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

的受管整合 AWS IoT Device Management 處於預覽版本中，可能會有所變更。如需存取，請從 受管整合主控台聯絡我們。

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。

什麼是 受管整合 AWS IoT Device Management？

透過 的受管整合功能 AWS IoT Device Management，開發人員可以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。他們可以使用單一使用者介面來控制、管理和操作各種裝置。

主題

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
- 的 受管整合入門 AWS IoT Device Management

受管整合概觀

下圖提供受管整合功能的高階概觀：

> **Note**
>
> 的受管整合目前 AWS IoT Device Management 不支援標記。這表示您無法在組織的標記政策中包含此功能的資源。如需詳細資訊，請參閱AWS 白皮書中的標記使用案例。

受管整合客戶是誰？

受管整合的客戶將使用 功能來自動化裝置設定程序，並在許多裝置之間提供互通性支援，無論裝置廠商或連線通訊協定為何。這些解決方案供應商為裝置提供整合功能，並與硬體製造商合作，以擴展其產品範圍。客戶將可以使用 定義的資料模型與裝置互動 AWS。

如需受管整合中的不同角色，請參閱下表：

角色	責任
製造商	• 製造裝置。 • 使用受管整合註冊裝置設定檔。
最終使用者	• 在其家中管理連線至受管整合的裝置。
客戶	• 建置個別的解決方案來設定和控制與受管整合通訊的特定裝置。 • 為自己的客戶和最終使用者提供服務。

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`managedThing`相比，了解受管整合的重要概念是 。

- 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。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 *`managedThing`*：透過 `managedThing`，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。

裝置類型

受管整合可管理許多類型的裝置。這些類型的裝置屬於以下三個類別之一：

- 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所建置和管理，其中包含用於直接連線的受管整合裝置 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裝置：當最終使用者第一次開啟雲端裝置的電源時，必須佈建其個別的第三方雲端供應商，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。它可以同時做為上述裝置類型和中樞。

Cloud-to-cloud術語

與受管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針對最終使用者和受管整合控制這些裝置，並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

受管整合使用兩種資料模型來組織資料，並在您的裝置之間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。
- 節點：裝置由多個節點組成（取自事項資料模型 的 AWS實作）。每個節點都會處理與其他節點的通訊。節點可唯一定址，以促進通訊。
- 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、錨燈開啟、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，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 Management Console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。

的 受管整合入門 AWS IoT Device Management

下列各節概述開始使用 受管整合所需的步驟。如需直接連線裝置、中樞連線裝置和cloud-to-cloud裝置的詳細資訊，請參閱 裝置類型。

主題

直接連線的裝置加入

下列步驟概述將直接連線裝置加入受管整合的工作流程。

主題

設定加密金鑰 （選用）

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。我們支援保護裝置資料的方法之一是利用安全加密金鑰進行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 。

呼叫 PutDefaultEncryptionConfiguration API 會授予您更新要使用之加密金鑰選項的存取權。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 DescribeDefaultEncryptionConfiguration 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。

如需使用受管整合end-to-end加密的詳細資訊，請參閱 受管整合的靜態資料加密。

如需 AWS KMS 服務的詳細資訊，請參閱 AWS Key Management Service

此步驟中使用的 APIs：

- PutDefaultEncryptionConfiguration
- DescribeDefaultEncryptionConfiguration

註冊自訂端點（強制性）

裝置與受管整合之間的雙向通訊有助於下列項目：

- 裝置命令的提示路由。
- 您的實體裝置和受管整合會對齊受管物件數位表示狀態。
- 安全傳輸您的裝置資料。

若要連線至受管整合，裝置需要專用端點來路由流量。呼叫 RegisterCustomEndpoint API 來建立此端點，以及設定伺服器信任的管理方式。自訂端點會存放在連線至受管整合的本機中樞或 Wi-Fi 裝置的裝置 SDK 中。

> **Important**
>
> 如果您收到 RegisterCustomEndpoint 失敗錯誤，請在 Service Quotas 主控台中請求從 0 增加到 1。https：//https://console.aws.amazon.com/servicequotas/

> **Note**
>
> 您可以略過雲端連線裝置的此步驟。

此步驟中使用的 APIs：

- `RegisterCustomEndpoint`

裝置佈建（強制性）

裝置佈建會在裝置或裝置機群與受管整合之間建立連結，以供未來的雙向通訊使用。呼叫 `CreateProvisioningProfile` API 以建立佈建範本和宣告憑證。

- 佈建範本是一種文件，可定義在佈建程序期間套用至裝置的一組資源和政策。它指定裝置在第一次連接到受管整合時應如何註冊和設定。此程序會自動化裝置設定程序，以確保每個裝置都安全且一致 AWS IoT 地與適當的許可、政策和組態整合到 中。
- 宣告憑證是在機群佈建期間使用的臨時憑證，而且只有在交付給最終使用者之前，唯一裝置憑證未在製造期間預先安裝在裝置上時。

下列清單概述裝置佈建工作流程和每個工作流程之間的差異：

- 單一裝置佈建
  - 佈建具有受管整合的單一裝置。
  - 工作流程
    - `CreateManagedThing`：根據佈建範本，使用受管整合建立新的受管物件（裝置）。
  - 如需終端裝置軟體開發套件 (SDK) 的詳細資訊，請參閱什麼是結束裝置 SDK？。
  - 如需單一裝置佈建的詳細資訊，請參閱單一物件佈建。
- 依宣告佈建機群
  - 由授權使用者佈建
    - 您需要建立組織裝置佈建工作流程專屬的 IAM 角色和政策（以便最終使用者可以將裝置佈建至受管整合）。如需為此工作流程建立 IAM 角色和政策的詳細資訊，請參閱為安裝裝置的使用者建立 IAM 政策和角色。
    - 工作流程
      - `CreateKeysAndCertificate`：為裝置建立臨時宣告憑證和金鑰。
      - `CreatePolicy`：建立定義裝置許可的政策。
      - `AttachPolicy`：將政策連接至臨時宣告憑證。
      - `CreateProvisioningTemplate`：建立佈建範本，以定義裝置的佈建方式。

    - RegisterThing：裝置佈建程序的一部分，根據佈建範本，在 IoT 登錄檔中註冊新物件（裝置）。
    - 此外，當裝置第一次使用佈建宣告連線到 AWS IoT Core 時，它會利用 MQTT 或 HTTPS 通訊協定進行安全通訊。在此過程中，AWS IoT Core 的內部機制會驗證宣告、套用佈建範本，並完成佈建程序。
- 使用宣告憑證佈建
  - 您需要建立連接到每個裝置宣告憑證的宣告憑證佈建政策，以便與受管整合進行初始聯絡，然後將其替換為裝置特定的憑證。若要使用宣告憑證工作流程完成佈建，您必須將硬體序號傳送至 MQTT 預留主題。
  - 工作流程
    - CreateKeysAndCertificate：為裝置建立臨時宣告憑證和金鑰。
    - CreatePolicy：建立定義裝置許可的政策。
    - AttachPolicy：將政策連接至臨時宣告憑證。
    - CreateProvisioningTemplate：建立佈建範本，以定義裝置的佈建方式。
    - RegisterThing：裝置佈建程序的一部分，根據佈建範本，在 IoT 登錄檔中註冊新物件（裝置）。
    - 此外，當裝置第一次使用佈建宣告連線到 AWS IoT Core 時，它會利用 MQTT 或 HTTPS 通訊協定進行安全通訊。在此過程中，AWS IoT Core 的內部機制會驗證宣告、套用佈建範本，並完成佈建程序。
  - 如需依宣告憑證佈建的詳細資訊，請參閱依宣告佈建。

如需佈建範本的詳細資訊，請參閱佈建範本。

此步驟中使用的 APIs：

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

- CreateProvisioningTemplate

受管整合終端裝置 SDK （強制性）

在初始製造期間，在裝置的韌體中新增結束裝置 SDK。如適用，請將加密金鑰、自訂端點地址、設定憑證、宣告憑證，以及您剛建立的佈建範本新增至終端裝置 SDK 以進行受管整合，以支援最終使用者的裝置佈建。

如需結束裝置 SDK 的詳細資訊，請參閱 什麼是結束裝置 SDK？

裝置與登入資料儲存庫的預先關聯 （選用）

在履行過程中，會掃描裝置的條碼，將裝置的資訊上傳至受管整合。這會自動呼叫 CreateManagedThing API 並建立受管物件，這是存放在受管整合中的實體裝置的數位表示法。此外，CreateManagedThingAPI 會自動傳回 deviceID以在裝置佈建期間使用。

如果可用，則可以在CreateManagedThing請求訊息中包含擁有者的資訊。包含此擁有者資訊允許擷取設定登入資料和預先定義的裝置功能，以包含在managedThing存放在受管整合中的 中。這支援縮短使用受管整合佈建裝置或裝置機群的時間。

如果無法使用擁有者的資訊，則 CreateManagedThing API 呼叫中的 owner 參數會保留空白，並在裝置開啟時於裝置加入期間更新。

在此步驟期間使用的 APIs：

- CreateManagedThing

裝置探索和加入 （選用）

最終使用者開啟裝置或視需要將其設定為配對模式後，即可使用下列探索和加入工作流程：

簡單設定 (SS)

最終使用者開啟 IoT 裝置的電源，並使用 受管整合應用程式掃描其 QR 碼。應用程式會在受管整合雲端上註冊裝置，並將其連線至 IoT Hub。

使用者引導式設定 (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 Hub 上的按鈕、使用 受管整合應用程式，或同時按下集線器和裝置上的按鈕。如果簡易設定失敗，請使用此方法。

- 智慧裝置：它會自動開始連線至本機 Hub 裝置，Hub 裝置會在其中共用本機網路憑證和 SSID，並將 Wi-Fi 裝置與本機 Hub 裝置建立關聯。接下來，智慧型裝置會嘗試連線至您先前使用伺服器名稱指示 (SNI) 擴充功能建立的自訂端點。
- 沒有智慧功能的 Wi-Fi 裝置：除了將 Wi-Fi 裝置與其建立關聯的本機 Hub 裝置之外，Wi-Fi 裝置將自動呼叫 `StartDeviceDiscovery` API，以開始 Wi-Fi 裝置與本機 Hub 裝置之間的配對程序。接著，Wi-Fi 裝置會嘗試連線至您先前使用伺服器名稱指示 (SNI) 延伸模組建立的自訂端點。
- 沒有行動應用程式設定的 Wi-Fi 裝置：在本機 Hub 裝置上，讓它開始接收 Wi-Fi 等所有無線電通訊協定。Wi-Fi 裝置會自動連線至本機 Hub 裝置，然後本機 Hub 裝置會將 Wi-Fi 裝置與其建立關聯。接著，Wi-Fi 裝置會嘗試連線至您先前使用伺服器名稱指示 (SNI) 延伸模組建立的自訂端點。

此步驟中使用的 API：

- `StartDeviceDiscovery`

裝置命令和控制

裝置加入完成後，您就可以開始傳送和接收裝置命令來管理裝置。下列清單說明一些管理裝置的案例：

- 傳送裝置命令：從您的裝置傳送和接收命令，以管理裝置的生命週期。
  - 使用的 APIs 取樣：`SendManagedThingCommand`。
- 更新裝置狀態：根據傳送的裝置功能和裝置命令，更新裝置的狀態。
  - 使用的 APIs 取樣：`GetManagedThingState`、`UpdateManagedThing`、`ListManagedThingState`和 `DeleteManagedThing`。
- 接收裝置事件：從傳送至受管整合的第三方雲端供應商接收有關 C2C 裝置的事件。
  - 使用的 APIs 取樣：`SendDeviceEvent`、`CreateLogLevel`、`CreateNotificationConfiguration`。

此步驟中使用的 APIs：

- `SendManagedThingCommand`
- `GetManagedThingState`
- `ListManagedThingState`
- `UpdateManagedThing`
- `DeleteManagedThing`

- `SendDeviceEvent`
- `CreateLogLevel`
- `CreateNotificationConfiguration`

API 索引

如需受管整合 APIs的詳細資訊，請參閱 受管整合 API 參考指南。

如需 AWS IoT Core APIs的詳細資訊，請參閱 AWS IoT Core API 參考指南。

中樞連線裝置加入

主題

行動應用程式協調 （選用）

為最終使用者提供行動應用程式，有助於直接從行動裝置管理裝置的一致使用者體驗。利用行動應用程式中的直覺式使用者介面，最終使用者可以呼叫各種受管整合 APIs 來控制、管理和操作其裝置。行動應用程式可以透過路由裝置中繼資料，例如擁有者 ID、支援的裝置通訊協定和裝置功能，來協助裝置探索。

此外，行動應用程式可協助將 受管整合 AWS 帳戶 中的 連結至第三方雲端，其中包含第三方雲端裝置的最終使用者帳戶和裝置資料。帳戶連結可確保最終使用者行動應用程式、受管整合 AWS 帳戶 中的和第三方雲端之間的裝置資料無縫路由。

設定加密金鑰 （選用）

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方法之一是利用安全加密金鑰進行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 。

呼叫 `PutDefaultEncryptionConfiguration` API 會授予您更新要使用之加密金鑰選項的存取權。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 `DescribeDefaultEncryptionConfiguration` 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。

如需使用 受管整合end-to-end加密的詳細資訊，請參閱 受管整合的靜態資料加密。

如需 AWS KMS 服務的詳細資訊，請參閱 AWS Key Management Service

此步驟中使用的 APIs：

- `PutDefaultEncryptionConfiguration`
- `DescribeDefaultEncryptionConfiguration`

註冊自訂端點 （強制性）

裝置與受管整合之間的雙向通訊可確保裝置命令的快速路由、實體裝置和受管整合的受管物件數位呈現狀態一致，以及裝置資料的安全傳輸。若要連線至受管整合，裝置需要專用端點來路由流量。除了設定伺服器信任的管理方式之外，呼叫 `RegisterCustomEndpoint` API 也會建立此端點。客戶端點將存放在連線至受管整合的本機中樞或 Wi-Fi 裝置的裝置 SDK 中。

> **Note**
>
> 您可以略過雲端連線裝置的此步驟。

此步驟中使用的 APIs：

- RegisterCustomEndpoint

裝置佈建（強制性）

裝置佈建會在您的裝置或裝置機群與受管整合之間建立連結，以供未來的雙向通訊使用。呼叫 CreateProvisioningProfile API 以建立佈建範本和宣告憑證。佈建範本是一種文件，可定義在佈建程序期間套用至裝置的一組資源和政策。它指定裝置在第一次連接到受管整合時應如何註冊和設定。此程序會自動化裝置設定程序，以確保每個裝置都安全且一致 AWS IoT 地與適當的許可、政策和組態整合到 中。宣告憑證是在機群佈建期間使用的臨時憑證，而且只有在交付給最終使用者之前，唯一裝置憑證未在製造期間預先安裝在裝置上時。

下列清單概述裝置佈建工作流程和每個工作流程之間的差異：

- 單一裝置佈建
  - 佈建具有受管整合的單一裝置。
  - 工作流程
    - CreateManagedThing：根據佈建範本，使用受管整合建立新的受管物件（裝置）。
  - 如需終端裝置軟體開發套件 (SDK) 的詳細資訊，請參閱什麼是結束裝置 SDK？。
  - 如需單一裝置佈建的詳細資訊，請參閱單一物件佈建。
- 依宣告佈建機群
  - 由授權使用者佈建
    - 您需要建立組織裝置佈建工作流程專屬的 IAM 角色和政策（以便最終使用者可以將裝置佈建至受管整合）。如需為此工作流程建立 IAM 角色和政策的詳細資訊，請參閱為安裝裝置的使用者建立 IAM 政策和角色。
    - 工作流程
      - CreateKeysAndCertificate：為裝置建立臨時宣告憑證和金鑰。
      - CreatePolicy：建立定義裝置許可的政策。
      - AttachPolicy：將政策連接至臨時宣告憑證。
      - CreateProvisioningTemplate：建立佈建範本，以定義裝置的佈建方式。
      - RegisterThing：裝置佈建程序的一部分，根據佈建範本，在 IoT 登錄檔中註冊新物件（裝置）。
      - 此外，當裝置第一次使用佈建宣告連線到 AWS IoT Core 時，它會利用 MQTT 或 HTTPS 通訊協定進行安全通訊。在此過程中，AWS IoT Core 的內部機制會驗證宣告、套用佈建範本，並完成佈建程序。

    - 如需授權使用者佈建的詳細資訊，請參閱由信任的使用者佈建。
- 使用宣告憑證佈建
    - 您需要建立連接到每個裝置宣告憑證的宣告憑證佈建政策，以便與受管整合進行初始聯絡，然後將其替換為裝置特定的憑證。若要使用宣告憑證工作流程完成佈建，您必須將硬體序號傳送至 MQTT 預留主題。
    - 工作流程
        - `CreateKeysAndCertificate`：為裝置建立臨時宣告憑證和金鑰。
        - `CreatePolicy`：建立定義裝置許可的政策。
        - `AttachPolicy`：將政策連接至臨時宣告憑證。
        - `CreateProvisioningTemplate`：建立佈建範本，以定義裝置的佈建方式。
        - `RegisterThing`：裝置佈建程序的一部分，根據佈建範本，在 IoT 登錄檔中註冊新物件（裝置）。
        - 此外，當裝置 AWS IoT Core 第一次使用佈建宣告連線至 時，它會利用 MQTT 或 HTTPS 通訊協定進行安全通訊。在此過程中， AWS IoT Core的內部機制會驗證宣告、套用佈建範本，並完成佈建程序。
    - 如需依宣告憑證佈建的詳細資訊，請參閱依宣告佈建。

如需佈建範本的詳細資訊，請參閱佈建範本。

此步驟中使用的 APIs：

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

受管整合中樞 SDK （強制性）

在初始製造期間，在裝置的韌體中新增受管整合 Hub SDK。如適用，請將加密金鑰、自訂端點地址、設定憑證、宣告憑證，以及您剛建立的佈建範本新增至 Hub SDK，以支援最終使用者的裝置佈建。

如需 Hub SDK 的詳細資訊，請參閱 Hub SDK 架構

裝置與登入資料儲存庫的預先關聯 （選用）

在履行過程中，可以透過掃描裝置的條碼，將裝置預先與最終使用者建立關聯。這會自動呼叫 `CreateManagedThing` API 並建立受管物件，這是存放在受管整合中的實體裝置的數位表示法。此外，`CreateManagedThing`API 會自動傳回 `deviceID`以在裝置佈建期間使用。

如果可用，可以在`CreateManagedThing`請求訊息中包含擁有者的資訊。包含此擁有者資訊允許擷取設定登入資料和預先定義的裝置功能，以包含在`managedThing`存放在受管整合中的 中。這支援縮短使用受管整合佈建裝置或裝置機群的時間。

如果無法使用擁有者的資訊，則 `CreateManagedThing` API 呼叫中的 `owner` 參數會保留空白，並在裝置開啟時於裝置加入期間更新。

在此步驟期間使用的 APIs：

- `CreateManagedThing`

裝置探索和加入 （強制性）

最終使用者開啟裝置或視需要將其設定為配對模式後，會根據裝置類型發生下列情況：

簡單設定 (SS)

最終使用者開啟 IoT 裝置的電源，並使用 受管整合應用程式掃描其 QR 碼。應用程式會在受管整合雲端上註冊裝置，並將其連線至 IoT Hub。

使用者引導式設定 (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 Hub 上的按鈕、使用 受管整合應用程式，或同時按下集線器和裝置上的按鈕。如果簡易設定失敗，請使用此方法。

裝置命令和控制

一旦裝置加入完成，您就可以開始傳送和接收裝置命令來管理裝置。下列清單說明一些管理裝置的案例：

- 傳送裝置命令：從您的裝置傳送和接收命令，以管理裝置的生命週期。
  - 使用的 APIs 取樣：SendManagedThingCommand。
- 更新裝置狀態：根據傳送的裝置生命週期和裝置命令，更新裝置的狀態。
  - 使用的 APIs 取樣：GetManagedThingState、UpdateManagedThing、ListManagedThingState和 DeleteManagedThing。
- 接收裝置事件：從傳送至受管整合的第三方雲端供應商接收有關 C2C 裝置的事件。
  - 使用的 APIs 取樣：SendDeviceEvent、CreateLogLevel、CreateNotificationConfiguration。

此步驟中使用的 APIs：

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

API 索引

如需受管整合 APIs的詳細資訊，請參閱 受管整合 API 參考指南。

如需 AWS IoT Core APIs的詳細資訊，請參閱 AWS IoT Core API 參考指南。

Cloud-to-cloud裝置加入

下列步驟概述從第三方雲端供應商將雲端裝置加入受管整合的工作流程。

> **Note**
>
> 當您設定cloud-to-cloud裝置的通知時，不需要使用 `RegisterCustomEndpoint` API。只有在您設定直接連線裝置或中樞連線裝置時，才需要此 API。

主題

行動應用程式協調 （強制性）

為最終使用者提供行動應用程式，有助於直接從行動裝置管理裝置的一致使用者體驗。利用行動應用程式中的直覺式使用者介面，最終使用者可以呼叫各種受管整合 APIs 來控制、管理和操作其裝置。行動應用程式可以透過路由裝置中繼資料，例如擁有者 ID、支援的裝置通訊協定和裝置功能，來協助裝置探索。

此外，行動應用程式可協助將 受管整合 AWS 帳戶 中的 連結至第三方雲端，其中包含第三方雲端裝置的最終使用者帳戶和裝置資料。帳戶連結可確保最終使用者行動應用程式、受管整合 AWS 帳戶 中的和第三方雲端之間的裝置資料無縫路由。

設定加密金鑰 （選用）

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進行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)

呼叫 PutDefaultEncryptionConfiguration API 可讓您存取 ，以更新您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 DescribeDefaultEncryptionConfiguration 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。

此步驟中使用的 APIs：

- PutDefaultEncryptionConfiguration
- DescribeDefaultEncryptionConfiguration

帳戶連結 （必要）

帳戶連結是使用最終使用者的登入資料將您的雲端環境連結至第三方供應商雲端的程序。在雲端環境與最終使用者的行動應用程式之間路由裝置命令和其他裝置相關資料時，需要此連結。

若要啟動帳戶連結，最終使用者將在支援雲端連線裝置的行動應用程式中傳送 StartAccountLinking API 命令。第三方雲端將傳回行動應用程式的 URL，並提示最終使用者輸入其第三方雲端登入憑證，並授權雲端環境與最終使用者行動應用程式之間的帳戶連結請求。

此步驟中使用的 APIs：

- StartAccountLinking

裝置探索 （強制性）

帳戶連結完成後，會自動呼叫 StartDeviceDiscovery API。第三方雲端會將與最終使用者的第三方帳戶相關聯的裝置清單發佈至 MQTT 主題 DevicesToApprove。最終使用者將在行動應用程式中核准選取的裝置，以使用受管整合進行裝置註冊。然後，系統會使用 CreateManagedThing API 命令，為每個已註冊的裝置自動產生受管整合 受管物件。受管整合受管物件是存放在受管整合中的實體裝置的數位表示。

此步驟中使用的 APIs：

- StartDeviceDiscovery
- CreateManagedThing

裝置命令和控制

一旦裝置加入完成，您就可以開始傳送和接收裝置命令來管理裝置。下列清單說明一些管理裝置的案例：

- 傳送裝置命令：從您的裝置傳送和接收命令，以管理裝置的生命週期。
  - 使用的 APIs 取樣：SendManagedThingCommand。
- 更新裝置狀態：根據傳送的裝置生命週期和裝置命令，更新裝置的狀態。
  - 使用的 APIs 取樣：GetManagedThingState、UpdateManagedThing、ListManagedThingState和 DeleteManagedThing。
- 接收裝置事件：從傳送至受管整合的第三方雲端供應商接收有關 C2C 裝置的事件。
  - 使用的 APIs 取樣：SendDeviceEvent、CreateLogLevel、CreateNotificationConfiguration。

此步驟中使用的 APIs：

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

API 索引

如需受管整合 APIs的詳細資訊，請參閱 受管整合 API 參考指南。

如需 AWS IoT Core APIs的詳細資訊，請參閱 AWS IoT Core API 參考指南。

裝置佈建

將裝置佈建至受管整合是初始加入程序中的重要步驟，可在使用第三方裝置時，促進實體裝置、本機中樞、受管整合和第三方雲端提供者之間的雙向、近乎即時的通訊。

什麼是裝置佈建？

裝置佈建可促進無縫的裝置加入程序、監督整個裝置生命週期，並針對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，用於管理各種裝置類型，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 `deviceId`。此識別符用於裝置在整個裝置生命週期的加入和管理。它完全由受管整合進行管理，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，此識別符會在受管整合中`managedThing`建立並連接至 。`managedThing` 是受管整合中實體裝置的數位表示法，可鏡像實體裝置的所有裝置中繼資料。對於第三方裝置，除了`deviceId`存放在受管整合中的 之外，他們還可能擁有自己的獨立唯一識別符，以代表實體裝置。

提供下列入門流程，以使用受管整合佈建您的裝置：

- 簡單設定 (SS)：最終使用者開啟 IoT 裝置電源，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。
- 使用者引導式設定 (UGS)：最終使用者在裝置上開機，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，無論裝置類型或裝置通訊協定為何，都能用於加入和管理裝置。

裝置和裝置設定檔生命週期

管理裝置和裝置描述檔的生命週期可確保裝置機群安全且有效率地執行。

主題

裝置

在初始加入程序期間，managedThing會建立稱為 之實體裝置的數位表示。managedThing 提供全域唯一識別符，以識別所有區域中受管整合中的裝置。在佈建期間，裝置會與本機中樞配對，以便與受管整合或第三方裝置的第三方雲端進行即時通訊。裝置也會與擁有者建立關聯，如 managedThing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managedThing中建立 開始，並在客戶managedThing使用 DeleteManagedThing API 刪除 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的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。當製造商在受管整合中將其刪除時，裝置設定檔的生命週期就會結束。

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，使用兩個資料模型。受管整合資料模型和 AWS的事項資料模型實作。它們都具有相似性，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" },
    "name": { "$ref": "aws.name" },       // Optional human readable name
    "capabilities": Capability[]
}
```

Capability

capability 代表裝置功能。

```
{
    "$id": "string",                // Schema identifier (e.g. namespace or
 resourceType)
```

```
    "name": "string",               // Human readable name
    "version": "string",            // e.g. 1.0
    "properties": Property[],
    "actions": Action[],
    "events": Event[]
}
```

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": { "$ref": "aws.name" }, //required
        "parameters": Map<String name, JSONNode value>,
        "responseCode": HTTPResponseCode,
        "errors": {
            "code": "string",
            "message": "string"
        }
    }
    ```
- 事件：基本上是過去狀態轉換的記錄。雖然 property代表目前狀態，但事件是過去的日誌，並包含單調增加的計數器、時間戳記和優先順序。它們可以擷取狀態轉換，以及 無法立即實現的資料建模property。
  - ```
    {
        "name": { "$ref": "aws.name" },        //required
        "parameters": Map<String name, JSONNode value>
    }
    ```
    ```

AWS 實作事項資料模型

AWS的事項資料模型實作會管理受管整合與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",
        ...
    }
    ```
    ```

管理 IoT 裝置命令和事件

裝置命令提供遠端管理實體裝置的功能，除了執行重要的安全性、軟體和硬體更新之外，還可確保完全控制裝置。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，以更新最終使用者。

主題

裝置命令

命令請求是客戶傳送至裝置的命令。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。

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，例如指示燈開啟的時間，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。

什麼是裝置狀態？

裝置狀態由 資源集合描述。資源是指結構描述描述描述的一組功能。每個資源狀態變更都有相關聯的版本編號。隨著從裝置新增或移除功能，與裝置相關聯的資源可能會隨著時間而變更。

裝置事件通知

最終使用者可以訂閱他們為更新特定裝置事件而建立的特定客戶受管目的地。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，則會收到通知。

設定受管整合通知

受管整合通知可管理客戶的所有通知，促進即時通訊，以便在其裝置上提供更新和洞見。無論是通知客戶裝置事件、裝置生命週期或裝置狀態，受管整合通知在提升整體客戶體驗方面都扮演重要角色。透過提供可行的資訊，客戶可以做出明智的決策並最佳化資源使用率。

主題

設定受管整合通知

若要設定受管整合通知，請完成下列四個步驟。

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。

目前，僅支援 Amazon Kinesis 資料串流做為受管整合通知的客戶受管目的地選項。

建立 Amazon Kinesis 串流存取角色

建立具有許可的 AWS Identity and Access Management 存取角色，以存取您剛建立的 Kinesis 串流

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。

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 來建立客戶受管目的地，其中將路由受管整合通知。針對 `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": "Random string"
    "RoleArn": "Your Role arn"
}
```

呼叫 **CreateNotificationConfiguration** API

最後，您將建立通知組態，透過將通知路由到 Amazon Kinesis 資料串流所代表的客戶受管目的地，來通知您所選事件類型。呼叫 CreateNotificationConfiguration API 以建立通知組態。在 destinationName 參數中，使用與使用 CreateDestination API 建立客戶受管目的地時最初建立的相同目的地名稱。

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

下列列出可透過 受管整合通知監控的事件類型：

- 說明連接器的關聯狀態。
- DEVICE_COMMAND
  - SendManagedThing API 命令的狀態。此有效值成功或失敗。

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

- DEVICE_COMMAND_REQUEST

- Web 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 https://webrtc.org/。

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

- DEVICE_EVENT
  - 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":[
```

```
    "arn:aws:iotmanagedintegrations:ca-central-1:123456789012:managed-
thing/1b15b39992f9460ba82c6c04595d1f4f"
  ],
  "payload":{
    "endpoints":[{
      "endpointId":"1",
      "capabilities":[{
        "id":"aws.DoorLock",
        "name":"Door Lock",
        "version":"1.0",
        "properties":[{
          "name":"ActuatorEnabled",
          "value":"true"
        }]
      }]
    }]
  }
}
```

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

```
  }
}
```

- DEVICE_OTA
  - 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:us-west-2:123456789012:managed-
thing/61889008880012345678"
  ],
  "payload": {
    "addedStates": {
      "endpoints": [{
        "endpointId": "nonEndpointId",
        "capabilities": [{
          "id": "aws.OnOff",
          "name": "On/Off",
          "version": "1.0",
          "properties": [{
            "name": "OnOff",
            "value": {
              "propertyValue": "\"onoff\"",
              "lastChangedAt": "2024-06-11T01:38:09.000414Z"
            }
          }
        ]}
      ]}
    ]}
  }
}
```

受管整合中樞 SDK

本章介紹使用 受管整合中樞 SDK 來加入和控制 IoT 中樞裝置。受管整合目前為公開預覽。若要下載最新版本的受管整合 Hub SDK，請從受管整合主控台聯絡我們。如需有關受管整合終端裝置 SDK 的資訊，請參閱 受管整合 終端裝置 SDK。

Hub SDK 架構

裝置加入簡介

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。

通訊協定外掛程式介面

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。

處理間通訊 (IPC) 用戶端與 CDMB

接收裝置功能報告，並將其從通訊協定特定的 CDMB 外掛程式轉送至受管整合。

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)

最終使用者會開啟 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。

當最終使用者收到裝置並開啟電源時，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。

裝置控制簡介

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

即使沒有網際網路連線，裝置也會在同一個 IoT 中樞上與其他裝置通訊。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。

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。它包含核心佈建器和協定特定的佈建器外掛程式，用於協定特定的加入任務。

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。

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。

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，設定您的中樞裝置與受管整合通訊。本節說明中樞加入子系統元件如何一起運作、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

將基本中樞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

```
curl 'https://api.iotmanagedintegrations.AWS-REGION.api.aws/custom-endpoint' \
  -H 'Content-Encoding: amz-1.0' \
  -H 'Content-Type: application/json; charset=UTF-8' \
```

```
  -H 'X-Amz-Target: iotmanagedintegrations.RegisterCustomEndpoint' \
  -H 'X-Amz-Security-Token: $AWS_SESSION_TOKEN' \
  --user "$AWS_ACCESS_KEY_ID:$AWS_SECRET_ACCESS_KEY" \
  --aws-sigv4 "aws:amz:AWS-REGION:iotmanagedintegrations" \
  -X POST --data '{}'
```

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--> 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到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 以產生下列項目：
  - 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

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

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。每個中樞都需要自己的受管物件，以及唯一的身分驗證資料。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，請指定下列參數：

- Role：將此值設定為 CONTROLLER。
- AuthenticationMaterial：包含下列欄位。
  - SN：此裝置的唯一序號
  - UPC：此裝置的通用產品代碼

- `Owner`：此受管物件的擁有者識別符。

> **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",
 "Owner": "ThingOwner1",
 "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 `UNCLAIMED`才能繼續。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

  將兩個檔案都放在 `/data/aws/iotmi/certs`目錄中。

  > **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碼、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（此產品的所有裝置皆相同）

```
{
    "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。

金鑰	值	由客戶新增？	備註
iot_claim_cert_path	您指定的檔案路徑 或 SECURE_STORAGE 。例如 /data/aws/iotmi/certs/claim_cert.pem	是	指定您要使用 或 的檔案路徑SECURE_STORAGE 。
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	您的裝置會使用此 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 。

金鑰	值	由客戶新增？	備註
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。
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。
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。

用於機群佈建

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

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**
>
> 裝置狀態管理員資料庫
> 檢查您的 `artifacts`目錄中是否存在`prov.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 releasereadme.md.tgz 檔案`hubSystemdSetup`目錄中的 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

檔案結構旨在促進組織和管理其各種元件，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：

```
IoTSmartHomeDevice-SDK-v0.4.0-linux-aarch64-20240106.tgz
###package/
    ###greengrass/
        ###artifacts/
        ###recipes/
    ###hubSystemdSetup/
```

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

初始設定

解壓縮 SDK 套件

```
tar -xzf IoTSmartHomeDevice-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 會進入中樞並執行安裝指令碼：

```
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。

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

- 使用安全儲存

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"
  }
}
```

處理程序間通訊 (IPC) APIs

受管整合中樞上的外部元件可以使用其代理程式元件和程序間通訊 (IPC) 與受管整合終端裝置 SDK 進行通訊。中樞上的外部元件範例是管理本機常式的協助程式（持續執行的背景程序）。在通訊期間，IPC 用戶端是發佈命令或其他請求並訂閱事件的外部元件。IPC 伺服器是受管整合終端裝置 SDK 中的代理程式元件。如需詳細資訊，請參閱設定 IPC 用戶端。

若要建置 IPC 用戶端，`IotmiLocalControllerClient`會提供 IPC 用戶端程式庫。此程式庫提供用戶端 APIs用於與 Agent 中的 IPC 伺服器通訊，包括傳送命令請求、查詢裝置狀態、訂閱事件（例如裝置狀態事件），以及處理事件型互動。

主題

設定 IPC 用戶端

`IotmiLocalControllerClient` 程式庫是基本 IPC APIs的包裝函式，可簡化和簡化在應用程式中實作 IPC 的程序。下列各節說明其提供的 APIs。

> **Note**
>
> 本主題專門針對做為 IPC 用戶端的外部元件，而非 IPC 伺服器的實作。

1. 建立 IPC 用戶端

   您必須先初始化 IPC 用戶端，才能用來處理請求。您可以在 `IotmiLocalControllerClient`程式庫中使用建構函數，以訂閱者內容 `char *subscriberCtx`做為參數，並根據它建立 IPC 用戶端管理員。以下是建立 IPC 用戶端的範例：

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

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

2. 訂閱事件

您可以訂閱 IPC 用戶端至目標 IPC 伺服器的事件。當 IPC 伺服器發佈用戶端訂閱的事件時，用戶端會收到該事件。若要訂閱，請使用 `registerSubscriber`函數並提供要訂閱的事件 IDs，以及自訂回呼。

以下是 `registerSubscriber`函數的定義及其範例用量：

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

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

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

`status` 會定義 來檢查操作 （例如訂閱或傳送請求） 是否成功。如果操作成功，傳回的狀態為 `IOTMI_STATUS_OK (= 0)`。

> **Note**
>
> IPC 程式庫具有下列訂閱訂閱者和事件數量上限的服務配額：
>
> - 每個程序的訂閱者數目上限：5
>
>   在 IPC 程式庫`IOTMI_IPC_MAX_SUBSCRIBER`中定義為 。
> - 定義的事件數目上限：32
>
>   在 IPC 程式庫`IOTMI_IPC_EVENT_PUBLIC_END`中定義為 。

- 每個訂閱者都有一個 32 位元事件欄位，其中每個位元對應於定義的事件。

3. 將 IPC 用戶端連線至伺服器

   `IotmiLocalControllerClient` 程式庫中的連線函數會執行任務，例如初始化 IPC 用戶端、註冊訂閱者，以及訂閱`registerSubscriber`函數中提供的事件。您可以在 IPC 用戶端上呼叫連線函數。

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

   在您傳送請求或進行其他操作`IOTMI_STATUS_OK`之前，請確認傳回的狀態為 。

4. 傳送命令請求和裝置狀態查詢

   Agent 中的 IPC 伺服器可以處理命令請求和裝置狀態請求。

   - 命令請求

     形成命令請求承載字串，然後呼叫 `sendCommandRequest`函數來傳送。例如：

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

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

     如需命令請求格式的詳細資訊，請參閱命令請求。

     Example 回呼函數

     IPC 伺服器會先傳送訊息確認`Command received, will send command response back`給 IPC 用戶端。收到此確認後，IPC 用戶端可以預期命令回應事件。

     ```
     void iotmiIpcMgr_commandRequestCb(iotmi_statusCode_t ret_status,
     ```

```
                                        void *ret_data, void *ret_client_ctx) {
    char* data = NULL;
    char *ctx = NULL;

    if (ret_status != IOTMI_STATUS_OK)
        return;

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

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

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

- 裝置狀態請求

  與 sendCommandRequest函數類似，此sendDeviceStateQuery函數也接受承載字串、對應的回呼和用戶端內容。

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

IPC 界面定義和承載

本節著重於專門用於代理程式和外部元件之間通訊的 IPC 介面，並提供這兩個元件之間 IPC APIs 的範例實作。在下列範例中，外部元件會管理本機常式。

在 IoTManagedIntegrationsDevice-IPC程式庫中，定義了下列命令和事件，以便在 代理程式和外部元件之間進行通訊。

```
typedef enum {
    // The async cmd used to send commands from the external component to Agent
    IOTMI_IPC_SVC_SEND_REQ_FROM_RE = 32,
```

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

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

命令請求

命令請求格式

- Example 命令請求

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

```
}
```

命令回應格式

- 如果來自外部元件的命令請求有效，代理程式會將其傳送至 CDMB （通用資料模型橋接器）。包含命令執行時間和其他資訊的實際命令回應不會立即傳回外部元件，因為處理命令需要一些時間。此命令回應是來自 代理程式的即時回應 （例如確認）。回應會告知外部元件受管整合收到 命令，如果沒有有效的本機字符， 會處理或捨棄該命令。命令回應會以字串格式傳送。

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

裝置狀態請求

外部元件會將裝置狀態請求傳送至 代理程式。請求會提供managedThingId裝置的 ，然後代理程式會回覆該裝置的狀態。

裝置狀態請求格式

- 您的裝置狀態請求必須具有已查詢裝置的 managedThingId 。

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

裝置狀態回應格式

- 如果裝置狀態請求有效，代理程式會以字串格式傳回狀態。

  Example 有效請求的裝置狀態回應

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

```
    }
}
```

如果裝置狀態請求無效（例如，如果沒有有效的字符、無法處理承載，或另一個錯誤案例），則代理程式會傳回回應。回應包含錯誤代碼和錯誤訊息。

Example 無效請求的裝置狀態回應

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

命令回應

Example 命令回應格式

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

```
    }
}
```

通知事件

Example 通知事件格式

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

設定中樞控制

Hub 控制是受管整合終端裝置 SDK 的延伸，可讓它與 Hub SDK 中的 `MQTTProxy` 元件連接。透過中樞控制，您可以使用終端裝置 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。集線器控制 SDK 將以單獨的套件提供，其中包含中樞 SDK，標記為 `hub-control-x.x.x`。

主題

- 先決條件
- 終端裝置 SDK 元件

- 與終端裝置 SDK 整合
- 範例：建置中樞控制
- 支援的範例
- 支援平台

先決條件

若要設定中樞控制，首先需要下列項目：

- 已加入終端裝置 SDK 0.4.0 版或更新版本的中樞。
- 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

您將從結束裝置 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. 遵循適用於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-Functions 的 API 操作。

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

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

c. 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 ，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
```

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

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
智慧燈泡/可調光燈/RGB 燈	OnOff、LevelControl、ColorControl

Zigbee 裝置類型	支援的功能
智慧插頭	OnOff
智慧切換	OnOff
LED 長條圖	OnOff、LevelControl、ColorControl
水閥	OnOff
門鎖	DoorLock、警示、PowerSourceConfiguration、計時器
輻射器閥	節溫器、OnOff、計時器
節溫器	節溫器、FanControl、OnOff、計時器
車庫門開門器	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

Z-Wave 裝置類型	支援的功能
智慧插頭	OnOff
門鎖	DoorLock
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

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。
- 彈性和集中式記錄：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

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

RuntimeLogConfigurations 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，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。日誌會存放在 /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：

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。

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 是由 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 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。 AWS IoT Device Management 的受管整合處於公開預覽狀態。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和自動化資料轉譯。

如需詳細的技術需求，請參閱 附錄。

終端裝置 SDK 架構和元件

本節說明終端裝置 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
```

建立自訂端點

在雲端環境中使用 GetCustomEndpoint API 命令，為device-to-cloud通訊建立自訂端點。

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

回應範例

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

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

回應範例

```
{ "Arn":"arn:aws:iotmanagedintegrations: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 iotmanagedintegrations 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:iotmanagedintegrations: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

裝置製造商和服務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，且裝置可以連線至 Wi-Fi 網路。

依宣告佈建機群

最終使用者可以使用佈建者來佈建唯一憑證，並使用依宣告佈建向受管整合進行註冊。

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。

任務處理常式

受管整合任務處理常式是一種元件，用於接收現場裝置的over-the-air更新。它提供下載自訂任務文件以進行韌體更新或執行遠端操作的功能。OTA 更新可用來更新裝置韌體、修正受影響裝置中的安全問題，甚至將檔案傳送至使用受管整合註冊的裝置。

任務處理常式的運作方式

使用任務處理常式之前，需要從雲端和裝置端執行下列設定步驟。

- 在裝置端，裝置製造商會準備韌體更新方法，以進行over-the-air(OTA) 更新。
- 在雲端方面，客戶會準備自訂定義的任務文件，描述遠端操作和建立任務。

此程序需要在雲端和裝置端進行設定。裝置製造商會實作韌體更新方法，同時讓客戶準備任務文件並建立更新任務。當裝置連線時：

1. 裝置會擷取待定任務清單
2. 任務處理常式會檢查清單中是否有一或多個任務執行，然後選取一個
3. 任務處理常式會執行任務文件中指定的動作
4. 任務處理常式會監控任務執行，然後使用 SUCCESS或 更新任務狀態 FAILED

任務處理常式實作

在裝置上實作處理over-the-air(OTA) 更新的金鑰操作。您將設定任務文件的 Amazon S3 存取權、透過 API 建立 OTA 任務、在裝置上處理任務文件，以及整合 OTA 代理程式。下圖中的步驟說明結束裝置 SDK 與 功能之間的互動如何處理over-the-air (OTA) 更新請求。

任務處理常式會透過這些金鑰操作處理 OTA 更新：

作業

- 上傳並啟動更新
- 設定 Amazon S3 存取
- 處理任務文件
- 實作 OTA 代理程式

上傳並啟動更新

將自訂任務文件 (JSON 格式） 上傳至 Amazon S3 儲存貯體，並使用 CreateOtaTask API 建立 OTA 任務。包含以下參數：

- `S3Url`：任務文件的 URL 位置
- `Target`：受管物件的 ARN 陣列 （最多 100 個裝置）

範例請求

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

設定 Amazon S3 存取

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：

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

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. 呼叫待定任務的 `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
- 設定環境
- 為您的裝置產生程式碼

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。

下圖說明程式碼產生器如何建立 C 來源檔案。

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

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。

範本	產生的來源	備註
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、字串、點陣圖、清單和結構。

設定環境

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

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 或更高版本

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. 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
     ```

2. 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 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。

為您的裝置產生程式碼

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codegen/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
```

此命令使用數個元件將您的 .matter 檔案轉換為 C 程式碼：

- codegen.py 來自 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 .matter 檔案中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-Functions 的 API 操作

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，以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，還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);
```

事件報告函數會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

遠端命令是由結束裝置 SDK 與 功能之間的互動所處理。下列動作說明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 用戶端會從 JSON 格式的受管整合接收訊息。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。如果是燈泡，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 Data Model Handler

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。

整合結束裝置 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. 設定建置環境

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

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

2. 設定網路

   使用範例應用程式之前，請先初始化網路，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();
```

3. 設定佈建參數

   `example/project_name/device_config.sh` 使用下列佈建參數修改組態檔案：

   > **Note**
   >
   > 在建置應用程式之前，請確定您已正確設定這些參數。

   佈建參數

巨集參數	描述	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。
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，請參閱建立自訂端點。
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 是針對裝置使用的 Application Layer Protocol Negotiation (ALPN) TLS 延伸設定。

4. 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-Functions 的 API 操作。

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

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

5. 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：

   a. 建立裝置代理程式。裝置代理程式是管理所有端點及其叢集的中央元件。

   b. 為您要支援的每個叢集結構填入回呼函數點

   c. 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);
}
```

6. 使用任務處理常式來取得任務文件

   a. 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;
   }
   ```

   b. 呼叫 `iotmi_JobsHandler_start` 以初始化任務處理常式。

   c. 呼叫 從受管整合`iotmi_JobsHandler_getJobDocument`擷取任務文件。

   d. 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." ) );

}
```

7. 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: XXXXXXXXXXX
[2406832728][MAIN][INFO] universalProductCode: XXXXXXXXXXX
[2406832728][MAIN][INFO] rootCertificatePath: XXXXXXXXX
[2406832728][MAIN][INFO] pClaimCertificatePath: XXXXXXXX
[2406832728][MAIN][INFO] pClaimKeyPath: XXXXXXXXXXXXXXXX
[2406832728][MAIN][INFO] deviceInfo.serialNumber XXXXXXXXXXX
[2406832728][MAIN][INFO] deviceInfo.universalProductCode XXXXXXXXXXXXXX
[2406832728][PKCS11][INFO] PKCS #11 successfully initialized.
[2406832728][MAIN][INFO] ============= Start certificate provisioning
 =============
[2406832728][PKCS11][INFO] ======== Loading Root CA and claim credentials
 through PKCS#11 interface ========
[2406832728][PKCS11][INFO] Writing certificate into label "Root Cert".
[2406832728][PKCS11][INFO] Creating a 0x1 type object.
[2406832728][PKCS11][INFO] Writing certificate into label "Claim Cert".
[2406832728][PKCS11][INFO] Creating a 0x1 type object.
[2406832728][PKCS11][INFO] Creating a 0x3 type object.
[2406832728][MAIN][INFO] ======== Fleet-provisioning-by-Claim ========
[2025-01-02 01:43:11.404995144][iotmi_device_sdkLog][INFO] [2406832728]
[MQTT_AGENT][INFO]
[2025-01-02 01:43:11.405106991][iotmi_device_sdkLog][INFO] Establishing a TLS
 session to XXXXXXXXXXXXXXX.account-prefix-ats.iot.region.amazonaws.com
[2025-01-02 01:43:11.405119166][iotmi_device_sdkLog][INFO]
[2025-01-02 01:43:11.844812513][iotmi_device_sdkLog][INFO] [2406833168]
[MQTT_AGENT][INFO]
```

```
[2025-01-02 01:43:11.844842576][iotmi_device_sdkLog][INFO] TLS session
 connected
[2025-01-02 01:43:11.844852105][iotmi_device_sdkLog][INFO]
[2025-01-02 01:43:12.296421687][iotmi_device_sdkLog][INFO] [2406833620]
[MQTT_AGENT][INFO]
[2025-01-02 01:43:12.296449663][iotmi_device_sdkLog][INFO] Session present: 0.
[2025-01-02 01:43:12.296458997][iotmi_device_sdkLog][INFO]
[2025-01-02 01:43:12.296467793][iotmi_device_sdkLog][INFO] [2406833620]
[MQTT_AGENT][INFO]
[2025-01-02 01:43:12.296476275][iotmi_device_sdkLog][INFO] MQTT connect with
 clean session.
[2025-01-02 01:43:12.296484350][iotmi_device_sdkLog][INFO]
[2025-01-02 01:43:13.171056119][iotmi_device_sdkLog][INFO] [2406834494]
[FLEET_PROVISIONING][INFO]
[2025-01-02 01:43:13.171082442][iotmi_device_sdkLog][INFO] Received accepted
 response from Fleet Provisioning CreateKeysAndCertificate API.
[2025-01-02 01:43:13.171092740][iotmi_device_sdkLog][INFO]
[2025-01-02 01:43:13.171122834][iotmi_device_sdkLog][INFO] [2406834494]
[FLEET_PROVISIONING][INFO]
[2025-01-02 01:43:13.171132400][iotmi_device_sdkLog][INFO] Received privatekey
 and certificate with Id: XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
[2025-01-02 01:43:13.171141107][iotmi_device_sdkLog][INFO]
[2406834494][PKCS11][INFO] Creating a 0x3 type object.
[2406834494][PKCS11][INFO] Writing certificate into label "Device Cert".
[2406834494][PKCS11][INFO] Creating a 0x1 type object.
[2025-01-02 01:43:18.584615126][iotmi_device_sdkLog][INFO] [2406839908]
[FLEET_PROVISIONING][INFO]
[2025-01-02 01:43:18.584662031][iotmi_device_sdkLog][INFO] Received accepted
 response from Fleet Provisioning RegisterThing API.
[2025-01-02 01:43:18.584671912][iotmi_device_sdkLog][INFO]
[2025-01-02 01:43:19.100030237][iotmi_device_sdkLog][INFO] [2406840423]
[FLEET_PROVISIONING][INFO]
[2025-01-02 01:43:19.100061720][iotmi_device_sdkLog][INFO] Fleet-provisioning
 iteration 1 is successful.
[2025-01-02 01:43:19.100072401][iotmi_device_sdkLog][INFO]
[2406840423][MQTT][ERROR] MQTT Connection Disconnected Successfully
[2025-01-02 01:43:19.216938181][iotmi_device_sdkLog][INFO] [2406840540]
[MQTT_AGENT][INFO]
[2025-01-02 01:43:19.216963713][iotmi_device_sdkLog][INFO] MQTT agent thread
 leaves thread loop for iotmiDev_MQTTAgentStop.
[2025-01-02 01:43:19.216973740][iotmi_device_sdkLog][INFO]
[2406840540][MAIN][INFO] iotmiDev_MQTTAgentStop is called to break thread loop
 function.
[2406840540][MAIN][INFO] Successfully provision the device.
```

```
[2406840540][MAIN][INFO] Client ID :
 XXXXXXXXXXXXXXXXXXX_XXXXXXXXXXXXXXXXXXXXXX
[2406840540][MAIN][INFO] Managed thing ID : XXXXXXXXXXXXXXXXXXXXX
[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
```

此示範會針對具有 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

端點	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。

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. 的受管整合 AWS IoT Device Management 處於公有預覽。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附錄 A：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。這些檔案為您需要實作的函數提供範本。

   - 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. 比較您平台的 MbedTLS 組態與 中的 SDK `platform/posix/mbedtls/mbedtls_config.h` 需求。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：

   - 在 檢閱使用 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 項目以呼叫現有的 main() 函數。

3. 確認您可以成功編譯示範應用程式。

附錄

主題

附錄 A：支援的平台

下表顯示 SDK 支援的平台。

支援平台

平台	架構	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

附錄 B：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）

SDK 和元件	空間需求（使用的位元組數）
預設 MQTT Agent 傳入佇列	320 位元組（可設定）

附錄 C：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);
```

什麼是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。 AWS IoT Smart Home Hub SDK 的裝置加入和裝置控制元件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。
- 通訊協定特定的應用程式堆疊，例如 Zigbee Cluster Library (ZCL) 和 BLE 網格。

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)，它提供常見的集合 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 IotManagedIntegrationsDeviceSDK-Middleware/telus-iot-ace-dpk/telus/dpk/ace_hal/zigbee 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 Zigbee SDK

Silicon Labs 開發套件會顯示在 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 IoTmanagedintegrationsMiddlewares/telus-iot-ace-zwave-mw/ 資料夾內。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
```

將裝置 SDK 的中介軟體部分整合到您的中樞

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
- 參考實作和程式碼組織

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） APIsacehal_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。需要下列兩次修改，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。

的受管整合中的安全性 AWS IoT Device Management

的雲端安全性 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，這些架構是為了滿足最安全敏感組織的需求而建置。

安全是 AWS 與您之間的共同責任。共同責任模型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 合規計劃的一部分，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 共同的責任模型和 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

的受管整合預設 AWS IoT Device Management 提供資料加密，以使用加密金鑰保護靜態的敏感客戶資料。

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 為 `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 ，以確認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`managedThing`代表實體裝置的 ，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 委託人使用加密金鑰。

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 ，而且您無法檢視、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 Management Console 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》中的建立 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 Management Console

若要更新 中的加密組態 AWS Management Console，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，您可以透過選取新 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) 是一種 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 是您可以免費使用 AWS 服務 的 。

主題

目標對象

AWS Identity and Access Management (IAM) 的使用方式會有所不同，取決於您在受管整合中所做的工作。

服務使用者 – 如果您使用 受管整合服務來執行任務，您的管理員會為您提供所需的登入資料和許可。當您使用更多受管整合功能來執行工作時，您可能需要額外的許可。了解存取許可的管理方式可協助您向管理員請求正確的許可。如果您無法存取受管整合中的功能，請參閱 對受管整合身分和存取進行故障診斷。

服務管理員 – 如果您負責公司中的受管整合資源，您可能擁有受管整合的完整存取權。您的任務是判斷服務使用者應存取的受管整合功能和資源。接著，您必須將請求提交給您的 IAM 管理員，來變更您服務使用者的許可。檢閱此頁面上的資訊，了解 IAM 的基本概念。若要進一步了解貴公司如何搭配 受管整合使用 IAM，請參閱 受管整合如何與 IAM 搭配使用。

IAM 管理員 – 如果您是 IAM 管理員，建議您了解撰寫政策以管理受管整合存取權的詳細資訊。若要檢視您可以在 IAM 中使用的範例受管整合身分型政策，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 身分 AWS 帳戶根使用者、IAM 使用者身分或擔任 IAM 角色身分進行身分驗證 （ 登入 AWS)。

您可以使用透過身分來源提供的憑證，以聯合身分 AWS 身分身分身分身分登入 。 AWS IAM Identity Center (IAM Identity Center) 使用者、您公司的單一登入身分驗證，以及您的 Google 或 Facebook 登入資料，都是聯合身分的範例。您以聯合身分登入時，您的管理員先前已設定使用 IAM 角色的聯合身分。當您使用聯合 AWS 身分存取 時，您會間接擔任角色。

根據您身分的使用者類型，您可以登入 AWS Management Console 或 AWS 存取入口網站。如需登入的詳細資訊 AWS，請參閱AWS 登入 《 使用者指南》中的如何登入您的 AWS 帳戶 。

如果您以 AWS 程式設計方式存取 ， AWS 會提供軟體開發套件 (SDK) 和命令列界面 (CLI)，以使用您的登入資料以密碼編譯方式簽署您的請求。如果您不使用 AWS 工具，則必須自行簽署請求。如需使用建議的方法自行簽署請求的詳細資訊，請參閱《IAM 使用者指南》中的適用於 API 請求的AWS Signature 第 4 版。

無論您使用何種身分驗證方法，您可能都需要提供額外的安全性資訊。例如， AWS 建議您使用多重驗證 (MFA) 來提高帳戶的安全性。如需更多資訊，請參閱《AWS IAM Identity Center 使用者指南》中的多重要素驗證和《IAM 使用者指南》中的 IAM 中的AWS 多重要素驗證。

AWS 帳戶 根使用者

當您建立 時 AWS 帳戶，您會從一個登入身分開始，該身分具有帳戶內所有 AWS 服務 和 資源的完整存取權。此身分稱為 AWS 帳戶 Theroot 使用者，可透過使用您用來建立帳戶的電子郵件地址和密碼登入來存取。強烈建議您不要以根使用者處理日常任務。保護您的根使用者憑證，並將其用來執行只能由根使用者執行的任務。如需這些任務的完整清單，了解需以根使用者登入的任務，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，包括需要管理員存取權的使用者，使用 聯合身分提供者 AWS 服務 來使用臨時憑證來存取 。

聯合身分是來自您的企業使用者目錄、Web 身分提供者、 AWS Directory Service、身分中心目錄或任何使用透過身分來源提供的憑證 AWS 服務 存取的使用者。當聯合身分存取時 AWS 帳戶，它們會擔任角色，而角色會提供臨時登入資料。

對於集中式存取權管理，我們建議您使用 AWS IAM Identity Center。您可以在 IAM Identity Center 中建立使用者和群組，也可以連接並同步到您自己的身分來源中的一組使用者 AWS 帳戶 和群組，以便在所有 和應用程式中使用。如需 IAM Identity Center 的詳細資訊，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 中的身分 AWS 帳戶 ，具有單一人員或應用程式的特定許可。建議您盡可能依賴臨時憑證，而不是擁有建立長期憑證 (例如密碼和存取金鑰) 的 IAM 使用者。但是如果特定使用案例需要擁有長期憑證的 IAM 使用者，建議您輪換存取金鑰。如需更多資訊，請參閱 IAM 使用者指南中的為需要長期憑證的使用案例定期輪換存取金鑰。

IAM 群組是一種指定 IAM 使用者集合的身分。您無法以群組身分簽署。您可以使用群組來一次為多名使用者指定許可。群組可讓管理大量使用者許可的程序變得更為容易。例如，您可以擁有一個名為 IAMAdmins 的群組，並給予該群組管理 IAM 資源的許可。

使用者與角色不同。使用者只會與單一人員或應用程式建立關聯，但角色的目的是在由任何需要它的人員取得。使用者擁有永久的長期憑證，但角色僅提供臨時憑證。如需更多資訊，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 中具有特定許可 AWS 帳戶 的身分。它類似 IAM 使用者，但不與特定的人員相關聯。若要暫時在 中擔任 IAM 角色 AWS Management Console，您可以從使用者切換至 IAM 角色（主控台）。您可以透過呼叫 AWS CLI 或 AWS API 操作或使用自訂 URL 來擔任角色。如需使用角色的方法詳細資訊，請參閱《IAM 使用者指南》中的擔任角色的方法。

使用臨時憑證的 IAM 角色在下列情況中非常有用：

- 聯合身分使用者存取 — 如需向聯合身分指派許可，請建立角色，並為角色定義許可。當聯合身分進行身分驗證時，該身分會與角色建立關聯，並獲授予由角色定義的許可。如需有關聯合角色的相關資訊，請參閱《IAM 使用者指南》中的為第三方身分提供者 (聯合) 建立角色。如果您使用 IAM Identity Center，則需要設定許可集。為控制身分驗證後可以存取的內容，IAM Identity Center 將許可集與 IAM 中的角色相關聯。如需有關許可集的資訊，請參閱 AWS IAM Identity Center 使用者指南中的許可集。
- 暫時 IAM 使用者許可 – IAM 使用者或角色可以擔任 IAM 角色來暫時針對特定任務採用不同的許可。
- 跨帳戶存取權：您可以使用 IAM 角色，允許不同帳戶中的某人 (信任的主體) 存取您帳戶的資源。角色是授予跨帳戶存取權的主要方式。不過，對於某些 AWS 服務，您可以直接將政策連接到資源（而不是使用角色做為代理）。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。
- 跨服務存取 – 有些 AWS 服務 使用其他 中的功能 AWS 服務。例如，當您在服務中進行呼叫時，該服務通常會在 Amazon EC2 中執行應用程式或將物件儲存在 Amazon Simple Storage Service (Amazon S3) 中。服務可能會使用呼叫主體的許可、使用服務角色或使用服務連結角色來執行此作業。
  - 轉送存取工作階段 (FAS) – 當您使用 IAM 使用者或角色在其中執行動作時 AWS，您被視為委託人。使用某些服務時，您可能會執行某個動作，進而在不同服務中啟動另一個動作。FAS 使用呼叫 的委託人許可 AWS 服務，結合 AWS 服務 請求向下游服務提出請求。只有在服務收到需要與其他 AWS 服務 或 資源互動才能完成的請求時，才會提出 FAS 請求。在此情況下，您必須具有執行這兩個動作的許可。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》。

- 服務角色 – 服務角色是服務擔任的 IAM 角色，可代表您執行動作。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權給 AWS 服務。
- 服務連結角色 – 服務連結角色是一種連結至 的服務角色類型 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。

- 在 Amazon EC2 上執行的應用程式 – 您可以使用 IAM 角色來管理在 EC2 執行個體上執行之應用程式的臨時登入資料，以及提出 AWS CLI 或 AWS API 請求。這是在 EC2 執行個體內儲存存取金鑰的較好方式。若要將 AWS 角色指派給 EC2 執行個體，並將其提供給其所有應用程式，您可以建立連接至執行個體的執行個體描述檔。執行個體設定檔包含該角色，並且可讓 EC2 執行個體上執行的程式取得臨時憑證。如需詳細資訊，請參閱《IAM 使用者指南》中的使用 IAM 角色來授予許可權給 Amazon EC2 執行個體上執行的應用程式。

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。政策是 中的物件，當與身分或資源建立關聯時， AWS 會定義其許可。當委託人（使用者、根使用者或角色工作階段）發出請求時， 會 AWS 評估這些政策。政策中的許可決定是否允許或拒絕請求。大多數政策會以 JSON 文件 AWS 的形式存放在 中。如需 JSON 政策文件結構和內容的詳細資訊，請參閱 IAM 使用者指南中的 JSON 政策概觀。

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。

預設情況下，使用者和角色沒有許可。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。然後，管理員可以將 IAM 政策新增至角色，使用者便能擔任這些角色。

IAM 政策定義該動作的許可，無論您使用何種方法來執行操作。例如，假設您有一個允許 `iam:GetRole` 動作的政策。具有該政策的使用者可以從 AWS Management Console AWS CLI、 或 AWS API 取得角色資訊。

身分型政策

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。

身分型政策可進一步分類成內嵌政策或受管政策。內嵌政策會直接內嵌到單一使用者、群組或角色。受管政策是獨立的政策，您可以連接到 中的多個使用者、群組和角色 AWS 帳戶。受管政策包括 AWS 受管政策和客戶受管政策。如需了解如何在受管政策及內嵌政策之間選擇，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

資源型政策是連接到資源的 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。

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。

存取控制清單 (ACL)

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。

Amazon S3 AWS WAF和 Amazon VPC 是支援 ACLs 的服務範例。如需進一步了解 ACL，請參閱 Amazon Simple Storage Service 開發人員指南中的存取控制清單 (ACL) 概觀。

其他政策類型

AWS 支援其他較不常見的政策類型。這些政策類型可設定較常見政策類型授予您的最大許可。

- 許可界限 – 許可範圍是一種進階功能，可供您設定身分型政策能授予 IAM 實體 (IAM 使用者或角色) 的最大許可。您可以為實體設定許可界限。所產生的許可會是實體的身分型政策和其許可界限的交集。會在 `Principal` 欄位中指定使用者或角色的資源型政策則不會受到許可界限限制。所有這類政策中的明確拒絕都會覆寫該允許。如需許可界限的詳細資訊，請參閱 IAM 使用者指南中的 IAM 實體許可界限。
- 服務控制政策 SCPs) – SCPs是 JSON 政策，可指定 中組織或組織單位 (OU) 的最大許可 AWS Organizations。 AWS Organizations 是一種服務，用於分組和集中管理您企業擁有 AWS 帳戶 的多個 。若您啟用組織中的所有功能，您可以將服務控制政策 (SCP) 套用到任何或所有帳戶。SCP 會限制成員帳戶中實體的許可，包括每個實體 AWS 帳戶根使用者。如需 Organizations 和 SCP 的詳細資訊，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) - RCP 是 JSON 政策，可用來設定您帳戶中資源的可用許可上限，採取這種方式就不需要更新附加至您所擁有的每個資源的 IAM 政策。RCP 會限制成員帳戶中資源的許可，並可能影響身分的有效許可，包括 AWS 帳戶根使用者，無論它們是否屬於您的組織。如需

Organizations 和 RCPs的詳細資訊，包括 AWS 服務 支援 RCPs 清單，請參閱AWS Organizations 《 使用者指南》中的資源控制政策 (RCPs)。

- 工作階段政策 – 工作階段政策是一種進階政策，您可以在透過撰寫程式的方式建立角色或聯合使用者的暫時工作階段時，做為參數傳遞。所產生工作階段的許可會是使用者或角色的身分型政策和工作階段政策的交集。許可也可以來自資源型政策。所有這類政策中的明確拒絕都會覆寫該允許。如需詳細資訊，請參閱 IAM 使用者指南中的工作階段政策。

多種政策類型

將多種政策類型套用到請求時，其結果形成的許可會更為複雜、更加難以理解。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 AWS 新增新操作和資源的唯讀許可。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 Management Console，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。

```
{
  "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 Management Console，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能力。

- `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。

```
{
  "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 受管 AWS 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	日期
受管整合開始追蹤變更	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。

可與 受管整合搭配使用的 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

IAM 功能	受管整合支援
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
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《AWS IAM 使用者指南》中的與 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。您無法在身分型政策中指定主體，因為這會套用至連接的使用者或角色。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，您可以指定在其他帳戶內的所有帳戶或 IAM 實體，做為資源型政策的主體。新增跨帳戶主體至資源型政策，只是建立信任關係的一半。當委託人和資源位於不同位置時 AWS 帳戶，信任帳戶中的 IAM 管理員也必須授予委託人實體（使用者或角色）存取資源的許可。其透過將身分型政策連接到實體來授與許可。不過，如果資源型政策會為相同帳戶中的主體授予存取，這時就不需要額外的身分型政策。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，哪個主體在什麼条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。政策動作通常具有與相關聯 AWS API 操作相同的名稱。有一些例外狀況，例如沒有相符的 API 操作的僅限許可動作。也有一些作業需要政策中的多個動作。這些額外的動作稱為相依動作。

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。陳述式必須包含 Resource 或 NotResource 元素。最佳實務是使用其 Amazon Resource Name (ARN) 來指定資源。您可以針對支援特定資源類型的動作 (稱為資源層級許可) 來這麼做。

對於不支援資源層級許可的動作 (例如列出操作)，請使用萬用字元 (*) 來表示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 (或 Condition 區塊) 可讓您指定使陳述式生效的條件。Condition 元素是選用項目。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。

若您在陳述式中指定多個 Condition 元素，或是在單一 Condition 元素中指定多個索引鍵， AWS 會使用邏輯 AND 操作評估他們。如果您為單一條件索引鍵指定多個值， 會使用邏輯OR操作 AWS 評估條件。必須符合所有條件，才會授與陳述式的許可。

您也可以在指定條件時使用預留位置變數。例如，您可以只在使用者使用其 IAM 使用者名稱標記時，將存取資源的許可授予該 IAM 使用者。如需更多資訊，請參閱 IAM 使用者指南中的 IAM 政策元素：變數和標籤。

AWS 支援全域條件金鑰和服務特定條件金鑰。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，可根據屬性來定義許可。在 中 AWS，這些屬性稱為標籤。您可以將標籤連接至 IAM 實體 （使用者或角色） 和許多 AWS 資源。為實體和資源加上標籤是 ABAC 的第一步。您接著要設計 ABAC 政策，允許在主體的標籤與其嘗試存取的資源標籤相符時操作。

ABAC 在成長快速的環境中相當有幫助，並能在政策管理變得繁瑣時提供協助。

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

當您使用臨時登入資料登入時，有些 AWS 服務 無法運作。如需詳細資訊，包括哪些 AWS 服務 使用臨時登入資料，請參閱《AWS 服務 IAM 使用者指南》中的 使用 IAM。

如果您 AWS Management Console 使用使用者名稱和密碼以外的任何方法登入 ，則表示您使用的是暫時登入資料。例如，當您 AWS 使用公司的單一登入 (SSO) 連結存取 時，該程序會自動建立臨時登

入資料。當您以使用者身分登入主控台，然後切換角色時，也會自動建立臨時憑證。如需切換角色的詳細資訊，請參閱《IAM 使用者指南》中的從使用者切換至 IAM 角色 (主控台)。

您可以使用 AWS CLI 或 AWS API 手動建立臨時登入資料。然後，您可以使用這些臨時登入資料來存取 AWS。 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 中的暫時性安全憑證。

受管整合的跨服務主體許可

支援轉寄存取工作階段 (FAS)：是

當您使用 IAM 使用者或角色在 中執行動作時 AWS，您會被視為委託人。使用某些服務時，您可能會執行某個動作，進而在不同服務中啟動另一個動作。FAS 使用呼叫 的委託人許可 AWS 服務，結合 AWS 服務 請求向下游服務提出請求。只有在服務收到需要與其他 AWS 服務 或 資源互動才能完成的請求時，才會提出 FAS 請求。在此情況下，您必須具有執行這兩個動作的許可。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權給 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。在表格中尋找服務，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。他們也無法使用 AWS Management Console、 AWS Command Line Interface (AWS CLI) 或 AWS API 來執行任務。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。然後，管理員可以將 IAM 政策新增至角色，使用者便能擔任這些角色。

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

以身分為基礎的政策會判斷您帳戶中的某人是否可以建立、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 AWS 受管政策來授予許多常見使用案例的許可。它們可在您的 中使用 AWS 帳戶。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 等使用服務動作 AWS 服務，您也可以使用條件來授予存取 AWS 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》https://docs.aws.amazon.com/IAM/latest/UserGuide/id_credentials_mfa_configure-api-require.html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上完成此動作的許可，或使用 AWS CLI 或 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": "*"
        }
    ]
}
```

對受管整合身分和存取進行故障診斷

使用下列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，則必須更新您的政策，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 沒有將角色傳遞至該服務的許可。

```
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之 資源 AWS 帳戶 的存取權，請參閱《IAM 使用者指南》中的在您擁有 AWS 帳戶 的另一個 中提供存取權給 IAM 使用者。
- 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》中的提供存取權給第三方 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。服務連結角色是由 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.`

```
{
  "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

您不需要手動建立一個服務連結角色。當您造成事件類型，例如呼叫 PutRuntimeLogConfiguration、 CreateEventLogConfiguration或 RegisterCustomEndpoint API 中的 AWS Management Console、 AWS CLI或 AWS API 命令時， 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。當您造成事件類型，例如呼叫 PutRuntimeLogConfiguration、 CreateEventLogConfiguration或 RegisterCustomEndpoint API 命令時，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」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 區域與端點。

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 中的 AWS Artifact報告。

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。AWS 提供下列資源以協助合規：

- 安全合規與治理 - 這些解決方案實作指南內容討論了架構考量，並提供部署安全與合規功能的步驟。
- HIPAA 合格服務參考 – 列出 HIPAA 合格服務。並非所有 AWS 服務 都符合 HIPAA 資格。
- AWS 合規資源 – 此工作手冊和指南集合可能適用於您的產業和位置。
- AWS 客戶合規指南 – 透過合規的角度了解共同責任模型。本指南摘要說明保護 ， AWS 服務 並將指引映射至跨多個架構 （ 包括國家標準技術研究所 (NIST)、支付卡產業安全標準委員會 (PCI) 和國際標準組織 (ISO)) 的安全控制之最佳實務。
- 《 AWS Config 開發人員指南》中的使用 規則評估資源 – AWS Config 服務會評估資源組態符合內部實務、產業準則和法規的程度。
- AWS Security Hub – 這 AWS 服務 可讓您全面檢視其中的安全狀態 AWS。Security Hub 使用安全控制，可評估您的 AWS 資源並檢查您的法規遵循是否符合安全業界標準和最佳實務。如需支援的服務和控制清單，請參閱「Security Hub 控制參考」。
- Amazon GuardDuty – 這會監控您的環境是否有可疑和惡意活動，以 AWS 服務 偵測對您 AWS 帳戶、工作負載、容器和資料的潛在威脅。GuardDuty 可滿足特定合規架構所規定的入侵偵測需求，以協助您因應 PCI DSS 等各種不同的合規需求。
- AWS Audit Manager – 這 AWS 服務 可協助您持續稽核 AWS 用量，以簡化您管理風險和符合法規和業界標準的方式。

受管整合中的彈性

AWS 全球基礎設施是以 AWS 區域 和 可用區域為基礎建置。 AWS 區域 提供多個實體分隔且隔離的可用區域，這些可用區域與低延遲、高輸送量和高度備援聯網連線。透過可用區域，您可以設計與操作的應用程式和資料庫，在可用區域之間自動容錯移轉而不會發生中斷。可用區域的可用性、容錯能力和擴展能力，均較單一或多個資料中心的傳統基礎設施還高。

如需 AWS 區域 和可用區域的詳細資訊，請參閱 AWS 全球基礎設施。

除了 AWS 全球基礎設施之外， 的受管整合 AWS IoT Device Management 還提供數種功能，以協助支援您的資料彈性和備份需求。

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：

- Amazon CloudWatch AWS 會即時監控您的 AWS 資源和您在 上執行的應用程式。您可以收集和追蹤指標、建立自訂儀板表，以及設定警示，在特定指標達到您指定的閾值時通知您或採取動作。例如，您可以讓 CloudWatch 追蹤 CPU 使用量或其他 Amazon EC2 執行個體指標，並在需要時自動啟動新的執行個體。如需詳細資訊，請參閱 Amazon CloudWatch 使用者指南。
- Amazon CloudWatch Logs 可讓您監控、存放和存取來自 Amazon EC2 執行個體、CloudTrail 及其他來源的日誌檔案。CloudWatch Logs 可監控日誌檔案中的資訊，並在達到特定閾值時通知您。您也可以將日誌資料存檔在高耐用性的儲存空間。如需詳細資訊，請參閱 Amazon CloudWatch Logs 使用者指南。
- Amazon EventBridge 可用來自動化您的 AWS 服務，並自動回應系統事件，例如應用程式可用性問題或資源變更。來自 AWS 服務的事件會以近乎即時的方式交付至 EventBridge。您可編寫簡單的規則，來指示您在意的事件，以及當事件符合規則時所要自動執行的動作。如需詳細資訊，請參閱 Amazon EventBridge 使用者指南。
- 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」。

監控與 Amazon CloudWatch 的受管整合

您可以使用 CloudWatch 監控受管整合，這會收集原始資料並將其處理為可讀且幾近即時的指標。這些統計資料會保留 15 個月，以便您存取歷史資訊，並更清楚 Web 應用程式或服務的執行效能。您也可以設定留意特定閾值的警示，當滿足這些閾值時傳送通知或採取動作。如需詳細資訊，請參閱 Amazon CloudWatch 使用者指南。

對於受管整合，您可能想要監看 *XXX*，並同時監看 *XXX*，並在######時######。

下表列出受管整合的指標和維度。

在 Amazon EventBridge 中監控受管整合事件

您可以在 EventBridge 中監控受管整合事件，該事件可從您自己的應用程式、software-as-a-service (SaaS) 應用程式 AWS 和服務提供即時資料串流。EventBridge 會將該資料路由到目標，例如 AWS

Lambda 和 Amazon Simple Notification Service。這些事件與 Amazon CloudWatch Events 中出現的事件相同，可提供近乎即時的系統事件串流，說明 AWS 資源的變更。

下列範例顯示受管整合的事件。

主題

eventName 事件

在此範例事件中，。

```
{
   "version": "0",
   "id": "01234567-EXAMPLE",
   "detail-type": "ServiceName ResourceType State Change",
   "source": "aws.servicename",
   "account": "123456789012",
   "time": "2019-06-12T10:23:43Z",
   "region": "us-east-2",
   "resources": [
     "arn:aws:servicename:us-east-2:123456789012:resourcename"
   ],
   "detail": {
     "event": "eventName",
     "detailOne": "something",
     "detailTwo": "12345678-1234-5678-abcd-12345678abcd",
     "detailThree": "something",
     "detailFour": "something"
   }
}
```

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 Management Console 都是多區域。您可以使用 AWS CLI建立單一或多區域追蹤。建議您建立多區域線索，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

管理事件提供在 資源上執行的管理操作的相關資訊 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。

受管整合會將所有受管整合控制平面操作記錄為管理事件。如需受管整合控制平面操作的清單，這些操作會將整合日誌管理到 CloudTrail，請參閱 受管整合 API 參考。

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 `StartDeviceDiscovery` API 操作的 CloudTrail 事件。

使用 **`StartDeviceDiscovery`** API 操作成功執行 CloudTrail 事件。

```
{
    "eventVersion": "1.09",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROA47CRX4JX4AEXAMPLE",
        "arn": "arn:aws:sts::123456789012:assumed-role/Admin/EXAMPLE",
        "accountId": "222222222222",
        "accessKeyId": "access-key-id",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROA47CRX4JXUNEXAMPLE",
                "arn": "arn:aws:iam::123456789012:role/Admin",
                "accountId": "222222222222",
                "userName": "Admin"
            },
            "attributes": {
                "creationDate": "2025-02-26T20:04:25Z",
                "mfaAuthenticated": "false"
            }
        }
    },
    "eventTime": "2025-02-26T20:11:33Z",
```

```
    "eventSource": "gamma-iotmanagedintegrations.amazonaws.com",
    "eventName": "StartDeviceDiscovery",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "15.248.7.123",
    "userAgent": "aws-sdk-java/2.30.21 md/io#sync md/http#Apache ua/2.1 os/
Mac_OS_X#15.3.1 lang/java#17.0.13 md/OpenJDK_64-Bit_Server_VM#17.0.13+11-LTS md/
vendor#Amazon.com_Inc. md/en_US cfg/auth-source#stat m/D,N",
    "requestParameters": {
        "DiscoveryType": "ZIGBEE",
        "ControllerIdentifier": "554a1e3f7c884e67a21e0cabac3a48e3"
    },
    "responseElements": {
        "X-Frame-Options": "DENY",
        "Access-Control-Expose-Headers": "Content-Length,Content-Type,X-Amzn-
Errortype,X-Amzn-Requestid",
        "Strict-Transport-Security": "max-age:47304000; includeSubDomains",
        "Cache-Control": "no-store, no-cache",
        "X-Content-Type-Options": "nosniff",
        "Content-Security-Policy": "upgrade-insecure-requests; default-src 'none';
 object-src 'none'; frame-ancestors 'none'; base-uri 'none'",
        "Pragma": "no-cache",
        "Id": "717023e159264ec5ba97293e4d884d3a",
        "StartedAt": 1740600693.789,
        "Arn": "arn:aws:iotmanagedintegrations::123456789012:device-
discovery/717023e159264ec5ba97293e4d884d3a"
    },
    "requestID": "29aa09b9-ad0e-42dc-8b7f-565a1a56c020",
    "eventID": "d8d0a6ab-b729-4aa5-8af0-9f605ee90d0f",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "123456789012",
    "eventCategory": "Management"
}
```


使用 **`StartDeviceDiscovery`** API 操作存取拒絕的 CloudTrail 事件。


```
{
    "eventVersion": "1.09",
    "userIdentity": {
        "type": "AssumedRole",
```

```
        "principalId": "AROA47CRX4JX4AEXAMPLE",
        "arn": "arn:aws:sts::123456789012:assumaDMINed-role/EXAMPLEExplicitDenyRole/
EXAMPLE",
        "accountId": "222222222222",
        "accessKeyId": "access-key-id",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROA47CRX4JXUNEXAMPLE",
                "arn": "arn:aws:iam::123456789012:role/EXAMPLEExplicitDenyRole",
                "accountId": "222222222222",
                "userName": "EXAMPLEExplicitDenyRole"
            },
            "attributes": {
                "creationDate": "2025-02-27T21:36:55Z",
                "mfaAuthenticated": "false"
            }
        },
        "invokedBy": "AWS Internal"
    },
    "eventTime": "2025-02-27T21:37:01Z",
    "eventSource": "gamma-iotmanagedintegrations.amazonaws.com",
    "eventName": "StartDeviceDiscovery",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "AWS Internal",
    "userAgent": "AWS Internal",
    "errorCode": "AccessDenied",
    "requestParameters": {
        "DiscoveryType": "CLOUD",
        "ClientToken": "ClientToken",
        "ConnectorAssociationIdentifier": "ConnectorAssociation"
    },
    "responseElements": {
        "message": "User: arn:aws:sts::123456789012:assumed-role/
EXAMPLEExplicitDenyRole/EXAMPLE is not authorized to perform:
 iotmanagedintegrations:StartDeviceDiscovery on resource:
 arn:aws:iotmanagedintegrations:us-east-1:123456789012:/device-discoveries with an
 explicit deny"
    },
    "requestID": "5eabd798-d79c-4d76-a5dd-115be230d77a",
    "eventID": "cc75660c-f628-462a-9e6e-83dab40c5246",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "managementEvent": true,
```

```
    "recipientAccountId": "123456789012",
    "eventCategory": "Management"
}
```

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
初始版本	受管整合開發人員指南的初始版本	2025 年 3 月 3 日