT Device Management 整合， AWS Secrets Manager 因此您可以將資料存放在 Secrets Manager 中，並在組態中使用秘密 ID。

受管整合如何使用秘密

開放授權 (OAuth) 是委派存取授權的開放標準，可讓使用者授予網站或應用程式在其他網站上存取其資訊的權限，而無需共用其密碼。這是第三方應用程式代表使用者存取使用者資料的安全方式，為共用密碼提供更安全的替代方案。

在 OAuth 中，用戶端 ID 和用戶端秘密是在請求存取字符時識別和驗證用戶端應用程式的登入資料。

的受管整合 AWS IoT Device Management 使用 OAuth 與使用 C2C 工作流程的客戶通訊。客戶需要提供用戶端 ID 和用戶端秘密才能通訊。受管整合客戶將在其 AWS 帳戶中存放用戶端 ID 和用戶端秘密，而受管整合會在客戶帳戶中讀取用戶端 ID 和用戶端秘密。

如何建立秘密

若要建立秘密，請遵循 AWS Secrets Manager 《 使用者指南》中的建立 AWS Secrets Manager 秘密中的步驟。

您必須使用客戶受管 AWS KMS 金鑰建立秘密，受管整合才能讀取秘密值。如需詳細資訊，請參閱 AWS Secrets Manager 《 使用者指南》中的AWS KMS 金鑰許可。

您還必須使用下一節中的 IAM 政策。

授予 受管整合的存取權 AWS IoT Device Management ，讓 擷取秘密

若要允許受管整合從 Secrets Manager 擷取秘密值，請在建立秘密時，在秘密的資源政策中包含下列許可。

JSON

```
{
  "Version":"2012-10-17",
  "Statement" : [ {
    "Effect" : "Allow",
    "Principal" : {
      "Service" : "iotmanagedintegrations.amazonaws.com"
    },
    "Action" : [ "secretsmanager:GetSecretValue" ],
    "Resource" : "*" ,
    "Condition": {
        "StringEquals": {
          "aws:SourceArn": "arn:aws:iotmanagedintegrations:AWS Region:account-
id:account-association:account-association-id"

        }
      }
  } ]
}
```

將下列陳述式新增至客戶受管 AWS KMS 金鑰的政策。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [{
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt",
                "kms:DescribeKey"
            ],
            "Principal": {
                "Service": [
                    "iotmanagedintegrations.amazonaws.com"
                ]
            },
            "Resource": [
            "arn:aws:kms:us-east-1:123456789012:key/*"
            ]
```

```
        }

    ]
}
```

受管整合的合規驗證

若要了解 是否 AWS 服務 在特定合規計劃範圍內，請參閱AWS 服務 合規計劃範圍內然後選擇您感興趣的合規計劃。如需一般資訊，請參閱AWS 合規計劃。

您可以使用 下載第三方稽核報告 AWS Artifact。如需詳細資訊，請參閱下載報告 in AWS Artifact

您使用 時的合規責任 AWS 服務 取決於資料的機密性、您公司的合規目標，以及適用的法律和法規。如需使用 時合規責任的詳細資訊 AWS 服務，請參閱 AWS 安全文件。

搭配界面 VPC 端點使用受管整合

您可以透過建立界面 Amazon VPC 端點，在 Amazon VPC 與 AWS IoT 受管整合之間建立私有連線。介面端點採用一種技術 AWS PrivateLink，可讓您使用私有 IP 地址來私下存取 服務。AWS PrivateLink 會將 VPC 和 IoT 受管整合之間的所有網路流量限制在 Amazon 網路。您不需要網際網路閘道、NAT 裝置或 VPN 連線。

您不需要使用 AWS PrivateLink，但建議使用。如需 AWS PrivateLink 和 VPC 端點的詳細資訊，請參閱《 AWS PrivateLink 指南》中的透過 存取 AWS 服務 AWS PrivateLink。

主題

- AWS IoT 受管整合 VPC 端點的考量
- 為 AWS IoT 受管整合建立介面 VPC 端點
- 測試您的 VPC 端點
- 透過 VPC 端點控制對 服務的存取
- 定價
- 限制

AWS IoT 受管整合 VPC 端點的考量

在為 AWS IoT 受管整合設定界面 VPC 端點之前，請檢閱《 AWS PrivateLink 指南》中的界面端點屬性和限制。

AWS IoT 受管整合支援透過界面 VPC 端點從 VPC 呼叫其所有 API 動作。

支援的端點

AWS IoT 受管整合支援下列服務介面的 VPC 端點：

- 控制平面 API： `com.amazonaws.region.iotmanagedintegrations.api`

不支援的端點

下列 AWS IoT 受管整合端點不支援 VPC 端點：

- MQTT 端點：MQTT 裝置通常部署在最終使用者環境中，而不是 AWS VPCs內，因此不需要 AWS PrivateLink 整合。
- OAuth 回呼端點：許多第三方平台無法在 AWS 基礎設施內運作，減少 AWS PrivateLink 支援 OAuth 流程的好處。

可用性

AWS IoT 受管整合 VPC 端點可在下列 AWS 區域使用：

- 加拿大 (中部) – ca-central-1
- 歐洲 (愛爾蘭) – eu-west-1

隨著 AWS IoT 受管整合擴展其可用性，將支援其他區域。

雙堆疊支援

AWS IoT 受管整合 VPC 端點支援 IPv4 和 IPv6 流量。您可以使用下列 IP 地址類型建立 VPC 端點：

- IPv4：將 IPv4 地址指派給端點網路介面
- IPv6：將 IPv6 地址指派給端點網路介面 （需要IPv6-only 的子網路）
- Dualstack：將 IPv4 和 IPv6 地址指派給端點網路介面

為 AWS IoT 受管整合建立介面 VPC 端點

您可以使用 Amazon VPC 主控台或 AWS CLI (AWS CLI) 為 AWS IoT 受管整合服務建立 VPC 端點。

為 AWS IoT 受管整合建立介面 VPC 端點 （主控台）

1. 在 Amazon VPC 主控台開啟 Amazon VPC 主控台。
2. 在導覽窗格中選擇端點。
3. 選擇建立端點。
4. 在 Service category (服務類別) 中，選擇 AWS services。
5. 針對服務名稱，選取對應至您 AWS 區域的服務名稱。例如：
   - `com.amazonaws.ca-central-1.iotmanagedintegrations.api`
   - `com.amazonaws.eu-west-1.iotmanagedintegrations.api`
6. 針對 VPC，選取您要從中存取 AWS IoT 受管整合的 VPC。
7. 對於其他設定，預設會選取啟用 DNS 名稱。我們建議您保留此設定。這可確保對 AWS IoT 受管整合公有服務端點的請求解析為您的 Amazon VPC 端點。
8. 針對子網路，選取要在其中建立端點網路介面的子網路。您可以為每個可用區域選擇一個子網路。
9. 針對 IP address type (IP 地址類型)，從下列選項中選擇：
   - IPv4：將 IPv4 地址指派給端點網路介面
   - IPv6：將 IPv6 地址指派給端點網路介面 （只有在所有選取的子網路都是IPv6-only時才支援）
   - Dualstack：將 IPv4 和 IPv6 地址指派給端點網路介面
10. 對於 Security group (安全群組)，選取要與端點網路介面建立關聯的安全群組。安全群組規則必須允許端點網路介面與 VPC 中與服務通訊的資源之間進行通訊。
11. 針對政策，選擇完整存取，以允許所有主體對界面端點上所有資源的所有操作。若要限制存取，請選擇自訂並指定政策。
12. (選用) 若要新增標籤，請選擇 Add new tag (新增標籤)，然後輸入標籤金鑰和值。
13. 選擇建立端點。

為 IoT 受管整合 (AWS CLI) 建立介面 VPC 端點

使用 create-vpc-endpoint 命令，並指定 VPC ID、VPC 端點類型 （介面）、服務名稱、將使用端點的子網路，以及與端點網路介面建立關聯的安全群組。

```
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
```

測試您的 VPC 端點

建立 VPC 端點之後，您可以從 VPC 中的 EC2 執行個體對 AWS IoT 受管整合進行 API 呼叫，以測試連線。

先決條件

- VPC 內私有子網路中的 EC2 執行個體
- AWS IoT 受管整合操作的適當 IAM 許可
- 允許 HTTPS 流量 （連接埠 443) 到 VPC 端點的安全群組規則

測試連線

1. 連線至私有子網路中的 Amazon EC2 執行個體。
2. 驗證私有 DNS 名稱的 DNS 解析：

```
dig api.iotmanagedintegrations.region.api.aws
```

3. 測試 HTTPS 連線能力：

```
curl -v https://api.iotmanagedintegrations.region.api.aws
```

4. 進行 AWS IoT 受管整合 API 呼叫：

```
aws iot-managed-integrations list-destinations \
  --region region \
  --endpoint-url https://api.iotmanagedintegrations.region.api.aws
```

將 取代`region`為您的 AWS 區域 （例如 `ca-central-1`)。

透過 VPC 端點控制對 服務的存取

VPC 端點政策是您在建立或修改端點時連接到介面 VPC 端點的 IAM 資源政策。如果您未在建立端點時連接政策，我們會以預設政策連接以允許完整存取服務。端點政策不會覆寫或取代 IAM 使用者政策或服務特定的政策。這個另行區分的政策會控制從端點到所指定之服務的存取。

端點政策必須以 JSON 格式撰寫。如需詳細資訊，請參閱 Amazon VPC 使用者指南中的使用 VPC 端點控制對服務的存取。

範例： AWS IoT 受管整合動作的 VPC 端點政策

以下是 AWS IoT 受管整合的端點政策範例。此政策允許使用者透過 VPC 端點連線至 AWS IoT 受管整合，以存取目的地，但拒絕存取登入資料儲存庫。

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

範例：限制存取特定 IAM 角色的 VPC 端點政策

下列 VPC 端點政策僅允許在其信任鏈中具有指定 IAM 角色的 IAM 主體存取 AWS IoT 受管整合。所有其他 IAM 主體都會遭到拒絕存取。

```
{
```

```
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "*",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:PrincipalArn": "arn:aws:iam::123456789012:role/
IoTManagedIntegrationsVPCRole"
                }
            }
        }
    ]
}
```

定價

您需要支付建立和使用介面 VPC 端點與 AWS IoT 受管整合的標準費率。如需詳細資訊，請參閱 AWS PrivateLink 定價。

限制

- CreateAccountAssociation API 旨在透過第三方雲端服務執行 OAuth，這需要請求離開 Amazon 網路。這對於使用 的客戶而言很重要， AWS PrivateLink 以在 VPC 中包含其流量，因為 AWS PrivateLink 無法為此 API 呼叫提供完整的end-to-end限制。
- AWS IoT 受管整合的 VPC 端點無法在 中使用 AWS GovCloud (US) Regions。

如需一般 VPC 端點限制，請參閱《Amazon VPC 使用者指南》中的界面端點屬性和限制。

連線至 AWS IoT Device Management FIPS 端點的受管整合

AWS IoT 提供支援聯邦資訊處理標準 (FIPS) 140-2 的控制平面端點。FIPS 相容端點與標準 AWS 端點不同。若要以符合 FIPS 規範的方式與 的 AWS IoT Device Management 受管整合互動，您必須將下列端點與符合 FIPS 規範的用戶端搭配使用。 AWS IoT 主控台不符合 FIPS 規範。

下列各節說明如何使用 REST API、 SDK 或 存取 FIPS 相容 AWS IoT 端點 AWS CLI。

控制平面端點

支援受管整合操作及其相關 AWS CLI 命令的 FIPS 相容控制平面端點會列在 FIPS Endpoints by Service 中。在 FIPS Endpoints by Service 中，尋找 AWS IoT Device Management 受管整合服務，並查詢 的端點 AWS 區域。

若要在存取受管整合操作時使用 FIPS 相容端點，請使用 AWS SDK 或 REST API 搭配適合您的端點 AWS 區域。

若要在執行受管整合 CLI 命令時使用 FIPS 相容端點，請將 --endpoint 參數與適用於 的端點新增至 AWS 區域 命令。

監控受管整合

監控是維護受管整合和其他 AWS 解決方案的可靠性、可用性和效能的重要部分。AWS 提供下列監控工具來監看受管整合、在發生錯誤時回報，以及適時採取自動動作：

- AWS CloudTrail 會擷取您 AWS 帳戶或代表您的帳戶發出的 API 呼叫和相關事件，並將日誌檔案交付至您指定的 Amazon S3 儲存貯體。您可以識別呼叫的使用者和帳戶 AWS、進行呼叫的來源 IP 地址，以及呼叫的時間。如需詳細資訊，請參閱「AWS CloudTrail 使用者指南」。

使用 記錄受管整合 API 呼叫 AWS CloudTrail

受管整合與 整合AWS CloudTrail，此服務提供由使用者、角色或 所採取動作的記錄 AWS 服務。CloudTrail 會將受管整合的所有 API 呼叫擷取為事件。擷取的呼叫包括從受管整合主控台的呼叫，以及對受管整合 API 操作的程式碼呼叫。您可以使用 CloudTrail 所收集的資訊，判斷對受管整合提出的請求、提出請求的 IP 地址、提出請求的時間，以及其他詳細資訊。

每一筆事件或日誌專案都會包含產生請求者的資訊。身分資訊可協助您判斷下列事項：

- 該請求是使用根使用者還是使用者憑證提出。
- 請求是否代表 IAM Identity Center 使用者提出。
- 提出該請求時，是否使用了特定角色或聯合身分使用者的暫時安全憑證。
- 該請求是否由另一項 AWS 服務服務提出。

當您建立帳戶 AWS 帳戶 時CloudTrail 會在 中處於作用中狀態，而且您會自動存取 CloudTrail 事件歷史記錄。CloudTrail 事件歷史記錄為 AWS 區域中過去 90 天記錄的管理事件，提供可檢視、可搜尋、可下載且不可變的記錄。如需詳細資訊，請參閱「AWS CloudTrail 使用者指南」中的使用 CloudTrail 事件歷史記錄。檢視事件歷史記錄不會產生 CloudTrail 費用。

如需 AWS 帳戶 過去 90 天內持續記錄的事件，請建立線索或 CloudTrail Lake 事件資料存放區。

CloudTrail 追蹤

線索能讓 CloudTrail 將日誌檔案交付至 Amazon S3 儲存貯體。使用 建立的所有線索 AWS 管理主控台 都是多區域。您可以使用 AWS CLI建立單一或多區域追蹤。建議您建立多區域追蹤，因為您擷取 AWS 區域 帳戶中所有 的活動。如果您建立單一區域追蹤，您只能檢視追蹤 AWS 區域中記錄的事件。如需追蹤的詳細資訊，請參閱《AWS CloudTrail 使用者指南》中的為您的 AWS 帳戶建立追蹤和為組織建立追蹤。

您可以透過建立追蹤，免費將持續管理事件的一個複本從 CloudTrail 傳遞至您的 Amazon S3 儲存貯體，但這樣做會產生 Amazon S3 儲存費用。如需 CloudTrail 定價的詳細資訊，請參閱 AWS CloudTrail 定價。如需 Amazon S3 定價的相關資訊，請參閱 Amazon S3 定價。

CloudTrail Lake 事件資料存放區

CloudTrail Lake 讓您能夠對事件執行 SQL 型查詢。CloudTrail Lake 會將分列式 JSON 格式的現有事件轉換為 Apache ORC 格式。ORC 是一種單欄式儲存格式，針對快速擷取資料進行了最佳化。系統會將事件彙總到事件資料存放區中，事件資料存放區是事件的不可變集合，其依據為您透過套用進階事件選取器選取的條件。套用於事件資料存放區的選取器控制哪些事件持續存在並可供您查詢。如需 CloudTrail Lake 的詳細資訊，請參閱AWS CloudTrail 《 使用者指南》中的使用 AWS CloudTrail Lake。

CloudTrail Lake 事件資料存放區和查詢會產生費用。建立事件資料存放區時，您可以選擇要用於事件資料存放區的定價選項。此定價選項將決定擷取和儲存事件的成本，以及事件資料存放區的預設和最長保留期。如需 CloudTrail 定價的詳細資訊，請參閱 AWS CloudTrail 定價。

CloudTrail 中的管理事件

管理事件提供有關在 資源上執行的管理操作的資訊 AWS 帳戶。這些也稱為控制平面操作。根據預設，CloudTrail 記錄管理事件。

受管整合會將下列受管整合控制平面操作記錄到 CloudTrail 做為管理事件。

- `CreateCloudConnector`
- `UpdateCloudConnector`
- `GetCloudConnector`
- `DeleteCloudConnector`
- `ListCloudConnectors`
- `CreateConnectorDestination`
- `UpdateConnectorDestination`
- `GetConnectorDestination`
- `DeleteConnectorDestination`
- `ListConnectorDestinations`
- `CreateAccountAssociation`
- `UpdateAccountAssociation`

- GetAccountAssociation
- DeleteAccountAssociation
- ListAccountAssociations
- StartAccountAssociationRefresh
- ListManagedThingAccountAssociations
- RegisterAccountAssociation
- DeregisterAccountAssociation
- SendConnectorEvent
- ListDeviceDiscoveries
- ListDiscoveredDevices

事件範例

一個事件代表任何來源提出的單一請求，並包含請求 API 操作的相關資訊、操作的日期和時間、請求參數等。CloudTrail 日誌檔案不是公有 API 呼叫的已排序堆疊追蹤，因此事件不會以任何特定順序顯示。

下列範例顯示示範成功 CreateCloudConnector API 操作的 CloudTrail 事件。

使用 **CreateCloudConnector** API 操作成功執行 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"
```

```
}
```

下列範例顯示示範成功 ListDiscoveredDevices API 操作的 CloudTrail 事件。

使用 **ListDiscoveredDevices** API 操作成功執行 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"
}
```

如需有關 CloudTrail 記錄內容的資訊，請參閱《AWS CloudTrail 使用者指南》中的 CloudTrail record contents。

受管整合開發人員指南的文件歷史記錄

下表說明受管整合的文件版本。

變更	描述	日期
一般可用性版本	受管整合開發人員指南的一般可用性版本	2025 年 6 月 25 日
初始預覽版本	受管整合的初始預覽版本開發人員指南	2025 年 3 月 3 日

本文為英文版的機器翻譯版本，如內容有任何歧義或不一致之處，概以英文版為準。