aws

Entwicklerhandbuch

Verwaltete Integrationen für AWS IoT Device Management

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Verwaltete Integrationen für AWS IoT Device Management: Entwicklerhandbuch

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Die Handelsmarken und Handelsaufmachung von Amazon dürfen nicht in einer Weise in Verbindung mit nicht von Amazon stammenden Produkten oder Services verwendet werden, durch die Kunden irregeführt werden könnten oder Amazon in schlechtem Licht dargestellt oder diskreditiert werden könnte. Alle anderen Handelsmarken, die nicht Eigentum von Amazon sind, gehören den jeweiligen Besitzern, die möglicherweise zu Amazon gehören oder nicht, mit Amazon verbunden sind oder von Amazon gesponsert werden.

Table of Contents

Wozu dienen verwaltete Integrationen? AWS IoT Device Management

Managed Integrations for AWS IoT Device Management hilft Anbietern von IoT-Lösungen dabei, die Steuerung und Verwaltung von IoT-Geräten von Hunderten von Herstellern zu vereinheitlichen. Sie können verwaltete Integrationen verwenden, um Arbeitsabläufe bei der Geräteeinrichtung zu automatisieren und die Interoperabilität zwischen vielen Geräten zu unterstützen, unabhängig vom Gerätehersteller oder dem Konnektivitätsprotokoll. Mit verwalteten Integrationen können Sie eine einzige Benutzeroberfläche und eine Reihe von Geräten verwenden, APIs um eine Reihe von Geräten zu steuern, zu verwalten und zu bedienen.

Themen

Unterstützte Regionen

Managed Integrations for AWS IoT Device Management wird in den folgenden Regionen unterstützt:

- Kanada (Zentral)
- Europa (Irland)

Verwenden Sie zum ersten Mal verwaltete Integrationen?

Wenn Sie zum ersten Mal verwaltete Integrationen verwenden, empfehlen wir Ihnen, zunächst die folgenden Abschnitte zu lesen:

- Verwaltete Integrationen einrichten
- Beginnen Sie mit verwalteten Integrationen für AWS IoT Device Management

Überblick über verwaltete Integrationen

Die folgende Abbildung bietet einen allgemeinen Überblick über verwaltete Integrationen

Terminologie für verwaltete Integrationen

Im Bereich verwalteter Integrationen gibt es viele Konzepte und Begriffe, die Sie für die Verwaltung Ihrer eigenen Geräteimplementierungen unbedingt verstehen sollten. In den folgenden Abschnitten werden diese wichtigsten Konzepte und Begriffe beschrieben, um ein besseres Verständnis verwalteter Integrationen zu vermitteln.

Allgemeine Terminologie für verwaltete Integrationen

Ein wichtiges Konzept, das Sie bei verwalteten Integrationen verstehen sollten, ist ein verwaltetes Ding im Vergleich zu anderen Dingen. AWS IoT Core

- AWS IoT Core Ding: Ein AWS IoT Core Ding ist ein AWS IoT Core Konstrukt, das für die digitale Repräsentation sorgt. Von Entwicklern wird erwartet, dass sie Richtlinien, Datenspeicher, Regeln, Aktionen, MQTT-Themen und die Übertragung des Gerätestatus an den Datenspeicher verwalten. Weitere Informationen darüber, was ein AWS IoT Core Ding ist, finden Sie unter Geräte verwalten mit AWS IoT.
- Verwaltete Integrationen Verwaltetes Ding: Mit einem verwalteten Ding bieten wir eine Abstraktion, um Geräteinteraktionen zu vereinfachen, sodass der Entwickler keine Elemente wie Regeln, Aktionen, MQTT-Themen und Richtlinien erstellen muss.

Cloud-to-cloud Terminologie

Physische Geräte, die in verwaltete Integrationen integriert werden, können von einem Cloud-Drittanbieter stammen. Um diese Geräte in verwaltete Integrationen einzubinden und mit dem Drittanbieter zu kommunizieren, deckt die folgende Terminologie einige der wichtigsten Konzepte ab, die diese Workflows unterstützen:

- Cloud-to-cloud (C2C) -Konnektor: Ein C2C-Konnektor stellt eine Verbindung zwischen verwalteten Integrationen und dem Cloud-Drittanbieter her.
- Cloud-Drittanbieter: Bei Geräten, die außerhalb von verwalteten Integrationen hergestellt und verwaltet werden, ermöglicht ein Drittanbieter-Cloud-Anbieter die Steuerung dieser Geräte für den

Endbenutzer, und Managed Integrations kommuniziert mit dem Drittanbieter-Cloud-Anbieter für verschiedene Workflows wie Gerätebefehle.

Terminologie des Datenmodells

Managed Integrations verwendet Datenmodelle für die Organisation von Daten und die end-to-end Kommunikation zwischen Ihren Geräten. Die folgende Terminologie behandelt einige der wichtigsten Konzepte zum Verständnis dieser beiden Datenmodelle:

- Gerät: Eine Einheit, die ein physisches Gerät (z. B. eine Video-Türklingel) darstellt, bei dem mehrere Knoten zusammenarbeiten, um einen vollständigen Funktionsumfang bereitzustellen.
- Endpunkt: Ein Endpunkt kapselt eine eigenständige Funktion (Klingelton, Bewegungserkennung, Beleuchtung in einer Video-Türklingel).
- Fähigkeit: Eine Einheit, die Komponenten darstellt, die benötigt werden, um eine Funktion in einem Endpunkt verfügbar zu machen (Taste oder ein Licht und ein Glockenspiel in der Klingelfunktion einer Video-Türklingel).
- Aktion: Eine Entität, die eine Interaktion mit einer Funktion eines Geräts darstellt (klingeln oder sehen, wer an der Tür ist).
- Ereignis: Eine Entität, die ein Ereignis im Zusammenhang mit einer Fähigkeit eines Geräts darstellt. Ein Gerät kann ein Ereignis senden, um ein Ereignis zu melden (ein Ereignis incident/alarm, an activity from a sensor etc. (e.g. there is knock/ring an der Tür).
- Eigenschaft: Eine Entität, die ein bestimmtes Attribut im Gerätestatus darstellt (die Glocke klingelt, das Licht auf der Veranda ist an, die Kamera zeichnet auf).
- Datenmodell: Die Datenschicht entspricht den Daten- und Verbelementen, die die Funktionalität der Anwendung unterstützen. Die Anwendung arbeitet mit diesen Datenstrukturen, wenn beabsichtigt ist, mit dem Gerät zu interagieren. Weitere Informationen finden Sie unter connectedhomeip auf der Website. GitHub
- Schema: Ein Schema ist eine Darstellung des Datenmodells im JSON-Format.

Verwaltete Integrationen einrichten

Die folgenden Abschnitte führen Sie durch die Ersteinrichtung für die Verwendung verwalteter Integrationen für AWS IoT Device Management.

Topics

Melden Sie sich an für eine AWS-Konto

Wenn Sie noch keine haben AWS-Konto, führen Sie die folgenden Schritte aus, um eine zu erstellen.

Um sich für eine anzumelden AWS-Konto

1. Öffnen Sie https://portal.aws.amazon.com/billing/die Anmeldung.
2. Folgen Sie den Online-Anweisungen.

   Während der Anmeldung erhalten Sie einen Telefonanruf oder eine Textnachricht und müssen einen Verifizierungscode über die Telefontasten eingeben.

   Wenn Sie sich für eine anmelden AWS-Konto, Root-Benutzer des AWS-Kontoswird eine erstellt. Der Root-Benutzer hat Zugriff auf alle AWS-Services und Ressourcen des Kontos. Als bewährte Sicherheitsmethode weisen Sie einem Administratorbenutzer Administratorzugriff zu und verwenden Sie nur den Root-Benutzer, um Aufgaben auszuführen, die Root-Benutzerzugriff erfordern.

AWS sendet Ihnen nach Abschluss des Anmeldevorgangs eine Bestätigungs-E-Mail. Du kannst jederzeit deine aktuellen Kontoaktivitäten einsehen und dein Konto verwalten, indem du zu https://aws.amazon.com/gehst und Mein Konto auswählst.

Erstellen eines Benutzers mit Administratorzugriff

Nachdem Sie sich für einen angemeldet haben AWS-Konto, sichern Sie Ihren Root-Benutzer des AWS-Kontos AWS IAM Identity Center, aktivieren und erstellen Sie einen Administratorbenutzer, sodass Sie den Root-Benutzer nicht für alltägliche Aufgaben verwenden.

Sichern Sie Ihre Root-Benutzer des AWS-Kontos

1. Melden Sie sich AWS-Managementkonsoleals Kontoinhaber an, indem Sie Root-Benutzer auswählen und Ihre AWS-Konto E-Mail-Adresse eingeben. Geben Sie auf der nächsten Seite Ihr Passwort ein.

   Hilfe bei der Anmeldung mit dem Root-Benutzer finden Sie unter Anmelden als Root-Benutzer im AWS-Anmeldung Benutzerhandbuch zu.
2. Aktivieren Sie die Multi-Faktor-Authentifizierung (MFA) für den Root-Benutzer.

   Anweisungen finden Sie unter Aktivieren eines virtuellen MFA-Geräts für Ihren AWS-Konto Root-Benutzer (Konsole) im IAM-Benutzerhandbuch.

Erstellen eines Benutzers mit Administratorzugriff

1. Aktivieren Sie das IAM Identity Center.

   Anweisungen finden Sie unter Aktivieren AWS IAM Identity Center im AWS IAM Identity Center Benutzerhandbuch.
2. Gewähren Sie einem Administratorbenutzer im IAM Identity Center Benutzerzugriff.

   Ein Tutorial zur Verwendung von IAM-Identity-Center-Verzeichnis als Identitätsquelle finden Sie IAM-Identity-Center-Verzeichnis im Benutzerhandbuch unter Benutzerzugriff mit der Standardeinstellung konfigurieren.AWS IAM Identity Center

Anmelden als Administratorbenutzer

- Um sich mit Ihrem IAM-Identity-Center-Benutzer anzumelden, verwenden Sie die Anmelde-URL, die an Ihre E-Mail-Adresse gesendet wurde, als Sie den IAM-Identity-Center-Benutzer erstellt haben.

  Hilfe bei der Anmeldung mit einem IAM Identity Center-Benutzer finden Sie im AWS-Anmeldung Benutzerhandbuch unter Anmeldung beim AWS Access-Portal.

Weiteren Benutzern Zugriff zuweisen

1. Erstellen Sie im IAM-Identity-Center einen Berechtigungssatz, der den bewährten Vorgehensweisen für die Anwendung von geringsten Berechtigungen folgt.

Anweisungen hierzu finden Sie unter Berechtigungssatz erstellen im AWS IAM Identity Center Benutzerhandbuch.

2. Weisen Sie Benutzer einer Gruppe zu und weisen Sie der Gruppe dann Single Sign-On-Zugriff zu.

   Eine genaue Anleitung finden Sie unter Gruppen hinzufügen im AWS IAM Identity Center Benutzerhandbuch.

Beginnen Sie mit verwalteten Integrationen für AWS IoT Device Management

In den folgenden Abschnitten werden die Schritte beschrieben, die Sie ergreifen müssen, um mit der Verwendung verwalteter Integrationen zu beginnen.

Themen

Gerätetypen

Managed Integrations verwaltet viele Arten von Geräten. Jedes Gerät gehört zu einer der folgenden drei Kategorien:

- Direkt verbundene Geräte: Dieser Gerätetyp stellt eine direkte Verbindung zu einem verwalteten Integrationsendpunkt her. In der Regel werden diese Geräte von Geräteherstellern gebaut und verwaltet, zu denen das Endgeräte-SDK für verwaltete Integrationen für die direkte Konnektivität gehört.
- Geräte mit Hub-Verbindung: Diese Geräte stellen über einen Hub eine Verbindung zu verwalteten Integrationen her, auf dem das Hub-SDK für verwaltete Integrationen ausgeführt wird, das die Funktionen für Geräteerkennung, Onboarding und Steuerung verwaltet. Endbenutzer können diese Geräte per Tastendruck oder Barcode-Scannen einbinden.

  Die folgenden zwei Workflows werden für das Onboarding eines mit einem Hub verbundenen Geräts unterstützt:

  - Ein vom Endbenutzer initiierter Tastendruck, um die Geräteerkennung zu starten
  - Barcode-basiertes Scannen zur Durchführung der Gerätezuweisung
- Cloud-to-cloud (C2C) -Geräte: Dabei handelt es sich um Geräte, die von Anbietern entwickelt und verwaltet werden, die ihre eigene Cloud-Infrastruktur und mobile Markenanwendungen zur Gerätesteuerung unterhalten. Kunden mit verwalteten Integrationen können auf einen Katalog vorgefertigter C2C-Konnektoren zugreifen oder ihre eigenen erstellen, um IoT-Lösungen zu entwickeln, die über eine einheitliche Schnittstelle mit Clouds mehrerer Drittanbieter funktionieren.

Wenn der Endbenutzer ein C2C-Gerät zum ersten Mal einschaltet, muss es bei seinem jeweiligen Drittanbieter für verwaltete Integrationen bereitgestellt werden, damit die Gerätefunktionen und Metadaten abgerufen werden können. Nach Abschluss dieses Bereitstellungs-Workflows können verwaltete Integrationen im Namen des Endbenutzers mit dem Cloud-Gerät und dem Cloud-Drittanbieter kommunizieren.

> **Note**
>
> Ein Hub ist kein bestimmter Gerätetyp, wie oben aufgeführt. Sein Zweck besteht darin, als Controller für Smart-Home-Geräte zu fungieren und eine Verbindung zwischen verwalteten Integrationen und Cloud-Drittanbietern herzustellen. Es kann sowohl als Gerätetyp wie oben aufgeführt als auch als Hub dienen.

Verschlüsselungsschlüssel konfigurieren

Sicherheit ist von größter Bedeutung für Daten, die zwischen Endbenutzern, verwalteten Integrationen und Clouds von Drittanbietern weitergeleitet werden. Eine der Methoden, die wir zum Schutz Ihrer Gerätedaten unterstützen, ist die Verschlüsselung mithilfe eines sicheren end-to-end Verschlüsselungsschlüssels für das Routing Ihrer Daten.

Als Kunde von verwalteten Integrationen haben Sie die folgenden zwei Optionen für die Verwendung von Verschlüsselungsschlüsseln:

- Verwenden Sie den standardmäßigen Verschlüsselungsschlüssel für verwaltete Integrationen.
- Geben Sie einen an AWS KMS key , den Sie erstellt haben.

Weitere Informationen zum AWS KMS-Service finden Sie unter Key Management Service (KMS)

Wenn Sie die PutDefaultEncryptionConfigurationAPI im API-Referenzhandbuch für verwaltete Integrationen aufrufen, können Sie aktualisieren, welche Verschlüsselungsschlüsseloption Sie verwenden möchten. Standardmäßig verwenden verwaltete Integrationen den verwalteten Standardverschlüsselungsschlüssel für verwaltete Integrationen. Sie können die Konfiguration Ihres Verschlüsselungsschlüssels jederzeit mithilfe der PutDefaultEncryptionConfigurationAPI aktualisieren.

Darüber hinaus werden beim Aufrufen des GetDefaultEncryptionConfigurationAPI-Befehls Informationen zur Verschlüsselungskonfiguration für das AWS Konto in der Standardregion oder in der angegebenen Region zurückgegeben.

Onboarding-Techniken

Nachfolgend sind die Arten des Onboardings aufgeführt:

Onboarding direkt verbundener Geräte

Schritte Bereitsteller zum Onboarding eines direkt verbundenen Geräts finden Sie unter.

Onboarding des Hubs

Die Schritte Integrieren Sie Ihre Hubs in verwaltete Integrationen zum Onboarding des Hubs finden Sie unter.

Onboarding von Geräten mit Hub-Verbindung

Schritte Geräte einbinden und im Hub bedienen zum Onboarding eines mit einem Hub verbundenen Geräts finden Sie unter.

Cloud-to-cloud Onboarding von Geräten

Schritte Verwenden Sie einen C2C-Connector (Cloud-to-Cloud) zum Integrieren eines Cloud-Geräts von einem Cloud-Drittanbieter in verwaltete Integrationen finden Sie unter.

Gerätebereitstellung

Die Gerätebereitstellung erleichtert das Onboarding von Geräten, überwacht den gesamten Gerätelebenszyklus und richtet ein zentrales Repository für Geräteinformationen ein, auf das andere Aspekte verwalteter Integrationen zugreifen können. Managed Integrations bietet eine einheitliche Oberfläche für die Verwaltung verschiedener Gerätetypen und bietet Platz für Kundengeräte von Erstanbietern, die direkt über ein Device Software Development Kit (SDK) verbunden sind, oder für Geräte (COTS), die indirekt über ein commercial-off-the-shelf Hub-Gerät verbunden sind.

Bei verwalteten Integrationen hat jedes Gerät, unabhängig vom Gerätetyp, eine weltweit eindeutige Kennung, die als bezeichnet wird. `managedThingId` Diese Kennung wird beim Onboarding und bei der Verwaltung des Geräts für den gesamten Gerätelebenszyklus verwendet. Es wird vollständig durch verwaltete Integrationen verwaltet und ist für dieses spezifische Gerät in allen verwalteten Integrationen insgesamt einzigartig. AWS-Regionen Wenn ein Gerät anfänglich zu verwalteten Integrationen hinzugefügt wird, wird diese Kennung erstellt und in verwalteten Integrationen an das verwaltete Objekt angehängt. Ein verwaltetes Ding ist eine digitale Darstellung des physischen Geräts innerhalb verwalteter Integrationen, um alle Geräte-Metadaten des physischen Geräts widerzuspiegeln. Geräte von Drittanbietern können zusätzlich zu den in verwalteten Integrationen `managedThingId` gespeicherten Kennungen, die das physische Gerät repräsentieren, ihre eigene, separate eindeutige Kennung haben, die für ihre Drittanbieter-Cloud spezifisch ist.

Geräte, die bereitgestellt werden, können je nachdem, in welcher Phase des Onboarding-Flows sie sich befinden, unterschiedliche Status haben. In der folgenden Liste werden die einzelnen Bereitstellungsstatus beschrieben:

- AKTIVIERT: Das Gerät wurde gefunden und Command and Control ist verfügbar.
- ENTDECKT: Das Gerät wurde gefunden, aber Command and Control ist noch nicht verfügbar.
- NICHT VERKNÜPFT: Das verwaltete Ding wurde erstellt, es sind jedoch weitere Aktionen erforderlich, um entdeckt zu werden. Es ist von den Controllern (Hubs) AWS Cloud oder AWS IoT Managed Integrations Controllern (Hubs) aus nicht erreichbar
- PRE_ASSOCIATED: Das verwaltete Ding wurde erstellt und kann automatisch erkannt werden, sobald es eingeschaltet oder verbunden ist. Es ist von den Controllern (Hubs) AWS Cloud oder AWS IoT Managed Integrations Controllern (Hubs) aus nicht erreichbar.
- DELETE_IN_PROGRESS: Der asynchrone Löschvorgang wurde gestartet.
- GELÖSCHT: Das Gerät wurde aus dem gelöscht. AWS Cloud

- ISOLIERT: Ein zuvor entdecktes oder aktiviertes verwaltetes Ding, das nicht mehr erreichbar ist. Zum Beispiel ein Gerät für eine Drittanbieter-Cloud, dessen Connector-Verknüpfungen alle gelöscht wurden.

Der folgende Onboarding-Ablauf gilt für die Bereitstellung Ihres Hubs mit verwalteten Integrationen:

Integrieren Sie Ihre Hubs in verwaltete Integrationen: Richten Sie Core Provisioner und protokollspezifische Plug-ins ein, die zusammenarbeiten, um die Geräteauthentifizierung, Kommunikation und Einrichtung zu übernehmen.

Die folgenden Onboarding-Flows stehen für die Bereitstellung Ihrer mit dem Hub verbundenen Geräte mit verwalteten Integrationen zur Verfügung:

- Einfache Einrichtung (SS): Der Endbenutzer schaltet das IoT-Gerät ein und scannt seinen QR-Code mithilfe der Anwendung des Geräteherstellers. Das Gerät wird dann in der Managed Integrations Cloud registriert und stellt eine Verbindung zum IoT-Hub her.
- Einrichtung ohne Benutzereingriff (ZTS): Das Gerät ist vorab in der Lieferkette vorab zugeordnet. Anstatt beispielsweise, dass Endbenutzer den QR-Code des Geräts scannen, wird dieser Schritt früher abgeschlossen, um das Gerät vorab mit den Kundenkonten zu verknüpfen.
- Benutzergeführte Einrichtung (UGS): Der Endbenutzer schaltet das Gerät ein und folgt interaktiven Schritten, um es in verwaltete Integrationen einzubinden. Dies kann das Drücken einer Taste am IoT-Hub, die Verwendung einer App des Geräteherstellers oder das Drücken von Tasten sowohl am Hub als auch am Gerät beinhalten. Sie können diese Methode verwenden, wenn die einfache Einrichtung fehlschlägt.

> **Note**
>
> Der Workflow zur Gerätebereitstellung in verwalteten Integrationen ist unabhängig von den Onboarding-Anforderungen für ein Gerät. Verwaltete Integrationen bieten eine optimierte Benutzeroberfläche für das Onboarding und die Verwaltung eines Geräts, unabhängig vom Gerätetyp oder Geräteprotokoll.

Lebenszyklus von Geräten und Geräteprofilen

Durch die Verwaltung des Lebenszyklus Ihrer Geräte und Geräteprofile wird sichergestellt, dass Ihre Geräteflotte sicher ist und effizient läuft.

Themen

Gerät

Beim ersten Onboarding erstellt Managed Integrations einen digitalen Zwilling Ihres physischen Geräts, ein sogenanntes Managed Thing. Das verwaltete Ding verfügt über einem`anagedThingID`, die eine globale eindeutige Kennung bereitstellt, mit der das Gerät in verwalteten Integrationen in allen Regionen identifiziert werden kann. Das Gerät wird während der Bereitstellung mit dem lokalen Hub verbunden, um eine Echtzeitkommunikation mit verwalteten Integrationen oder einer Drittanbieter-Cloud für Geräte von Drittanbietern zu ermöglichen. Ein Gerät ist auch einem Besitzer zugeordnet, der anhand des öffentlichen `owner` Parameters APIs für ein verwaltetes Ding identifiziert wird, z. B. `GetManagedThing` Das Gerät ist je nach Gerätetyp mit dem entsprechenden Geräteprofil verknüpft.

> **Note**
>
> Ein physisches Gerät kann über mehrere Datensätze verfügen, wenn es mehrfach für verschiedene Kunden bereitgestellt wird.

Der Gerätelebenszyklus beginnt mit der Erstellung des verwalteten Dings in verwalteten Integrationen mithilfe der `CreateManagedThing` API und endet, wenn der Kunde das verwaltete Ding mithilfe der API löscht. `DeleteManagedThing` Der Lebenszyklus eines Geräts wird von den folgenden Personen verwaltet: APIs

- `CreateManagedThing`
- `ListManagedThings`
- `GetManagedThing`

- `UpdateManagedThing`
- `DeleteManagedThing`

Geräteprofil

Ein Geräteprofil steht für einen bestimmten Gerätetyp, z. B. eine Glühbirne oder eine Türklingel. Es ist einem Hersteller zugeordnet und enthält die Funktionen des Geräts. Das Geräteprofil speichert die Authentifizierungsmaterialien, die für Anfragen zur Einrichtung der Gerätekonnektivität mit verwalteten Integrationen benötigt werden. Bei den verwendeten Authentifizierungsmaterialien handelt es sich um den Barcode des Geräts.

Während des Herstellungsprozesses des Geräts kann der Hersteller seine Geräteprofile bei verwalteten Integrationen registrieren. Auf diese Weise kann der Hersteller während der Onboarding- und Bereitstellungsabläufe die erforderlichen Materialien für die Geräte aus verwalteten Integrationen beziehen. Die Metadaten aus dem Geräteprofil werden auf dem physischen Gerät gespeichert oder auf der Gerätekennzeichnung aufgedruckt. Der Lebenszyklus des Geräteprofils endet, wenn der Hersteller es in verwalteten Integrationen löscht.

Datenmodelle

Ein Datenmodell stellt die Organisationshierarchie dar, in der Daten innerhalb eines Systems organisiert sind. Darüber hinaus unterstützt es die end-to-end Kommunikation in Ihrer gesamten Geräteimplementierung. Für verwaltete Integrationen werden zwei Datenmodelle verwendet. Das Datenmodell für verwaltete Integrationen und die AWS Implementierung des Matter Data Model. Sie weisen Ähnlichkeiten auf, weisen aber auch subtile Unterschiede auf, die in den folgenden Themen beschrieben werden.

Bei Geräten von Drittanbietern werden beide Datenmodelle für die Kommunikation zwischen dem Endbenutzer, verwalteten Integrationen und dem Cloud-Drittanbieter verwendet. Um Nachrichten wie Gerätebefehle und Geräteereignisse aus den beiden Datenmodellen zu übersetzen, wird die Cloud-to-Cloud Connector-Funktionalität genutzt.

Themen

Datenmodell für verwaltete Integrationen

Das Datenmodell für verwaltete Integrationen verwaltet die gesamte Kommunikation zwischen dem Endbenutzer und den verwalteten Integrationen.

Gerätehierarchie

Die `capability` Datenelemente `endpoint` und werden verwendet, um ein Gerät im Datenmodell für verwaltete Integrationen zu beschreiben.

`endpoint`

Das `endpoint` steht für die logischen Schnittstellen oder Dienste, die von der Funktion angeboten werden.

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

```
}
```

`Capability`

Das `capability` steht für die Funktionen des Geräts.

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

Für das `capability` Datenelement gibt es drei Elemente, aus denen dieses Element besteht: `propertyaction`, und`event`. Sie können verwendet werden, um mit dem Gerät zu interagieren und es zu überwachen.

- Eigenschaft: Status, die vom Gerät gespeichert werden, z. B. die aktuelle Helligkeitsstufe einer dimmbaren Leuchte.
  - ```
    {
        "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

    }
    ```
- Aktion: Aufgaben, die ausgeführt werden können, z. B. das Verriegeln einer Tür an einem Türschloss. Aktionen können zu Reaktionen und Ergebnissen führen.
  - ```
    {
        "name": { "$ref": "/schema-versions/definition/aws.name@1.0" }, //required
        "parameters": Map<String name, JSONNode value>,
        "responseCode": HTTPResponseCode,
        "errors": {
    ```

```
        "code": "string",
        "message": "string"
    }
}
```

- Ereignis: Im Wesentlichen eine Aufzeichnung vergangener Zustandsübergänge. Ereignisse `property` stellen zwar die aktuellen Zustände dar, sind aber ein Journal der Vergangenheit und beinhalten einen monoton ansteigenden Zähler, einen Zeitstempel und eine Priorität. Sie ermöglichen die Erfassung von Zustandsübergängen sowie die Datenmodellierung, die mit dieser Methode nicht ohne weiteres erreicht werden kann. `property`
  - ```
    {
        "name": { "$ref": "/schema-versions/definition/aws.name@1.0" },        //
    required
        "parameters": Map<String name, JSONNode value>
    }
    ```

AWS Implementierung des Matter-Datenmodells

Die AWS Implementierung des Matter Data Model verwaltet die gesamte Kommunikation zwischen verwalteten Integrationen und Cloud-Drittanbietern.

Weitere Informationen finden Sie unter Matter Data Model: Entwicklerressourcen.

Gerätehierarchie

Es gibt zwei Datenelemente, die zur Beschreibung eines Geräts verwendet werden: `endpoint` und`cluster`.

`endpoint`

Das `endpoint` steht für die logischen Schnittstellen oder Dienste, die von der Funktion angeboten werden.

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

`cluster`
```

Das `cluster` steht für die Funktionen des Geräts.

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

Für das `cluster` Datenelement gibt es drei Elemente, aus denen dieses Element besteht: `attribute`command, und`event`. Sie können verwendet werden, um mit dem Gerät zu interagieren und es zu überwachen.

- Attribut: Status, die vom Gerät gespeichert werden, z. B. die aktuelle Helligkeitsstufe einer dimmbaren Leuchte.
  - ```
    {
        "id" (hexadecimalString): (JsonNode) value
    }
    ```
- Befehl: Aufgaben, die ausgeführt werden können, z. B. das Verriegeln einer Tür an einem Türschloss. Befehle können zu Antworten und Ergebnissen führen.
  - ```
    "id": {
        "fieldId": "fieldValue",
        ...
        "responseCode": HTTPResponseCode,
        "errors": {
            "code": "string",
            "message": "string"
        }
    }
    ```
- Ereignis: Im Wesentlichen eine Aufzeichnung vergangener Zustandsübergänge. Ereignisse `attributes` stellen zwar die aktuellen Zustände dar, sind aber ein Journal der Vergangenheit und beinhalten einen monoton ansteigenden Zähler, einen Zeitstempel und eine Priorität. Sie ermöglichen die Erfassung von Zustandsübergängen sowie die Datenmodellierung, die mit dieser Methode nicht ohne weiteres erreicht werden kann. `attributes`
  - ```
    "id": {
        "fieldId": "fieldValue",
    ```
    ```

```
    ...
}
```

Schemata für Datenmodelle

Managed Integrations unterstützt zwei Schematypen: Capability und Typdefinition. Wenn Sie ein benutzerdefiniertes Datenmodell erstellen, verwenden Sie ein JSON-Schemadokument, um beide Schematypen zu definieren. Jedes Schemadokument hat ein Limit von 50.000 Zeichen.

Funktionsschema

Eine Fähigkeit ist ein grundlegender Baustein, der bestimmte Funktionen innerhalb eines Endpunkts darstellt. Mithilfe von Funktionen können Sie Gerätestatus und -verhalten anhand von Eigenschaften, Aktionen und Ereignissen modellieren. Mithilfe von Eigenschaften können Sie die Statusattribute des Geräts flexibel mit jedem deklarativen Datentyp modellieren. Aktionen und Ereignisse modellieren das Verhalten des Geräts, einschließlich Befehle, die es ausführen kann, und Signalen, die es melden kann.

Im Folgenden wird eine allgemeine Struktur eines Funktionsschemas dargestellt.

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

Aktion

Eine Entität, die eine Interaktion mit einer Fähigkeit eines Geräts darstellt. Läuten Sie beispielsweise die Glocke oder sehen Sie sich an, wer an der Tür steht.

Ereignis

Eine Entität, die ein Ereignis aus einer Fähigkeit eines Geräts darstellt. Ein Gerät kann ein Ereignis senden, um einen Vorfall, einen Alarm oder eine Aktivität eines Sensors zu melden, z. B. ein Klopfen an der Tür.

Eigenschaft

Eine Entität, die ein bestimmtes Attribut im Status des Geräts darstellt. Zum Beispiel klingelt eine Glocke oder das Licht auf der Veranda ist an

Jede Funktion umfasst eine eindeutige Namespace-ID, Versionsinformationen und eine Beschreibung ihres Zwecks. Das Schemadokument verwendet semantische Versionierung, um die Abwärtskompatibilität aufrechtzuerhalten und gleichzeitig neue Funktionen zu ermöglichen.

Weitere Informationen finden Sie unter Schema für Funktionsdefinitionen.

Typdefinitionsschema

Eine Typdefinition ist ein deklarativer strukturierter Datentyp, der Wiederverwendbarkeit und Zusammensetzbarkeit ermöglicht. Sie definiert, wie Informationen formatiert und eingeschränkt werden sollten. Verwenden Sie Typdefinitionen, um standardisierte Datenformate für Ihre IoT-Lösung zu erstellen.

Jede Typdefinition umfasst:

- Eine eindeutige Namespace-ID
- Title
- Beschreibung
- Eigenschaften, die Datenformatierung und Einschränkungen definieren

Typen können entweder einfache Grundelemente wie Ganzzahlen oder Zeichenketten mit definierten Grenzwerten oder komplexe Strukturen wie Aufzählungen oder benutzerdefinierte Objekte mit mehreren Feldern sein. Typdefinitionen verwenden die JSON-Schemasyntax, um Einschränkungen wie Mindest- und Höchstwerte, Zeichenkettenlängen und zulässige Muster anzugeben.

Weitere Informationen finden Sie unter Schema für Typdefinitionen.

Schema für Funktionsdefinitionen

Eine Fähigkeit wird mithilfe eines deklarativen JSON-Dokuments dokumentiert, das einen klaren Vertrag darüber enthält, wie die Fähigkeit innerhalb des Systems funktionieren soll.

Bei einer Fähigkeit sind die obligatorischen Elemente `$idname`,`extrinsicId`, `extrinsicVersion` und mindestens ein Element in mindestens einem der folgenden Abschnitte:

- `properties`
- `actions`
- `events`

Die optionalen Elemente in einer Fähigkeit sind `$reftitle`,`description`,`version`, `$defs` und`extrinsicProperties`. Für eine Fähigkeit `$ref` muss auf verwiesen werden`aws.capability`.

In den folgenden Abschnitten wird das für Funktionsdefinitionen verwendete Schema detailliert beschrieben.

$id (verpflichtend)

Das $id-Element identifiziert die Schemadefinition. Es muss dieser Struktur folgen:

- Beginnen Sie mit dem `/schema-versions/` URI-Präfix
- Schließen Sie den `capability` Schematyp ein
- Verwenden Sie einen Schrägstrich (`/`) als URI-Pfadtrennzeichen
- Fügen Sie die Schema-Identität ein, wobei die Fragmente durch Punkte (`.`) getrennt sind
- Verwenden Sie das `@` Zeichen, um die Schema-ID und die Version zu trennen
- Beenden Sie mit der Serverversion und verwenden Sie Punkte (`.`), um Versionsfragmente zu trennen

Die Schema-Identität muss mit einem Stamm-Namespace beginnen, der 3-12 Zeichen lang ist, gefolgt von einem optionalen Unternamespace und einem Namen.

Die Semver-Version umfasst eine HAUPTVERSION (bis zu 3 Ziffern), eine NEBENVERSION (bis zu 3 Ziffern) und eine optionale PATCH-Version (bis zu 4 Ziffern).

> **Note**
>
> Sie können die reservierten `aws` Namespaces nicht verwenden oder `matter`

Example Beispiel $id

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

$ref

Das `$ref` Element verweist auf eine bestehende Fähigkeit innerhalb des Systems. Es unterliegt den gleichen Einschränkungen wie das `$id` Element.

> **Note**
>
> Es muss eine Typdefinition oder Fähigkeit mit dem in der `$ref` Datei angegebenen Wert vorhanden sein.

Example Beispiel $ref

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

Name (verpflichtend)

Das Namenselement ist eine Zeichenfolge, die den Entitätsnamen im Schemadokument darstellt. Es enthält häufig Abkürzungen und muss den folgenden Regeln entsprechen:

- Enthält nur alphanumerische Zeichen, Punkte (.), Schrägstriche (/), Bindestriche (-) und Leerzeichen
- Beginne mit einem Buchstaben
- Maximal 64 Zeichen

Das name-Element wird in der Benutzeroberfläche und in der Dokumentation der Amazon Web Services Services-Konsole verwendet.

Example Beispielnamen

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

Titel

Das Titelelement ist eine beschreibende Zeichenfolge für die Entität, die durch das Schemadokument repräsentiert wird. Es kann beliebige Zeichen enthalten und wird in der Dokumentation verwendet. Die maximale Länge für einen Funktionstitel beträgt 256 Zeichen.

Example Beispieltitel

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

description

Das `description` Element bietet eine detaillierte Erläuterung der Entität, die durch das Schemadokument repräsentiert wird. Es kann beliebige Zeichen enthalten und wird in der Dokumentation verwendet. Die maximale Länge für eine Funktionsbeschreibung beträgt 2048 Zeichen

Example Beispiel für eine Beschreibung

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

version

Das Element `version` ist optional. Es ist eine Zeichenfolge, die die Version des Schemadokuments darstellt. Es gelten die folgenden Einschränkungen:

- Verwendet das Semver-Format, wobei die folgenden Versionsfragmente durch `.` (Punkte) getrennt sind.
  - `MAJOR`Version, maximal 3 Ziffern
  - `MINOR`Version, maximal 3 Ziffern
  - `PATCH`Version (optional), maximal 4 Ziffern
- Die Länge kann zwischen 3 und 12 Zeichen liegen.

Example Beispielversionen

```
1.0
```

```
1.12
```

```
1.4.1
```

Mit Capability-Versionen arbeiten

Eine Fähigkeit ist eine unveränderliche versionierte Entität. Es wird erwartet, dass bei jeder Änderung eine neue Version erstellt wird. Das System verwendet semantische Versionierung im Format MAJOR.MINOR.PATCH, wobei:

- Die MAJOR-Version nimmt zu, wenn abwärtsinkompatible API-Änderungen vorgenommen werden
- Die MINOR-Version nimmt zu, wenn Funktionen auf abwärtskompatible Weise hinzugefügt werden
- Die PATCH-Version erhöht sich, wenn geringfügige Funktionserweiterungen vorgenommen werden, die sich nicht negativ auswirken.

Die von Matter-Clustern abgeleiteten Funktionen basieren auf Version 1.4, und es wird erwartet, dass jede Matter-Version in das System importiert wird. Da die Matter-Version sowohl die MAJOR- als auch die MINOR-Ebene von Semver verwendet, können verwaltete Integrationen nur PATCH-Versionen verwenden.

Wenn Sie PATCH-Versionen für Matter hinzufügen, müssen Sie berücksichtigen, dass Matter sequentielle Revisionen verwendet. Alle PATCH-Versionen müssen der in der Matter-Spezifikation dokumentierten Version entsprechen und diese müssen abwärtskompatibel sein.

Um alle Probleme mit der Abwärtsinkompatibilität zu beheben, müssen Sie mit der Connectivity Standards Alliance (CSA) zusammenarbeiten, um die Probleme in der Spezifikation zu lösen und eine neue Version zu veröffentlichen.

AWS-verwaltete Funktionen wurden mit einer ersten Version von veröffentlicht. `1.0` Mit diesen können alle drei Versionsebenen verwendet werden.

ExtrinsicVersion (verpflichtend)

Dies ist eine Zeichenfolge, die eine Version darstellt, die außerhalb des Systems verwaltet wird. AWS IoT Für `extrinsicVersion` Matter-Funktionen entspricht `revision`

Er wird als Ganzzahl mit Zeichenketten dargestellt, und die Länge kann zwischen 1 und 10 numerischen Ziffern liegen.

Example Beispielversionen

```
7
```

```
1567
```

extrinsICid (verpflichtend)

Das `extrinsicId` Element stellt eine Kennung dar, die außerhalb des IoT-Systems von Amazon Web Services verwaltet wird. Bei Matter-Funktionen wird es je nach Kontext `clusterId` `attributeId` `commandId` `eventId`,`fieldId`,, oder zugeordnet.

Dabei `extrinsicId` kann es sich entweder um eine stringifizierte dezimale Ganzzahl (1—10 Ziffern) oder um eine stringifizierte hexadezimale Ganzzahl (0x- oder 0X-Präfix, gefolgt von 1—8 hexadezimalen Ziffern) handeln.

> **Note**
>
> Für AWS ist die Lieferanten-ID (VID) 0x1577 und für Matter 0. Das System stellt sicher, dass benutzerdefinierte Schemas diese für Funktionen reservierten VIDs Daten nicht verwenden.

Example Beispiel: ExtrinsICIDs

```
0018
0x001A
0x15771002
```

$defs

Der `$defs` Abschnitt ist eine Zuordnung von Unterschemas, auf die innerhalb des Schemadokuments verwiesen werden kann, sofern das JSON-Schema dies zulässt. In dieser Map wird der Schlüssel in den lokalen Referenzdefinitionen verwendet und der Wert stellt das JSON-Schema bereit.

> **Note**
>
> Das System erzwingt nur, dass `$defs` es sich um eine gültige Map handelt und dass jedes Unterschema ein gültiges JSON-Schema ist. Es werden keine zusätzlichen Regeln durchgesetzt.

Beachten Sie bei der Arbeit mit Definitionen die folgenden Einschränkungen:

- Verwenden Sie in Definitionsnamen nur URI-freundliche Zeichen
- Stellen Sie sicher, dass jeder Wert ein gültiges Unterschema ist
- Schließen Sie eine beliebige Anzahl von Unterschemas ein, die innerhalb der Größenbeschränkungen für Schemadokumente liegen

Extrinsische Eigenschaften

Das `extrinsicProperties` Element enthält eine Reihe von Eigenschaften, die in einem externen System definiert sind, aber innerhalb des Datenmodells verwaltet werden. Bei Matter-Fähigkeiten ist es verschiedenen unmodellierten oder teilweise modellierten Elementen innerhalb eines ZCL-Clusters, -Attributs, -Befehls oder -Ereignisses zugeordnet.

Extrinsische Eigenschaften müssen den folgenden Einschränkungen entsprechen:

- Eigenschaftsnamen müssen alphanumerisch ohne Leerzeichen oder Sonderzeichen sein
- Eigenschaftswerte können beliebige JSON-Schemawerte sein
- Maximal 20 Eigenschaften

Das System unterstützt verschiedene`extrinsicProperties`, darunter `accessapiMaturity`,`cli`,`cliFunctionName`, und andere. Diese Eigenschaften erleichtern Transformationen von ACL zu Datenmodellen AWS (und umgekehrt).

> **Note**
>
> Extrinsische Eigenschaften werden für die Elemente`action`, `eventproperty`, und `struct` fields einer Fähigkeit unterstützt, nicht jedoch für die Fähigkeit oder den Cluster selbst.

Systemgestützte extrinsische Eigenschaften

Das System verfolgt die folgenden unmodellierten oder teilweise modellierten Cluster-, Attribut-, Befehls- oder Ereignisattribute wie `extrinsicProperties` bei Transformationen zu oder von ZCL:

`access`

Jedes Zugriffsobjekt enthält Folgendes:

- op- Operation, die als eine `enum` mit den Werten: `readwrite`, oder modelliert wurde `invoke`
- `privilege`- Privileg, modelliert als ein `enum` mit den Werten:`view`,, `proxy_viewoperate`, oder `manage` `administer`
- `role`- Unbegrenzte Zeichenfolge, die eine Operatorrolle darstellt

`apiMaturity`

Eine unbegrenzte einfache Zeichenfolge, die den Reifegrad darstellt. Dies wird in ZCL als `enum` mit den Werten modelliert:`stable`,, oder `provisional` `internal` `deprecated`

`side`

Modelliert als eine Aufzählung mit den Werten:,, und `either` `server` `client`

Boolesche Eigenschaften

Die folgenden Eigenschaften sind boolesche Flags:

- `isFabricScoped`
- `isFabricSensitive`
- `mustUseAtomicWrite`
- `mustUseTimedInvoke`

Eigenschaften von Zeichenketten

Die folgenden Eigenschaften werden als unbegrenzte Zeichenketten dargestellt:

- `cli`
- `cliFunctionName`
- `functionName`
- `group`
- `introducedIn`

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

Überlegungen zur Transformation

Bei ZCL-Transformationen `extrinsicProperties` werden sie ohne Verarbeitung in einer Map gespeichert. Benutzerdefinierte Schemas, die Discovery verwenden, werden keiner ZCL-Transformation unterzogen. Wenn Sie jedoch in future ZCL-Transformationen für benutzerdefinierte Schemas implementieren möchten, müssen Sie alle unbegrenzten einfachen Zeichenfolgentypen modellieren `extrinsicProperties` und Einschränkungen wie Aufzählungen, Muster (Regex) und Länge definieren. Durch diese Vorbereitung wird gewährleistet, dass diese Eigenschaften während der Transformation ordnungsgemäß behandelt werden.

Im Gegensatz dazu `extrinsicProperties` sind Transformationen für AWS zwei Konnektoren überhaupt nicht enthalten, da diese Details im Konnektorformat nicht erforderlich sind.

Eigenschaften

Eigenschaften stellen den vom Gerät verwalteten Status der Funktion dar. Jeder Status ist als Schlüssel-Wert-Paar definiert, wobei der Schlüssel den Namen des Status und der Wert die Definition des Zustands beschreibt.

Beachten Sie bei der Arbeit mit Eigenschaften die folgenden Einschränkungen:

- Verwenden Sie in Eigenschaftsnamen nur alphanumerische Zeichen ohne Leerzeichen oder Sonderzeichen
- Fügen Sie eine beliebige Anzahl von Eigenschaften ein, die innerhalb der Größenbeschränkungen für Schemadokumente liegen

Mit Eigenschaften arbeiten

Eine Eigenschaft innerhalb einer Funktion ist ein grundlegendes Element, das einen bestimmten Status eines Geräts darstellt, das durch verwaltete Integrationen unterstützt wird. Sie stellt den aktuellen Zustand oder die aktuelle Konfiguration des Geräts dar. Durch die Standardisierung der Definition und Struktur dieser Eigenschaften stellen Smart-Home-Systeme sicher, dass Geräte verschiedener Hersteller effektiv kommunizieren können, wodurch ein nahtloses und interoperables Erlebnis entsteht.

Für eine Fähigkeitseigenschaft sind `extrinsicId` die obligatorischen Elemente und. `value` Die optionalen Elemente in einer Funktionseigenschaft sind `descriptionretrievable`,`mutable`, `reportable` und`extrinsicProperties`.

Wert

Eine unbegrenzte Struktur, die es Buildern ermöglicht, beliebige JSON-Schema-konforme Einschränkungen festzulegen, um den Datentyp dieser Eigenschaft zu definieren.

Beachten Sie bei der Definition von Werten die folgenden Einschränkungen:

- Verwenden Sie `type` für einfache Typen alle anderen systemeigenen JSON-Schema-Einschränkungen wie oder `maxLength` `maximum`
- Verwenden Sie für zusammengesetzte Typen, `oneOf`, `allOf`, oder. `anyOf` Das System unterstützt das `not` Schlüsselwort nicht
- Um auf einen beliebigen globalen Typ zu verweisen, verwenden Sie es `$ref` mit einer gültigen, auffindbaren Referenz
- Für die Nullfähigkeit folgen Sie der OpenAPI-Typschemadefinition, indem Sie das nullable -Attribut mit einem booleschen Flag versehen (`true`falls Null ein zulässiger Wert ist)

Beispiel:

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

Abrufbar

Ein boolescher Wert, der beschreibt, ob der Status lesbar ist oder nicht.

Der Aspekt der Lesbarkeit des Status wird auf die Implementierung der Funktion durch das Gerät zurückgestellt. Das Gerät entscheidet, ob ein bestimmter Status lesbar ist oder nicht. Es wird noch nicht unterstützt, dass dieser Aspekt des Status im Fähigkeitsbericht gemeldet wird und daher nicht innerhalb des Systems durchgesetzt wird.

Beispiel: oder `true` `false`

Mutable

Ein boolescher Wert, der beschreibt, ob der Status beschreibbar ist oder nicht.

Der Aspekt der Schreibfähigkeit des Status wird auf die Implementierung der Funktion durch das Gerät zurückgestellt. Das Gerät entscheidet, ob ein bestimmter Status beschreibbar ist oder nicht. Es wird noch nicht unterstützt, dass dieser Aspekt des Zustands im Fähigkeitsbericht gemeldet wird und daher nicht innerhalb des Systems durchgesetzt wird.

Beispiel: oder `true` `false`

Meldepflichtig

Ein boolescher Wert, der beschreibt, ob der Status vom Gerät gemeldet wird, wenn sich der Status ändert.

Der Aspekt der Berichtbarkeit des Zustands wird auf die Implementierung der Funktion durch das Gerät zurückgestellt. Das Gerät entscheidet, ob ein bestimmter Status meldepflichtig ist oder nicht. Es wird noch nicht unterstützt, dass dieser Aspekt des Zustands im Fähigkeitsbericht gemeldet wird und daher nicht innerhalb des Systems durchgesetzt wird.

Beispiel: oder `true` `false`

Aktionen

Aktionen sind schemaverwaltete Operationen, die einem Anfrage-Antwort-Modell folgen. Jede Aktion steht für einen vom Gerät implementierten Vorgang.

Beachten Sie bei der Implementierung von Aktionen die folgenden Einschränkungen:

- Nehmen Sie nur eindeutige Aktionen in das Aktions-Array auf
- Schließen Sie eine beliebige Anzahl von Aktionen ein, die innerhalb der Größenbeschränkungen für Schemadokumente liegen

Verwenden von Aktionen

Eine Aktion ist eine standardisierte Methode, um mit Gerätefunktionen in einem verwalteten Integrationssystem zu interagieren und diese zu steuern. Sie stellt einen bestimmten Befehl oder Vorgang dar, der auf einem Gerät ausgeführt werden kann, und verfügt über ein strukturiertes Format zur Modellierung aller erforderlichen Anforderungs- oder Antwortparameter. Diese Aktionen dienen als Brücke zwischen Benutzerabsichten und Geräteoperationen und ermöglichen eine konsistente und zuverlässige Steuerung über verschiedene Arten von intelligenten Geräten hinweg.

Für eine Aktion sind die obligatorischen Elemente `name` und`extrinsicId`. Die optionalen Elemente sind `description`, `extrinsicProperties`, `request` und`response`.

Beschreibung

Die Beschreibung hat eine maximale Längenbeschränkung von 1536 Zeichen.

Anforderung

Der Anforderungsabschnitt ist optional und kann weggelassen werden, wenn keine Anforderungsparameter vorhanden sind. Wenn er weggelassen wird, unterstützt das System das Senden einer Anfrage ohne Nutzlast, indem es einfach den Namen von `Action` verwendet. Dies wird bei einfachen Aktionen verwendet, z. B. beim Ein- oder Ausschalten eines Lichts.

Die komplexen Aktionen benötigen zusätzliche Parameter. Beispielsweise kann eine Anfrage zum Streamen von Kamerabildern Parameter über das zu verwendende Streaming-Protokoll oder darüber enthalten, ob der Stream an ein bestimmtes Anzeigegerät gesendet werden soll.

Für eine Aktionsanfrage ist das obligatorische Element`parameters`. Die optionalen Elemente sind `description`, `extrinsicId`, und`extrinsicProperties`.

Beschreibung der Anforderung

Die Beschreibung folgt demselben Format wie Abschnitt 3.5, mit einer maximalen Länge von 2048 Zeichen.

Antwort

Bei verwalteten Integrationen erreicht jede über die SendManagedThingCommandAPI gesendete Aktionsanfrage das Gerät und erwartet eine asynchrone Antwort. Die Aktionsantwort definiert die Struktur dieser Antwort.

Für eine Aktionsanfrage ist das obligatorische Element`parameters`. Die optionalen Elemente sind `name`,`description`,`extrinsicId`,`extrinsicProperties`, `errors` und`responseCode`.

Beschreibung der Antwort

Die Beschreibung folgt demselben Format wie description und hat eine maximale Länge von 2048 Zeichen.

Name der Antwort

Der Name folgt demselben Format wieName (verpflichtend), mit diesen zusätzlichen Details:

- Der herkömmliche Name einer Antwort wird abgeleitet, indem er `Response` an den Aktionsnamen angehängt wird.
- Wenn Sie einen anderen Namen verwenden möchten, können Sie ihn in diesem `name` Element angeben. Wenn in der Antwort a angegeben `name` wird, hat dieser Wert Vorrang vor dem herkömmlichen Namen.

Fehler

Ein unbegrenztes Array von eindeutigen Nachrichten, die in der Antwort bereitgestellt werden, falls bei der Verarbeitung der Anfrage Fehler auftreten.

Einschränkungen:

- Ein Nachrichtenelement wird als JSON-Objekt mit den folgenden Feldern deklariert:
  - `code`: Eine Zeichenfolge mit alphanumerischen Zeichen und _ (Unterstrichen) mit einer Länge zwischen 1 und 64 Zeichen
  - `message`: Ein unbegrenzter Zeichenkettenwert

Example Beispiel für eine Fehlermeldung

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

Antwortcode

Ein Integer-Code, der anzeigt, wie die Anfrage behandelt wurde. Wir empfehlen, dass der Gerätecode einen Code zurückgibt, der die Spezifikation für den Statuscode der HTTP-Serverantwort verwendet, um Einheitlichkeit innerhalb des Systems zu gewährleisten.

Einschränkung: Ein ganzzahliger Wert im Bereich von 100 bis 599.

Anfrage- oder Antwortparameter

Der Parameterbereich ist als eine Zuordnung von Namens- und Subschemapaaren definiert. In den Anforderungsparametern können beliebig viele Parameter definiert werden, sofern sie in das Schemadokument passen.

Parameternamen dürfen nur alphanumerische Zeichen enthalten. Leerzeichen oder andere Zeichen sind nicht zulässig.

Parameterfeld

Die obligatorischen Elemente in a `parameter` sind `extrinsicId` und`value`. Die optionalen Elemente sind `description` und`extrinsicProperties`.

Das Beschreibungselement hat dasselbe Format wiedescription, mit einer maximalen Länge von 1024 Zeichen.

`extrinsicId`und **`extrinsicProperties`** überschreibt

die `extrinsicId` und `extrinsicProperties` folgen demselben Format wie extrinsICid (verpflichtend) undExtrinsische Eigenschaften, mit diesen zusätzlichen Details:

- Wenn in der Anfrage oder Antwort ein angegeben `extrinsicId` wird, hat dieser Wert eine höhere Priorität als der auf Aktionsebene angegebene Wert. Das System muss `extrinsicId` zuerst die request/response Ebene verwenden. Fehlt sie, muss die Aktionsebene verwendet werden `extrinsicId`
- Wenn sie in der Anfrage oder Antwort angegeben `extrinsicProperties` werden, haben diese Eigenschaften eine höhere Priorität als der Va-Wert, der auf der Aktionsebene angegeben wurde. Das System muss die Aktionsebene übernehmen `extrinsicProperties` und die auf der Ebene bereitgestellten Schlüssel-Wert-Paare ersetzen request/response `extrinsicProperties`

Example ExtrinsicID und ExtrinsicProperties überschreiben das Beispiel

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

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

Im obigen Beispiel wären die effektiven Werte für die Aktionsanforderung:

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

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

Integrierte Aktionen

Für alle Funktionen können Sie benutzerdefinierte Aktionen mit den Schlüsselwörtern `ReadState` und ausführen`UpdateState`. Diese beiden Aktionsschlüsselwörter wirken sich auf die im Datenmodell definierten Eigenschaften der Funktion aus.

ReadState

Sendet einen Befehl an die`managedThing`, um die Werte ihrer Statuseigenschaften zu lesen. Wird `ReadState` verwendet, um die Aktualisierung des Gerätestatus zu erzwingen.

UpdateState

Sendet einen Befehl zum Aktualisieren einiger Eigenschaften.

Das Erzwingen einer Gerätestatussynchronisierung kann in den folgenden Szenarien nützlich sein:

1. Das Gerät war für einen bestimmten Zeitraum offline und hat keine Ereignisse ausgelöst.
2. Das Gerät wurde gerade bereitgestellt und hat noch keinen Status, der in der Cloud verwaltet wird.
3. Der Gerätestatus stimmt nicht mit dem tatsächlichen Zustand des Geräts überein.

ReadState Beispiele

Prüfen Sie mithilfe der SendManagedThingCommandAPI, ob das Licht ein- oder ausgeschaltet ist:

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

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

Lesen Sie alle Statuseigenschaften für die `matter.OnOff` Fähigkeit:

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

```
    }
  ]
}
```

UpdateState Beispiel

Ändern Sie das `OnTime` für ein Licht mithilfe der SendManagedThingCommandAPI:

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

--Ereignisse

Ereignisse sind schemaverwaltete, unidirektionale Signale, die vom Gerät implementiert werden.

Implementieren Sie Ereignisse gemäß den folgenden Einschränkungen:

- Schließt nur eindeutige Ereignisse in das Event-Array ein
- Schließen Sie eine beliebige Anzahl von Ereignissen ein, die innerhalb der Größenbeschränkungen für Schemadokumente liegen

Ereignisse in verwalteten Integrationssystemen

Arbeiten mit Ereignissen

Eine Veranstaltung ist eine standardisierte Methode, um sich proaktiv über Änderungen an einem Gerät oder seiner Umgebung zu informieren. Es stellt ein modelliertes Ereignis dar, das das Gerät an die Cloud sendet, um Informationen über etwas bereitzustellen, das auf dem Gerät geändert oder in seiner Umgebung erkannt wurde. Da diese Ereignisse modelliert sind, können Kunden sie in einem Kontrollablauf verwenden, um auf bestimmte Ereignisse und die darin enthaltenen Details zu reagieren.

Für ein Ereignis sind die obligatorischen Elemente `name` und`extrinsicId`. Die optionalen Elemente sind `description`und`extrinsicProperties`, und`request`.

Beschreibung

Die Beschreibung folgt demselben Format wie unter beschriebendescription, mit einer maximalen Länge von 512 Zeichen.

Anforderung

Der `request` Abschnitt ist optional und kann weggelassen werden, wenn keine Anforderungsparameter vorhanden sind. Wenn dieser Parameter weggelassen wird, unterstützt das System ein Gerät, das eine Ereignisanforderung ohne Nutzlast sendet, indem es einfach den Namen des Ereignisses verwendet. Dies wird bei einfachen Ereignissen verwendet, z. B. bei einem Sensorausfall an einer Pumpe oder wenn der Alarm bei einem Rauch- oder Kohlenmonoxidmelder stummgeschaltet ist.

Die komplexen Aktionen benötigen zusätzliche Parameter. Beispielsweise kann eine Anfrage zum Streamen von Kamerabildern Parameter über das zu verwendende Streaming-Protokoll oder darüber enthalten, ob der Stream an ein bestimmtes Anzeigegerät gesendet werden soll.

Für eine Veranstaltungsanfrage ist das obligatorische Element`parameters`. Es gibt keine optionalen Elemente.

Antwort

Antworten auf Ereignisse werden derzeit nicht unterstützt.

Schema für Typdefinitionen

In den folgenden Abschnitten wird das für Typdefinitionen verwendete Schema detailliert beschrieben.

$id

Das $id-Element identifiziert die Schemadefinition. Es muss dieser Struktur folgen:

- Beginnen Sie mit dem `/schema-versions/` URI-Präfix
- Schließen Sie den `definition` Schematyp ein
- Verwenden Sie einen Schrägstrich (`/`) als URI-Pfadtrennzeichen
- Fügen Sie die Schema-Identität ein, wobei die Fragmente durch Punkte (`.`) getrennt sind
- Verwenden Sie das `@` Zeichen, um die Schema-ID und die Version zu trennen
- Beenden Sie mit der Serverversion und verwenden Sie Punkte (`.`), um Versionsfragmente zu trennen

Die Schema-Identität muss mit einem Stamm-Namespace beginnen, der 3-12 Zeichen lang ist, gefolgt von einem optionalen Unternamespace und einem Namen.

Die Semver-Version umfasst eine HAUPTVERSION (bis zu 3 Ziffern), eine NEBENVERSION (bis zu 3 Ziffern) und eine optionale PATCH-Version (bis zu 4 Ziffern).

> **Note**
>
> Sie können die reservierten `aws` Namespaces nicht verwenden oder `matter`

Example Beispiel $id

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

$ref

Das $ref-Element verweist auf eine bestehende Typdefinition innerhalb des Systems. Es folgt denselben Einschränkungen wie das `$id` Element.

> **Note**
>
> Es muss eine Typdefinition oder Fähigkeit mit dem in der `$ref` Datei angegebenen Wert vorhanden sein.

Example Beispiel $ref

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

Name

Das Namenselement ist eine Zeichenfolge, die den Entitätsnamen im Schemadokument darstellt. Es enthält häufig Abkürzungen und muss den folgenden Regeln entsprechen:

- Enthält nur alphanumerische Zeichen, Punkte (.), Schrägstriche (/), Bindestriche (-) und Leerzeichen
- Beginne mit einem Buchstaben
- Maximal 192 Zeichen

Das name-Element wird in der Benutzeroberfläche und in der Dokumentation der Amazon Web Services Services-Konsole verwendet.

Example Beispielnamen

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

Titel

Das Titelelement ist eine beschreibende Zeichenfolge für die Entität, die durch das Schemadokument repräsentiert wird. Es kann beliebige Zeichen enthalten und wird in der Dokumentation verwendet.

Example Beispieltitel

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

description

Das `description` Element bietet eine detaillierte Erläuterung der Entität, die durch das Schemadokument repräsentiert wird. Es kann beliebige Zeichen enthalten und wird in der Dokumentation verwendet.

Example Beispiel für eine Beschreibung

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

Extrinsische ID

Das `extrinsicId` Element stellt eine Kennung dar, die außerhalb des IoT-Systems von Amazon Web Services verwaltet wird. Bei Matter-Funktionen wird es je nach Kontext `clusterId` `attributeId` `commandId` `eventId`,`fieldId`,, oder zugeordnet.

Dabei `extrinsicId` kann es sich entweder um eine stringifizierte dezimale Ganzzahl (1—10 Ziffern) oder um eine stringifizierte hexadezimale Ganzzahl (0x- oder 0X-Präfix, gefolgt von 1—8 hexadezimalen Ziffern) handeln.

> **Note**
>
> Für AWS ist die Lieferanten-ID (VID) 0x1577 und für Matter 0. Das System stellt sicher, dass benutzerdefinierte Schemas diese für Funktionen reservierten VIDs Daten nicht verwenden.

Example Beispiel: ExtrinsICIDs

```
0018
0x001A
```

```
0x15771002
```

Extrinsische Eigenschaften

Das `extrinsicProperties` Element enthält eine Reihe von Eigenschaften, die in einem externen System definiert sind, aber innerhalb des Datenmodells verwaltet werden. Bei Matter-Fähigkeiten ist es verschiedenen unmodellierten oder teilweise modellierten Elementen innerhalb eines ZCL-Clusters, -Attributs, -Befehls oder -Ereignisses zugeordnet.

Extrinsische Eigenschaften müssen den folgenden Einschränkungen entsprechen:

- Eigenschaftsnamen müssen alphanumerisch ohne Leerzeichen oder Sonderzeichen sein
- Eigenschaftswerte können beliebige JSON-Schemawerte sein
- Maximal 20 Eigenschaften

Das System unterstützt verschiedene`extrinsicProperties`, darunter `accessapiMaturity`,`cli`,`cliFunctionName`, und andere. Diese Eigenschaften erleichtern Transformationen von ACL zu Datenmodellen AWS (und umgekehrt).

> **Note**
>
> Extrinsische Eigenschaften werden für die Elemente`action`, `eventproperty`, und `struct` fields einer Fähigkeit unterstützt, nicht jedoch für die Fähigkeit oder den Cluster selbst.

Systemgestützte extrinsische Eigenschaften

Das System verfolgt die folgenden unmodellierten oder teilweise modellierten Cluster-, Attribut-, Befehls- oder Ereignisattribute wie `extrinsicProperties` bei Transformationen zu oder von ZCL:

`access`

Jedes Zugriffsobjekt enthält Folgendes:

- op- Operation, die als eine `enum` mit den Werten: `readwrite`, oder modelliert wurde `invoke`
- `privilege`- Privileg, modelliert als ein `enum` mit den Werten:`view`,, `proxy_viewoperate`, oder `manage` `administer`
- `role`- Unbegrenzte Zeichenfolge, die eine Operatorrolle darstellt

`apiMaturity`

Eine unbegrenzte einfache Zeichenfolge, die den Reifegrad darstellt. Dies wird in ZCL als `enum` mit den Werten modelliert:`stable`,, oder `provisional` `internal` `deprecated`

`side`

Modelliert als eine Aufzählung mit den Werten:,, und `either` `server` `client`

Boolesche Eigenschaften

Die folgenden Eigenschaften sind boolesche Flags:

- `isFabricScoped`
- `isFabricSensitive`
- `mustUseAtomicWrite`
- `mustUseTimedInvoke`

Eigenschaften von Zeichenketten

Die folgenden Eigenschaften werden als unbegrenzte Zeichenketten dargestellt:

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

- `storage`

Überlegungen zur Transformation

Bei ZCL-Transformationen `extrinsicProperties` werden sie ohne Verarbeitung in einer Map gespeichert. Benutzerdefinierte Schemas, die Discovery verwenden, werden keiner ZCL-Transformation unterzogen. Wenn Sie jedoch in future ZCL-Transformationen für benutzerdefinierte Schemas implementieren möchten, müssen Sie alle unbegrenzten einfachen Zeichenfolgentypen modellieren `extrinsicProperties` und Einschränkungen wie Aufzählungen, Muster (Regex) und Länge definieren. Durch diese Vorbereitung wird gewährleistet, dass diese Eigenschaften während der Transformation ordnungsgemäß behandelt werden.

Im Gegensatz dazu `extrinsicProperties` sind Transformationen für AWS zwei Konnektoren überhaupt nicht enthalten, da diese Details im Konnektorformat nicht erforderlich sind.

Erstellung und Verwendung von Typdefinitionen in Capability-Schema-Dokumenten

Alle Elemente in den Schemas werden in Typdefinitionen aufgelöst. Bei diesen Typdefinitionen handelt es sich entweder um primitive Typdefinitionen (wie Boolesche Werte, Zeichenketten, Zahlen) oder um Typdefinitionen mit Namensräumen (Typdefinitionen, die der Einfachheit halber aus primitiven Typdefinitionen erstellt wurden).

Wenn Sie ein benutzerdefiniertes Schema definieren, können Sie sowohl primitive Definitionen als auch Namespace-Typdefinitionen verwenden.

Inhalt

- matter-Typen
- aws-Typen
  - Definition des Bitmap-Typs
  - Definition des Aufzählungstyps

Primitive Typdefinitionen

Primitive Typdefinitionen sind die Bausteine für alle Typdefinitionen, die in verwalteten Integrationen definiert sind. Alle Namespace-Definitionen, einschließlich benutzerdefinierter Typdefinitionen, werden entweder über das `$ref` Schlüsselwort oder das Schlüsselwort zu einer primitiven Typdefinition aufgelöst. `type`

Alle primitiven Typen können mithilfe des `nullable` Schlüsselworts auf Null gesetzt werden, und Sie können alle primitiven Typen mithilfe des Schlüsselworts identifizieren. `type`

Boolesche Werte

Boolesche Typen unterstützen Standardwerte.

Beispieldefinition:

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

Unterstützung für Integer-Typen

Integer-Typen unterstützen Folgendes:

- `default`-Werte
- `maximum`-Werte
- `minimum`-Werte
- `exclusiveMaximum`-Werte
- `exclusiveMinimum`-Werte

- `multipleOf`-Werte

Wenn `x` der Wert validiert wird, muss Folgendes zutreffen:

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

> **Note**
>
> Zahlen mit einem Bruchteil von Null werden als ganze Zahlen betrachtet, Fließkommazahlen werden jedoch zurückgewiesen.
>
> ```
> 1.0 // Schema-Compliant
> 3.1415926 // NOT Schema-Compliant
> ```

Sie können zwar sowohl als auch `exclusiveMinimum` oder beide `minimum` `maximum` und angeben`exclusiveMaximum`, wir empfehlen jedoch nicht, beide gleichzeitig zu verwenden.

Beispieldefinitionen:

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

Alternative Definition:

```
{
    "type" : "integer",
    "default" : 2,
    "nullable" : true,
    "exclusiveMaximum" : 11,
```

```
    "exclusiveMinimum" : -1,
    "multipleOf": 2
}
```

Zahlen

Verwenden Sie den Zahlentyp für jeden numerischen Typ, einschließlich Ganzzahlen und Fließkommazahlen.

Zahlentypen unterstützen Folgendes:

- `default`-Werte
- `maximum`-Werte
- `minimum`-Werte
- `exclusiveMaximum`-Werte
- `exclusiveMinimum`-Werte
- `multipleOf`Werte. Das Vielfache kann eine Fließkommazahl sein.

Wenn `x` der Wert validiert wird, muss Folgendes zutreffen:

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

Sie können zwar sowohl als auch `minimum` `exclusiveMinimum` oder beide `maximum` und angeben`exclusiveMaximum`, wir empfehlen jedoch nicht, beide gleichzeitig zu verwenden.

Beispieldefinitionen:

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

Alternative Definition:

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

Zeichenfolgen

String-Typen unterstützen Folgendes:

- `default`-Werte
- Längenbeschränkungen (müssen nicht negative Zahlen sein), einschließlich Werte für `maxLength` und `minLength`
- `pattern`Werte für reguläre Ausdrücke

Wenn Sie reguläre Ausdrücke definieren, ist die Zeichenfolge gültig, wenn der Ausdruck mit einer beliebigen Stelle in der Zeichenfolge übereinstimmt. Der reguläre Ausdruck p entspricht beispielsweise jeder Zeichenfolge, die ein p enthält, z. B. „apple“, und nicht nur der Zeichenfolge „p“. Aus Gründen der Übersichtlichkeit empfehlen wir, reguläre Ausdrücke mit `^...$` (z. B.^p$) zu umgeben, es sei denn, Sie haben einen bestimmten Grund, dies nicht zu tun.

Beispieldefinition:

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

Null-Werte

Nulltypen akzeptieren nur einen einzigen Wert:`null`.

Beispieldefinition:

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

Arrays

Array-Typen unterstützen Folgendes:

- `default`— eine Liste, die als Standardwert verwendet wird.
- `items`— Die JSON-Typdefinition, die allen Array-Elementen auferlegt wird.
- Längenbeschränkungen (muss eine nicht negative Zahl sein)
  - `minItems`
  - `maxItems`
- `pattern`Werte für Regex
- `uniqueItems`— ein boolescher Wert, der angibt, ob die Elemente im Array eindeutig sein müssen
- `prefixItems`— ein Array, bei dem jedes Element ein Schema ist, das jedem Index des Arrays des Dokuments entspricht. Das heißt, ein Array, bei dem das erste Element das erste Element des Eingabe-Arrays validiert, das zweite Element das zweite Element des Eingabe-Arrays und so weiter.

Beispieldefinition:

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

Beispiele für die Array-Validierung:

```
//Examples:
```

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

Alternative Definition mit Tupelvalidierung:

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

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

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

Objekte

Objekttypen unterstützen Folgendes:

- Eigentumsbeschränkungen
  - `properties`— Definieren Sie die Eigenschaften (Schlüssel-Wert-Paare) eines Objekts mithilfe des Schlüsselwortsproperties. Der Wert von `properties` ist ein Objekt, wobei jeder Schlüssel der Name einer Eigenschaft ist und jeder Wert ein Schema ist, das zur Validierung dieser Eigenschaft verwendet wird. Jede Eigenschaft, die mit keinem der Eigenschaftsnamen im `properties` Schlüsselwort übereinstimmt, wird von diesem Schlüsselwort ignoriert.
  - `required`— Standardmäßig sind die durch das `properties` Schlüsselwort definierten Eigenschaften nicht erforderlich. Sie können jedoch mithilfe des `required` Schlüsselworts eine Liste der erforderlichen Eigenschaften bereitstellen. Das `required` Schlüsselwort benötigt ein Array von null oder mehr Zeichenketten. Jede dieser Zeichenketten muss einzigartig sein.
  - `propertyNames`— Dieses Schlüsselwort ermöglicht die Kontrolle über das RegEx Muster für Eigenschaftsnamen. Sie könnten beispielsweise erzwingen, dass alle Eigenschaften eines Objekts Namen haben, die einer bestimmten Konvention folgen.

- `patternProperties`— Dadurch werden reguläre Ausdrücke Schemas zugeordnet. Wenn ein Eigenschaftsname mit dem angegebenen regulären Ausdruck übereinstimmt, muss der Eigenschaftswert anhand des entsprechenden Schemas validiert werden. Verwenden Sie dies beispielsweise, `patternProperties` um anzugeben, dass ein Wert einem bestimmten Schema entsprechen soll, wenn ein bestimmter Eigenschaftsname gegeben ist.
- `additionalProperties`— Dieses Schlüsselwort steuert, wie zusätzliche Eigenschaften behandelt werden. Zusätzliche Eigenschaften sind Eigenschaften, deren Namen nicht im Schlüsselwort properties aufgeführt sind oder die mit einem der regulären Ausdrücke in übereinstimmen`patternProperties`. Standardmäßig sind zusätzliche Eigenschaften zulässig. Wenn Sie dieses Feld auf setzen, `false` bedeutet dies, dass keine zusätzlichen Eigenschaften zulässig sind.
- `unevaluatedProperties`— Dieses Schlüsselwort ist ähnlich, `additionalProperties` außer dass es Eigenschaften erkennen kann, die in Unterschemas deklariert sind. `unevaluatedProperties`sammelt alle Eigenschaften, die bei der Verarbeitung der Schemas erfolgreich validiert wurden, und verwendet diese als zulässige Eigenschaftsliste. Auf diese Weise können Sie komplexere Aufgaben ausführen, z. B. das bedingte Hinzufügen von Eigenschaften. Weitere Informationen finden Sie im folgenden Beispiel.

- `anyOf`— Der Wert dieses Schlüsselworts MUSS ein nicht leeres Array sein. Jedes Element des Arrays MUSS ein gültiges JSON-Schema sein. Eine Instanz validiert erfolgreich anhand dieses Schlüsselworts, wenn sie erfolgreich anhand mindestens eines Schemas validiert wird, das durch den Wert dieses Schlüsselworts definiert ist.
- `oneOf`— Der Wert dieses Schlüsselworts MUSS ein nicht leeres Array sein. Jedes Element des Arrays MUSS ein gültiges JSON-Schema sein. Eine Instanz validiert erfolgreich anhand dieses Schlüsselworts, wenn sie erfolgreich anhand genau eines Schemas validiert wird, das durch den Wert dieses Schlüsselworts definiert ist.

Beispiel für erforderlich:

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

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

```
// NOT Schema Compliant
{}
```

PropertyNames Beispiel:

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

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

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

PatternProperties Beispiel:

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

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

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

AdditionalProperties Beispiel:

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

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

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

UnevaluatedProperties Beispiel:

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

// Schema Compliant
{
  "standard_field": "some value",
  "@id": 123,
  "@timestamp": 1678886400
}
// This passes because "standard_field" is evaluated by properties,
// "@id" and "@timestamp" are evaluated by patternProperties,
```

```
// and no other properties remain unevaluated.

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

AnyOf Beispiel:

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

// Schema Compliant
"short"
12

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

OneOf Beispiel:

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

// Schema Compliant
10
9

// NOT Schema compliant
2  // Not a multiple of either 5 or 3
```

```
15 // Multiple of both 5 and 3 is rejected.
```

Typdefinitionen mit Namespaces

Typdefinitionen mit Namespaces sind Typen, die aus primitiven Typen aufgebaut sind. Diese Typen müssen dem Format folgen. *`namespace`*`.`*`typename`*`.` Managed Integrations stellt vordefinierte Typen unter den Namespaces und zur Verfügung. `aws` `matter` Sie können jeden Namespace für benutzerdefinierte Typen verwenden, mit Ausnahme der reservierten Namespaces und der Namespaces. `aws` `matter`

Um nach verfügbaren Typdefinitionen mit Namespaces zu suchen, verwenden Sie die API, bei der der ListSchemaVersionsFilter auf eingestellt ist. `Type` `definition`

`matter`-Typen

Suchen Sie mithilfe der ListSchemaVersionsAPI nach Datentypen im `matter` Namespace, wobei der Filter auf `matter` und der `Namespace` `Type` Filter auf eingestellt ist. `definition`

`aws`-Typen

Suchen Sie mithilfe der ListSchemaVersionsAPI nach Datentypen im `aws` Namespace, wobei der `Namespace` Filter auf `aws` und der `Type` Filter auf eingestellt ist. `definition`

Definition des Bitmap-Typs

Bitmaps haben zwei erforderliche Eigenschaften:

- `type`muss ein Objekt sein
- `properties`muss ein Objekt sein, das jede Bitdefinition enthält. Jedes Bit ist ein Objekt mit Eigenschaften `extrinsicId` und`value`. Der Wert jedes Bits muss eine Ganzzahl mit einem Mindestwert von 0 und einem Höchstwert von mindestens 1 sein.

Beispiel für eine Bitmap-Definition:

```
{
    "title" : "Sample Bitmap Type",
    "description" : "Type definition for SampleBitmap.",
    "$ref" : "/schema-versions/definition/aws.bitmap@1.0 ",
    "type" : "object",
```

```
    "additionalProperties" : false,
    "properties" : {
        "Bit1" : {
            "extrinsicId" : "0x0000",
            "value" : {
                "type" : "integer",
                "maximum" : 1,
                "minimum" : 0
            }
        },
        "Bit2" : {
            "extrinsicId" : "0x0001",
            "value" : {
                "type" : "integer",
                "maximum" : 1,
                "minimum" : 0
            }
        }
    }
}

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

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

Definition des Aufzählungstyps

Aufzählungen benötigen drei Eigenschaften:

- `type`muss ein Objekt sein
- `enum`muss ein Array von eindeutigen Zeichenketten mit mindestens einem Element sein
- `extrinsicIdMap`ist ein Objekt mit Eigenschaften, bei denen es sich um Enum-Werte handelt. Der Wert jeder der Eigenschaften sollte der extrinsische Bezeichner sein, der dem Enum-Wert entspricht.

Beispiel für eine Enum-Definition:

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

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

// NOT Schema Compliant
"NotAnEnumValue"
```

Befehle und Ereignisse für IoT-Geräte verwalten

Gerätebefehle bieten die Möglichkeit, ein physisches Gerät aus der Ferne zu verwalten und so die vollständige Kontrolle über das Gerät zu gewährleisten. Außerdem können wichtige Sicherheits-, Software- und Hardwareupdates durchgeführt werden. Bei einer großen Geräteflotte erhalten Sie einen Überblick über Ihre gesamte Geräteimplementierung, wenn Sie wissen, wann ein Gerät einen Befehl ausführt. Ein Gerätebefehl oder ein automatisches Update löst eine Änderung des Gerätestatus aus, wodurch wiederum ein neues Geräteereignis ausgelöst wird. Dieses Geräteereignis löst eine Benachrichtigung aus, die automatisch an ein vom Kunden verwaltetes Ziel gesendet wird.

Themen

Gerätebefehle

Eine Befehlsanforderung ist der Befehl, der an das Gerät gesendet wird. Eine Befehlsanforderung enthält eine Nutzlast, die die auszuführende Aktion spezifiziert, z. B. das Einschalten der Glühbirne. Um einen Gerätebefehl zu senden, wird die `SendManagedThingCommand` API im Namen des Endbenutzers von verwalteten Integrationen aufgerufen und die Befehlsanforderung wird an das Gerät gesendet.

Die Antwort auf a `SendManagedThingCommand` ist ein, `traceId` und Sie können `traceId` damit die Befehlsübermittlung und alle damit verbundenen Befehlsantwort-Workflows verfolgen, wo immer dies möglich ist.

Weitere Informationen zum `SendManagedThingCommand` API-Vorgang finden Sie unter SendManagedThingCommand.

Aktion **`UpdateState`**

Um den Status eines Geräts zu aktualisieren, z. B. die Uhrzeit, zu der ein Licht aufleuchtet, verwenden Sie die `UpdateState` Aktion beim Aufrufen der `SendManagedThingCommand` API. Geben Sie die Datenmodelleigenschaft und den neuen Wert an, den Sie aktualisieren möchten`parameters`. Das folgende Beispiel zeigt eine `SendManagedThingCommand` API-Anfrage, auf `OnTime` die eine Glühbirne aktualisiert wird`5`.

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

Aktion **`ReadState`**

Um den neuesten Status eines Geräts einschließlich der aktuellen Werte aller Datenmodelleigenschaften abzurufen, verwenden Sie die `ReadState` Aktion beim Aufrufen der `SendManagedThingCommand` API. `propertiesToRead`In können Sie die folgenden Optionen verwenden:

- Geben Sie eine bestimmte Datenmodelleigenschaft an, um den aktuellen Wert abzurufen, z. B. um `OnOff` festzustellen, ob ein Licht an oder aus ist.
- Verwenden Sie den Platzhalteroperator (*), um alle Gerätestatuseigenschaften für eine Funktion zu lesen.

Die folgenden Beispiele veranschaulichen beide Szenarien für eine `SendManagedThingCommand` API-Anfrage, bei der die `ReadState` Aktion verwendet wird:

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

Geräteereignisse

Ein Geräteereignis beinhaltet den aktuellen Status des Geräts. Dies kann bedeuten, dass das Gerät seinen Status geändert hat oder seinen Status meldet, auch wenn sich der Status nicht geändert hat. Dazu gehören Eigenschaftsberichte und Ereignisse, die im Datenmodell definiert sind. Ein Ereignis könnte sein, dass ein Waschmaschinenzyklus abgeschlossen wurde oder der Thermostat die vom Endbenutzer eingestellte Zieltemperatur erreicht hat.

Benachrichtigungen über Geräteereignisse

Ein Endbenutzer kann bestimmte, vom Kunden verwaltete Ziele abonnieren, die er für Updates zu bestimmten Geräteereignissen erstellt. Rufen Sie die API auf, um ein vom Kunden verwaltetes Ziel zu erstellen. `CreateDestination` Wenn das Gerät ein Geräteereignis an verwaltete Integrationen meldet, wird das vom Kunden verwaltete Ziel benachrichtigt, falls eines existiert.

Taggen Ihrer Ressourcen für verwaltete Integrationen

Um Ihnen bei der Verwaltung und Organisation Ihrer Ressourcen zu helfen, können Sie optional jeder dieser Ressourcen Ihre eigenen Metadaten in Form von Tags zuweisen. In diesem Abschnitt werden Tags und deren Erstellung beschrieben.

Grundlagen zu Tags (Markierungen)

Sie können Tags verwenden, um Ihre verwalteten Integrationsressourcen auf unterschiedliche Weise zu kategorisieren (z. B. nach Zweck, Eigentümer oder Umgebung). Dies ist nützlich, wenn Sie viele Ressourcen desselben Typs haben. In diesem Fall können Sie schnell bestimmte Ressourcen basierend auf den zugewiesenen Tags bestimmen. Jedes Tag besteht aus einem Schlüssel und einem optionalen Wert, die Sie beide selbst definieren können. Sie können zum Beispiel eine Reihe von Tags für Ihre Objekttypen definieren, die Ihnen helfen, Ihre Geräte nach Typ nachzuverfolgen. Wir empfehlen die Erstellung von Tag-Schlüsseln, die die Anforderungen der jeweiligen Ressourcenart erfüllen. Die Verwendung einheitlicher Tag-Schlüssel vereinfacht das Verwalten der -Ressourcen.

Sie können die Ressourcen auf Grundlage der hinzugefügten oder angewendeten Tags durchsuchen und filtern. Sie können auch Tags verwenden, um den Zugriff auf Ihre Ressourcen zu steuern (in Verwenden von Tags mit IAM-Richtlinien beschrieben).

Aus Gründen der Benutzerfreundlichkeit bietet der Tag-Editor in der AWS Management Console eine zentrale, einheitliche Möglichkeit, Ihre Tags zu erstellen und zu verwalten. Weitere Informationen finden Sie unter Arbeiten mit dem Tag-Editor in Arbeiten mit der AWS Management Console.

Sie können auch mithilfe der API AWS CLI und der verwalteten Integrationen mit Tags arbeiten. Sie können Tags verwalteten Dingen, Bereitstellungsprofilen, Anmeldeinformationssperren und over-the-air (OTA-) Aufgaben zuordnen, wenn Sie sie erstellen, indem Sie das `Tags` Feld in den folgenden Befehlen verwenden:

- CreateManagedThing
- CreateProvisioningProfile
- CreateCredentialLocker
- CreateOtaTask
- CreateAccountAssociation

Sie können Tags für vorhandene Ressourcen, die das Markieren unterstützen, hinzufügen, ändern oder löschen. Verwenden Sie dazu die folgenden Befehle:

- TagResource
- ListTagsForResource
- UntagResource

Sie können Tag (Markierung)-Schlüssel und -Werte bearbeiten und Tags (Markierungen) jederzeit von einer Ressource entfernen. Sie können den Wert eines Tags (Markierung) zwar auf eine leere Zeichenfolge, jedoch nicht null festlegen. Wenn Sie ein Tag (Markierung) mit demselben Schlüssel wie ein vorhandener Tag (Markierung) für die Ressource hinzufügen, wird der alte Wert mit dem neuen überschrieben. Wenn Sie eine Ressource löschen, werden alle der Ressource zugeordneten Tags ebenfalls gelöscht.

Tag-Beschränkungen und -Einschränkungen

Die folgenden grundlegenden Einschränkungen gelten für Tags (Markierungen):

- Maximale Anzahl von Tags pro Ressource: 50
- Maximale Schlüssellänge: 127 Unicode-Zeichen in UTF-8
- Maximale Wertlänge: 255 Unicode-Zeichen in UTF-8
- Bei Tag-Schlüsseln und -Werten muss die Groß- und Kleinschreibung beachtet werden.
- Verwenden Sie nicht das Präfix `aws:` in Ihren Tag-Namen oder -Werten. Es ist für die Verwendung reserviert. AWS Sie können keine Tag-Namen oder Werte mit diesem Präfix bearbeiten oder löschen. Tags mit diesem Präfix werden nicht zum Limit für Tags pro Ressource gezählt.
- Wenn Ihr Markierungsschema für mehrere Services und Ressourcen verwendet wird, denken Sie daran, dass die zulässigen Zeichen bei anderen Services möglicherweise eingeschränkt sind. Zulässige Zeichen: Buchstaben, Leerzeichen und Zahlen, die in UTF-8 darstellbar sind, sowie die folgenden Sonderzeichen: + - = . _ : / @.

Verwenden von Tags mit IAM-Richtlinien

Sie können tagbasierte Berechtigungen auf Ressourcenebene in den IAM-Richtlinien anwenden, die Sie für API-Aktionen mit verwalteten Integrationen verwenden. Dies ermöglicht Ihnen eine bessere Kontrolle darüber, welche Ressourcen ein Benutzer erstellen, ändern oder verwenden kann. Sie

können das `Condition`-Element (auch als `Condition`-Block bezeichnet) mit den folgenden Bedingungskontextschlüsseln und Werten in einer IAM-Richtlinie zum Steuern des Benutzerzugriffs (Berechtigungen) basierend auf den Tags einer Ressource verwenden:

- Verwenden Sie `aws:ResourceTag/`*`tag-key`*`: `*`tag-value`*, um Benutzeraktionen für Ressourcen mit bestimmten Tags zuzulassen oder zu verweigern.
- Verwenden Sie `aws:RequestTag/`*`tag-key`*`: `*`tag-value`*, um festzulegen, dass ein bestimmtes Tag verwendet (oder nicht verwendet) wird, wenn Sie eine API-Anfrage stellen, um eine Ressource zu erstellen oder zu ändern, die Tags zulässt.
- Verwenden Sie `aws:TagKeys: [`*`tag-key`*`, ...]`, um zu verlangen, dass ein bestimmter Satz von Tag-Schlüsseln verwendet wird (oder nicht), wenn eine API-Anforderung zum Erstellen einer Ressource durchgeführt wird, die Tags zulässt.

> **Note**
>
> Die Bedingungskontextschlüssel und -werte in einer IAM-Richtlinie gelten nur für verwaltete Integrationsaktionen, bei denen eine Kennung für eine Ressource, die markiert werden kann, ein erforderlicher Parameter ist. Beispielsweise GetCustomEndpointist die Verwendung von auf der Grundlage von Zustandskontext-Schlüsseln und -Werten nicht zulässig oder verweigert, weil in dieser Anfrage auf keine markierbare Ressource (verwaltete Dinge, Bereitstellungsprofile, Anmeldeinformationssperren, over-the-air Aufgaben) verwiesen wird. Weitere Informationen zu Ressourcen für verwaltete Integrationen, die taggbar sind, und zu den Bedingungsschlüsseln, die sie unterstützen, finden Sie unter Funktionen für Aktionen, Ressourcen und Bedingungsschlüssel für verwaltete Integrationen von. AWS IoT AWS IoT Device Management

Weitere Informationen finden Sie unter Zugriffssteuerung mit Tags im AWS Identity and Access Management Benutzerhandbuch. Der Abschnitt IAM-JSON-Richtlinienreferenz dieses Handbuchs enthält die detaillierte Syntax sowie Beschreibungen und Beispiele für Elemente, Variablen und die Auswertungslogik von JSON-Richtlinien in IAM.

Die folgende Beispielrichtlinie wendet zwei auf Tags basierende Einschränkungen für die Aktion an. `CreateManagedThing` Ein von dieser Richtlinie eingeschränkter IAM-Benutzer:

- Es kann kein verwaltetes Objekt mit dem Tag „env=prod“ erstellt werden (im Beispiel siehe Zeile). `"aws:RequestTag/env" : "prod"`

- Ein verwaltetes Ding, das ein vorhandenes Tag „env=prod“ hat, kann nicht geändert oder darauf zugegriffen werden (im Beispiel siehe Zeile). `"aws:ResourceTag/env" : "prod"`

JSON

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

```
      ],
      "Resource": "*"
    }
  ]
}
```

Sie können auch mehrere Tag-Werte für einen bestimmten Tag-Schlüssel angeben, indem Sie sie wie folgt in einer Liste angeben:

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

> **Note**
>
> Wenn Sie Benutzern den Zugriff zu Ressourcen auf der Grundlage von Tags (Markierungen) gewähren oder verweigern, müssen Sie daran denken, Benutzern explizit das Hinzufügen und Entfernen dieser Tags (Markierungen) von den jeweiligen Ressourcen unmöglich zu machen. Andernfalls können Benutzer möglicherweise Ihre Einschränkungen umgehen und sich Zugriff auf eine Ressource verschaffen, indem sie ihre Tags (Markierungen) modifizieren.

Benachrichtigungen über verwaltete Integrationen

Benachrichtigungen über verwaltete Integrationen liefern Updates und wichtige Erkenntnisse aus Geräten. Zu den Benachrichtigungen gehören Connector-Ereignisse, Gerätebefehle, Lebenszyklusereignisse, OTA-Updates (Over-the-Air) und Fehlerberichte. Diese Erkenntnisse liefern verwertbare Informationen, um automatisierte Workflows zu erstellen, sofortige Maßnahmen zu ergreifen oder Ereignisdaten zur Fehlerbehebung zu speichern.

Derzeit werden nur Amazon Kinesis Kinesis-Datenstreams als Ziel für Benachrichtigungen über verwaltete Integrationen unterstützt. Sie müssen zunächst einen Amazon Kinesis Kinesis-Datenstream einrichten und verwalteten Integrationen Zugriff auf den Datenstream gewähren, bevor Sie Benachrichtigungen einrichten können.

Amazon Kinesis für Benachrichtigungen einrichten

Schritte zur Einrichtung von Amazon Kinesis

- Schritt 1: Einen Amazon Kinesis Kinesis-Datenstream erstellen
- Schritt 2: Erstellen Sie eine Berechtigungsrichtlinie
- Schritt 3: Navigieren Sie zum IAM-Dashboard und wählen Sie Rollen
- Schritt 4: Verwenden Sie eine benutzerdefinierte Vertrauensrichtlinie
- Schritt 5: Wenden Sie Ihre Berechtigungsrichtlinie an
- Schritt 6: Geben Sie einen Rollennamen ein

Gehen Sie wie folgt vor, um Amazon Kinesis für Benachrichtigungen über verwaltete Integrationen einzurichten:

Schritt 1: Einen Amazon Kinesis Kinesis-Datenstream erstellen

Ein Amazon Kinesis Data Stream kann eine große Datenmenge in Echtzeit aufnehmen, die Daten dauerhaft speichern und sie für Anwendungen verfügbar machen.

So erstellen Sie einen Amazon Kinesis Kinesis-Datenstream

- Um einen Kinesis-Datenstream zu erstellen, folgen Sie den unter Kinesis-Datenstreams erstellen und verwalten beschriebenen Schritte.

Schritt 2: Erstellen Sie eine Berechtigungsrichtlinie

Erstellen Sie eine Berechtigungsrichtlinie, die verwalteten Integrationen den Zugriff auf Ihren Kinesis-Datenstrom ermöglicht.

Um eine Berechtigungsrichtlinie zu erstellen

- Um eine Berechtigungsrichtlinie zu erstellen, kopieren Sie die unten stehende Richtlinie und folgen Sie den unter Richtlinien mithilfe des JSON-Editors erstellen beschriebenen Schritte

  JSON

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

Schritt 3: Navigieren Sie zum IAM-Dashboard und wählen Sie Rollen

Öffnen Sie das IAM-Dashboard und klicken Sie auf Rollen.

Um zum IAM-Dashboard zu navigieren

- Öffnen Sie das IAM-Dashboard und klicken Sie auf Rollen.

  Weitere Informationen finden Sie unter IAM-Rollenerstellung im AWS Identity and Access ManagementBenutzerhandbuch.

Schritt 4: Verwenden Sie eine benutzerdefinierte Vertrauensrichtlinie

Sie können eine benutzerdefinierte Vertrauensrichtlinie verwenden, um verwalteten Integrationen Zugriff auf den Kinesis-Datenstrom zu gewähren.

Um eine benutzerdefinierte Vertrauensrichtlinie zu verwenden

- Erstellen Sie eine neue Rolle und wählen Sie Benutzerdefinierte Vertrauensrichtlinie. Klicken Sie auf Weiter.

  Die folgende Richtlinie ermöglicht es verwalteten Integrationen, diese Rolle zu übernehmen, und die `Condition` Erklärung trägt dazu bei, verwirrende Probleme mit Stellvertretern zu vermeiden.

  JSON

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

Schritt 5: Wenden Sie Ihre Berechtigungsrichtlinie an

Fügen Sie der Rolle die in Schritt 2 erstellte Berechtigungsrichtlinie hinzu.

Um eine Berechtigungsrichtlinie hinzuzufügen

- Suchen Sie auf der Seite „Berechtigungen hinzufügen“ nach der Berechtigungsrichtlinie, die Sie in Schritt 2 erstellt haben, und fügen Sie sie hinzu. Klicken Sie auf Weiter.

Schritt 6: Geben Sie einen Rollennamen ein

- Geben Sie einen Rollennamen ein und klicken Sie auf Rolle erstellen.

Richten Sie Benachrichtigungen für verwaltete Integrationen ein

Schritte zur Einrichtung von Benachrichtigungen

- Schritt 1: Erteilen Sie Benutzern die Erlaubnis, die CreateDestination API aufzurufen
- Schritt 2: Rufen Sie die API auf CreateDestination
- Schritt 3: Rufen Sie die API auf CreateNotificationConfiguration

Gehen Sie wie folgt vor, um Benachrichtigungen für verwaltete Integrationen einzurichten:

Schritt 1: Erteilen Sie Benutzern die Erlaubnis, die CreateDestination API aufzurufen

- Erteilen Sie Benutzerberechtigungen zum Aufrufen der **`CreateDestination`** API

  Die folgende Richtlinie definiert die Anforderungen, unter denen der Benutzer die CreateDestinationAPI aufrufen kann.

  Informationen zum Erlangen von Passrolle-Berechtigungen für verwaltete Integrationen finden Sie unter Gewähren von AWS Identity and Access ManagementBenutzerberechtigungen zur Übergabe einer Rolle an einen AWS Dienst im Benutzerhandbuch.

  JSON

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

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

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

Schritt 2: Rufen Sie die API auf CreateDestination

- Rufen Sie die **`CreateDestination`** API auf

  Nachdem Sie Ihren Amazon Kinesis Kinesis-Datenstream und Ihre Stream-Zugriffsrolle erstellt haben, rufen Sie die CreateDestinationAPI auf, um Ihr Benachrichtigungsziel zu erstellen, an das die Benachrichtigungen weitergeleitet werden. Verwenden Sie für den `DeliveryDestinationArn` Parameter den `arn` aus Ihrem neuen Amazon Kinesis Kinesis-Datenstream.

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

  > **Note**
  >
  > `ClientToken`ist ein Idempotenz-Token. Wenn Sie eine Anfrage, die zunächst erfolgreich abgeschlossen wurde, mit demselben Client-Token und denselben Parametern erneut versuchen, ist der Wiederholungsversuch erfolgreich, ohne dass weitere Aktionen ausgeführt werden.

Schritt 3: Rufen Sie die API auf CreateNotificationConfiguration

- Rufen Sie die **`CreateNotificationConfiguration`** API auf

  Verwenden Sie abschließend die CreateNotificationConfigurationAPI, um die Benachrichtigungskonfiguration zu erstellen, die die ausgewählten Ereignistypen an Ihr Ziel weiterleitet, das durch den Kinesis-Datenstrom dargestellt wird. Verwenden Sie im `DestinationName` Parameter denselben Zielnamen wie beim ersten Aufruf der `CreateDestination` API.

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

Ereignistypen, die mit verwalteten Integrationen überwacht werden

Die folgenden Ereignistypen werden mit Benachrichtigungen über verwaltete Integrationen überwacht:

- `DEVICE_COMMAND`
  - Der Status des SendManagedThingCommandAPI-Befehls. Gültige Werte sind entweder `succeeded` oder `failed`.

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

```
                "traceId":"1234567890abcdef0",
                "receivedAt":"2017-12-22T18:43:48Z",
                "executedAt":"2017-12-22T18:43:48Z",
                "result":"failed"
              }
}
```

- `DEVICE_COMMAND_REQUEST`
  - Die Befehlsanforderung von Web Real-Time Communication (WebRTC).

    Der WebRTC-Standard ermöglicht die Kommunikation zwischen zwei Peers. Diese Peers können Video-, Audio- und beliebige Daten in Echtzeit übertragen. Managed Integrations unterstützt WebRTC, um diese Arten von Streaming zwischen einer mobilen Kundenanwendung und dem Gerät eines Endbenutzers zu ermöglichen. Weitere Informationen zum WebRTC-Standard finden Sie unter WebRTC.

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

- `DEVICE_DISCOVERY_STATUS`
  - Der Erkennungsstatus des Geräts.

```
{
      "version":"0",
      "messageId":"6a7e8feb-b491-4cf7-a9f1-bf3703467718",
      "messageType":"DEVICE_DISCOVERY_STATUS",
      "source":"aws.iotmanagedintegrations",
      "customerAccountId":"123456789012",
      "timestamp":"2017-12-22T18:43:48Z",
      "region":"ca-central-1",
      "resources":[
        "arn:aws:iotmanagedintegrations:ca-central-1:123456789012:managed-
thing/6a7e8feb-b491-4cf7-a9f1-bf3703467718"
      ],
      "payload":{
        "deviceCount": 1,
        "deviceDiscoveryId": "123",
        "status": "SUCCEEDED"
      }
}
```

- DEVICE_EVENT
  - Eine Benachrichtigung über das Eintreten eines Geräteereignisses.

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

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

- DEVICE_LIFE_CYCLE
  - Der Status des Gerätelebenszyklus.

```
{
      "version": "1.0.0",
      "messageId": "8d1e311a473f44f89d821531a0907b05",
      "messageType": "DEVICE_LIFE_CYCLE",
      "source": "aws.iotmanagedintegrations",
      "customerAccountId": "123456789012",
      "timestamp": "2024-11-14T19:55:57.568284645Z",
      "region": "ca-central-1",
      "resources": [
        "arn:aws:iotmanagedintegrations:ca-central-1:123456789012:managed-thing/
d5c280b423a042f3933eed09cf408657"
        ],
      "payload": {
        "deviceDetails": {
          "id": "d5c280b423a042f3933eed09cf408657",
          "arn": "arn:aws:iotmanagedintegrations:ca-central-1:123456789012:managed-
thing/d5c280b423a042f3933eed09cf408657",
          "createdAt": "2024-11-14T19:55:57.515841147Z",
          "updatedAt": "2024-11-14T19:55:57.515841559Z"
        },
        "status": "UNCLAIMED"
      }
}
```

- DEVICE_OTA
  - Eine OTA-Benachrichtigung für ein Gerät.
- DEVICE_STATE
  - Eine Benachrichtigung, wenn der Status eines Geräts aktualisiert wurde.

```
{
      "messageType": "DEVICE_STATE",
      "source": "aws.iotmanagedintegrations",
      "customerAccountId": "123456789012",
```

```
      "timestamp": "1731623291671",
      "resources": [
        "arn:aws:iotmanagedintegrations:ca-central-1:123456789012:managed-
thing/61889008880012345678"
      ],
      "payload": {
        "addedStates": {
          "endpoints": [{
            "endpointId": "nonEndpointId",
            "capabilities": [{
              "id": "aws.OnOff",
              "name": "On/Off",
              "version": "1.0",
              "properties": [{
                "name": "OnOff",
                "value": {
                  "propertyValue": "\"onoff\"",
                  "lastChangedAt": "2024-06-11T01:38:09.000414Z"
                }
              }
            ]}
          ]}
        ]}
      }
}
```

Cloud-to-Cloud (C2C) -Anschlüsse

Ein cloud-to-cloud Connector ermöglicht es Ihnen, eine bidirektionale Kommunikation zwischen Geräten von Drittanbietern und zu ermöglichen. AWS

Themen

Was ist ein cloud-to-cloud (C2C) -Anschluss?

Ein cloud-to-cloud Connector ist ein vorgefertigtes Softwarepaket, das ihn sicher mit AWS Cloud dem Endpunkt eines Cloud-Drittanbieters verbindet. Mithilfe des C2C-Connectors können Lösungsanbieter verwaltete Integrationen für AWS IoT Device Management nutzen, um Geräte zu steuern, die mit Clouds von Drittanbietern verbunden sind.

Verwaltete Integrationen umfassen einen Katalog von Konnektoren, über den AWS Kunden Konnektoren anzeigen und auswählen können, in die sie integrieren möchten. Weitere Informationen finden Sie unter Was ist der C2C-Steckverbinderkatalog?.

Bei verwalteten Integrationen muss jeder Konnektor als Funktion implementiert werden. AWS Lambda

Was ist der C2C-Steckverbinderkatalog?

Der Connector-Katalog für verwaltete Integrationen für AWS IoT Device Management ist eine Sammlung von C2C-Konnektoren, die die bidirektionale Kommunikation zwischen verwalteten Integrationen für AWS IoT Device Management und einem Cloud-Drittanbieter ermöglichen. Sie können sich die Konnektoren im oder im ansehen. AWS-Managementkonsole AWS CLI

Um die Konsole zum Anzeigen des Connector-Katalogs für verwaltete Integrationen zu verwenden

1. Öffnen Sie die Konsole für verwaltete Integrationen
2. Wählen Sie im linken Navigationsbereich Managed Integrations aus
3. Wählen Sie im linken Navigationsbereich der Konsole für verwaltete Integrationen die Option Katalog aus.

AWS Lambda fungiert als C2C-Anschlüsse

Jede Lambda-Funktion des C2C-Konnektors übersetzt und transportiert Befehle und Ereignisse zwischen verwalteten Integrationen und den entsprechenden Aktionen auf Plattformen von Drittanbietern. Weitere Informationen zu Lambda finden Sie unter Was ist AWS Lambda.

Nehmen wir zum Beispiel an, ein Endbenutzer besitzt eine intelligente Glühbirne, die von einem Drittanbieter hergestellt wurde. Mit einem C2C-Anschluss kann ein Endbenutzer über eine verwaltete Integrationsplattform einen Befehl zum Ein- oder Ausschalten dieser Leuchte ausgeben. Dieser Befehl wird dann an die im Connector gehostete Lambda-Funktion weitergeleitet, die die Anfrage in einen API-Aufruf an die Drittanbieterplattform übersetzt, um das Gerät ein- oder auszuschalten.

Die Lambda-Funktion ist erforderlich, wenn Sie die `CreateCloudConnector` API aufrufen. Der Code, der in der Lambda-Funktion bereitgestellt wird, muss alle unter genannten Schnittstellen und Funktionen implementieren. Erstellen Sie einen C2C-Connector (Cloud-to-Cloud)

Connector-Workflow für verwaltete Integrationen

Entwickler müssen C2C-Konnektoren mit verwalteten Integrationen für registrieren. AWS IoT Device Management Durch diesen Registrierungsprozess wird eine logische Connector-Ressource erstellt, auf die Kunden zugreifen können, um den Connector zu verwenden.

> **Note**
>
> Ein C2C-Connector ist ein Satz von Metadaten, der in verwalteten Integrationen für AWS IoT Device Management erstellt wurde, um den Konnektor zu beschreiben.

Das folgende Diagramm zeigt die Rolle eines C2C-Connectors beim Senden eines Befehls von der mobilen Anwendung an ein mit der Cloud verbundenes Gerät. Der C2C-Konnektor fungiert als

Übersetzungsebene zwischen verwalteten Integrationen für AWS IoT Device Management und einer Cloud-Plattform eines Drittanbieters.

Richtlinien für die Verwendung eines C2C (cloud-to-cloud) - Connectors

Jeder von Ihnen erstellte C2C-Konnektor ist Ihr Inhalt, und jeder von einem anderen Kunden erstellte C2C-Connector, auf den Sie zugreifen, ist Inhalt von Drittanbietern. AWS erstellt oder verwaltet keine C2C-Konnektoren im Rahmen verwalteter Integrationen.

Sie können Ihre C2C-Konnektoren mit anderen Kunden von verwalteten Integrationen teilen. Wenn Sie dies tun, autorisieren Sie AWS als Ihr Dienstanbieter, diese C2C-Konnektoren und die zugehörigen Kontaktinformationen auf der AWS Konsole aufzulisten, und Sie verstehen, dass andere AWS Kunden Sie kontaktieren könnten. Sie sind allein dafür verantwortlich, Kunden Zugang zu Ihren C2C-Steckverbindern zu gewähren, und für alle Bedingungen, die den Zugang anderer AWS Kunden zu Ihren C2C-Steckverbindern regeln, verantwortlich.

Erstellen Sie einen C2C-Connector (Cloud-to-Cloud)

In den folgenden Abschnitten werden die Schritte zur Erstellung eines C2C-Connectors (Cloud-to-Cloud) für verwaltete Integrationen für AWS IoT Device Management behandelt.

Topics

Voraussetzungen

Bevor Sie einen C2C-Connector (Cloud-to-Cloud) erstellen, benötigen Sie Folgendes:

- Und AWS-Konto um Ihren C2C-Connector zu hosten und ihn über verwaltete Integrationen zu registrieren. Weitere Informationen finden Sie unter Erstellen eines. AWS-Konto
- Wenn Sie Ihren Connector erstellen, benötigen Sie bestimmte IAM-Berechtigungen. Um das
- Stellen Sie sicher, dass die Cloud-Drittanbieter, für die der Connector vorgesehen ist, die OAuth 2.0-Autorisierung unterstützen. Weitere Informationen finden Sie unter OAuth 2.0-Anforderungen für die Kontoverknüpfung.

  Um den Connector zu testen, muss der Entwickler des Connectors außerdem über Folgendes verfügen:

  - Eine Client-ID aus der Drittanbieter-Cloud, die mit Ihrem C2C-Connector verknüpft werden soll
  - Ein Client-Geheimnis aus der Drittanbieter-Cloud, das mit Ihrem C2C-Connector verknüpft werden soll
  - Eine OAuth 2.0-Autorisierungs-URL
  - Eine OAuth 2.0-Token-URL
- Alle API-Schlüssel, die von Ihrer Drittanbieter-API benötigt werden
- Alle API-Schlüssel, die für Ihre API-Registrierung oder Ihr Allowlisting eines Drittanbieters für die von gehostete OAuth Callback-URL erforderlich sind. AWS Einige Drittanbieter erlauben ausdrücklich eine OAuth Weiterleitungs-URL, während andere über einen Workflow verfügen, bei dem sich Benutzer anmelden und die URL registrieren können. OAuth Wenden Sie sich an den jeweiligen Drittanbieter, um zu erfahren, was erforderlich ist, um den Umleitungsendpunkt für verwaltete Integrationen OAuth auf die Zulassungsliste zu setzen

Erforderliche Berechtigungen

Wenn Sie Ihren Connector erstellen, benötigen Sie bestimmte IAM-Berechtigungen. Zusätzlich zu den `iotmanagedintegrations:` Berechtigungen für die Aktionen benötigen Sie die folgenden Berechtigungen:

- CreateAccountAssociation, CreateConnectorDestinationGetAccountAssociation, und StartAccountAssociationRefresh, benötigen `secretsmanager:GetSecretValue`
- CreateCloudConnector erfordert `lambda:Invoke`

Weitere Informationen zu `iotmanagedintegrations:` Berechtigungen und Aktionen finden Sie unter Durch AWS verwaltete Integrationen definierte Aktionen

Anforderungen an den C2C-Anschluss

Der von Ihnen entwickelte C2C-Konnektor erleichtert die bidirektionale Kommunikation zwischen verwalteten Integrationen für AWS IoT Device Management und einer Drittanbieter-Cloud. Der Konnektor muss Schnittstellen für verwaltete Integrationen für AWS IoT Device Management implementieren, um Aktionen im Namen von Endbenutzern durchzuführen. Diese Schnittstellen bieten Funktionen zur Erkennung von Endbenutzergeräten, zur Initiierung von Gerätebefehlen, die von verwalteten Integrationen für AWS IoT Device Management gesendet werden, und zur Identifizierung von Benutzern anhand eines Zugriffstokens. Um den Gerätebetrieb zu unterstützen, muss der Konnektor die Übersetzung der Anfrage- und Antwortnachrichten zwischen verwalteten Integrationen für AWS IoT Device Management und der zugehörigen Drittanbieterplattform verwalten.

Die folgenden Anforderungen gelten für den C2C-Connector:

- Der Autorisierungsserver eines Drittanbieters muss den OAuth 2.0-Standards sowie den unter aufgeführten Konfigurationen entsprechen. OAuth Anforderungen an die Konfiguration
- Ein C2C-Konnektor ist erforderlich, um Identifikatoren aus AWS Implementierungen des Themendatenmodells zu interpretieren und die Antworten und Ereignisse auszugeben, die den AWS Implementierungen des Themendatenmodells entsprechen. Weitere Informationen finden Sie unter AWS Implementierung des Matter-Datenmodells.
- Ein C2C-Connector muss in der Lage sein, die verwalteten Integrationen für AWS IoT Device Management APIs mit `SigV4` Authentifizierung aufzurufen. Für asynchrone Ereignisse, die mit der SendConnectorEvent API gesendet werden, müssen dieselben AWS-Konto Anmeldeinformationen, die zur Registrierung des Connectors verwendet wurden, zum Signieren der entsprechenden Anfrage verwendet werden. SendConnectorEvent
- Der Konnektor muss die `AWS.DeactivateUser` Operationen`AWS.ActivateUser`, `AWS.DiscoverDevices``AWS.SendCommand`, und implementieren.
- Wenn Ihr C2C-Connector Ereignisse von Drittanbietern empfängt, die sich auf Gerätebefehle oder Geräteerkennung beziehen, muss er diese an verwaltete Integrationen mit der `SendConnectorEvent` API weiterleiten. Weitere Informationen zu diesen Ereignissen und der `SendConnectorEvent` API finden Sie unter. SendConnectorEvent

> **Note**
>
> Die `SendConnectorEvent` API ist Teil des Managed Integrations SDK und wird verwendet, anstatt Anfragen manuell zu erstellen und zu signieren.

OAuth 2.0-Anforderungen für die Kontoverknüpfung

Jeder C2C-Connector ist auf einen OAuth 2.0-Autorisierungsserver angewiesen, um Endbenutzer zu authentifizieren. Über diesen Server verknüpfen Endbenutzer ihre Drittanbieterkonten mit der Geräteplattform des Kunden. Die Kontoverknüpfung ist der erste Schritt, den ein Endbenutzer benötigt, um Geräte zu verwenden, die von Ihrem C2C-Anschluss unterstützt werden. Weitere Informationen zu den verschiedenen Rollen bei der Kontoverknüpfung und OAuth 2.0 finden Sie unterRollen für die Kontoverknüpfung.

Ihr C2C-Connector muss zwar keine spezifische Geschäftslogik implementieren, um den Autorisierungsablauf zu unterstützen, aber der mit Ihrem C2C-Connector verknüpfte OAuth2 .0-Autorisierungsserver muss die Anforderungen erfüllen. OAuth Anforderungen an die Konfiguration

> **Note**
>
> Verwaltete Integrationen unterstützen AWS IoT Device Management nur OAuth 2.0 mit einem Autorisierungscodefluss. Weitere Informationen finden Sie in RFC 6749.

Die Kontoverknüpfung ist ein Prozess, der es verwalteten Integrationen und dem Connector ermöglicht, mithilfe eines Zugriffstokens auf die Geräte eines Endbenutzers zuzugreifen. Dieses Token bietet verwaltete Integrationen für AWS IoT Device Management mit Zustimmung des Endbenutzers, sodass der Connector über API-Aufrufe mit den Daten des Endbenutzers interagieren kann. Weitere Informationen finden Sie unter Workflow zur Kontoverknüpfung.

Wir empfehlen, diese sensiblen Token nicht in irgendwelchen Protokollen zu protokollieren. Wenn sie jedoch in Protokollen gespeichert werden, empfehlen wir Ihnen, die Datenschutzrichtlinien für CloudWatch Logs zu verwenden, um die Token in den Protokollen zu maskieren. Weitere Informationen finden Sie unter Schützen sensibler Protokolldaten durch Maskierung.

Managed Integrations for erhält AWS IoT Device Management kein Zugriffstoken direkt, sondern über den Authorization Code Grant Type. Zunächst müssen verwaltete Integrationen für AWS IoT Device Management einen Autorisierungscode erhalten. Anschließend wird der Code gegen ein Zugriffstoken und ein Aktualisierungstoken ausgetauscht. Das Aktualisierungstoken wird verwendet, um ein neues Zugriffstoken anzufordern, wenn das alte Zugriffstoken abläuft. Wenn sowohl das Zugriffstoken als auch das Aktualisierungstoken abgelaufen sind, müssen Sie die Kontoverknüpfung erneut ausführen. Sie können dies mit der `StartAccountAssociationRefresh` API-Operation tun.

> **Important**
>
> Das ausgegebene Zugriffstoken muss pro Benutzer, aber nicht pro Client gültig sein. OAuth Das Token sollte nicht den Zugriff auf alle Geräte aller Benutzer unter dem Client ermöglichen.
> Der Autorisierungsserver muss einen der folgenden Schritte ausführen:
>
> - Stellen Sie Zugriffstoken aus, die eine extrahierbare Endbenutzer-ID (Ressourcenbesitzer) enthalten, z. B. ein JWT-Token.
> - Gibt die Endbenutzer-ID für jedes ausgegebene Zugriffstoken zurück.

OAuth Anforderungen an die Konfiguration

Die folgende Tabelle zeigt die erforderlichen Parameter von Ihrem OAuth Autorisierungsserver für verwaltete Integrationen für AWS IoT Device Management, um die Kontoverknüpfung durchzuführen:

OAuth Serverparameter

Feld	Erforderlich	Kommentar
`clientId`	Ja	Eine öffentliche Kennung für Ihre Anwendung. Sie wird zur Initiierung von Authentif izierungsabläufen verwendet und kann öffentlich geteilt werden.
`clientSecret`	Ja	Ein geheimer Schlüssel, der zur Authentifizierung der Anwendung beim Autorisie rungsserver verwendet wird, insbesondere beim Austausch eines Autorisierungscodes gegen ein Zugriffstoken. Er sollte vertraulich behandelt und nicht öffentlich weitergeg eben werden.

`authorizationType`	Ja	Die Art der Autorisierung, die von dieser Autorisierungskonfiguration unterstützt wird. Derzeit ist "OAuth 2.0" der einzige Wert, der unterstützt wird.
`authUrl`	Ja	Die Autorisierungs-URL für den Cloud-Drittanbieter.
`tokenUrl`	Ja	Die Token-URL für den Cloud-Drittanbieter.
`tokenEndpointAuthenticationScheme`	Ja	Authentifizierungsschema entweder „HTTP_BASIC“ oder „REQUEST_BODY_CREDENTIALS“. HTTP_BASIC signalisiert, dass die Client-Anmeldeinformationen im Autorisierungsheader enthalten sind, während die Ladder signalisiert, dass sie im Anforderungstext enthalten sind.

Der OAuth Server, den Sie verwenden, muss so konfiguriert sein, dass die Zeichenkettenwerte des Zugriffstokens Base64-kodiert mit dem UTF-8-Zeichensatz sein müssen.

Rollen für die Kontoverknüpfung

Um einen C2C-Connector zu erstellen, benötigen Sie einen OAuth 2.0-Autorisierungsserver und eine Kontoverknüpfung. Weitere Informationen finden Sie unter Workflow zur Kontoverknüpfung.

OAuth 2.0 definiert die folgenden vier Rollen bei der Implementierung der Kontoverknüpfung:

1. Autorisierungsserver
2. Besitzer der Ressource (Endbenutzer)
3. Ressourcenserver

4. Client

Im Folgenden wird jede dieser OAuth Rollen definiert:

Autorisierungsserver

Der Autorisierungsserver ist der Server, der die Identität eines Endbenutzers in einer Drittanbieter-Cloud identifiziert und authentifiziert. Die von diesem Server bereitgestellten Zugriffstoken können das Kundenplattformkonto des AWS Endbenutzers mit seinem Plattformkonto eines Drittanbieters verknüpfen. Dieser Vorgang wird als Kontoverknüpfung bezeichnet.

Der Autorisierungsserver unterstützt die Kontoverknüpfung, indem er Folgendes bereitstellt:

- Zeigt eine Anmeldeseite an, auf der sich der Endbenutzer bei Ihrem System anmelden kann. Dies wird in der Regel als Autorisierungsendpunkt bezeichnet.
- Authentifiziert den Endbenutzer in Ihrem System.
- Generiert einen Autorisierungscode, der den Endbenutzer identifiziert.
- Übergibt den Autorisierungscode an verwaltete Integrationen für AWS IoT Device Management.
- Akzeptiert den Autorisierungscode von verwalteten Integrationen für AWS IoT Device Management und gibt ein Zugriffstoken zurück, mit dem verwaltete Integrationen für AWS IoT Device Management auf die Daten des Endbenutzers in Ihrem System zugreifen können. Dies erfolgt in der Regel über eine separate URI, die als Token-URI oder Endpunkt bezeichnet wird.

> **Important**
>
> Der Autorisierungsserver muss den OAuth 2.0-Autorisierungscodefluss unterstützen, um mit verwalteten Integrationen für AWS IoT Device Management Connector verwendet zu werden. Verwaltete Integrationen für AWS IoT Device Management unterstützen auch den Autorisierungscodefluss mit Proof Key for Code Exchange (PKCE).
> Der Autorisierungsserver muss entweder:
>
> - Zugriffstoken ausgeben, die eine extrahierbare Endbenutzer- oder Ressourcenbesitzer-ID enthalten, z. B. JWT-Token
> - In der Lage sein, die Endbenutzer-ID für jedes ausgegebene Zugriffstoken zurückzugeben
>
> Andernfalls kann Ihr Connector den erforderlichen `AWS.ActivateUser` Vorgang nicht unterstützen. Dadurch wird die Verwendung von Connectoren bei verwalteten Integrationen verhindert.

Wenn der Connector-Entwickler oder -Besitzer keinen eigenen Autorisierungsserver unterhält, muss der verwendete Autorisierungsserver die Autorisierung für Ressourcen bereitstellen, die von der Drittanbieterplattform des Connector-Entwicklers verwaltet werden. Das bedeutet, dass alle Token, die über verwaltete Integrationen vom Autorisierungsserver empfangen werden, aussagekräftige Sicherheitsgrenzen für Geräte (die Ressource) vorsehen müssen. Beispielsweise erlaubt ein Endbenutzer-Token keine Befehle auf dem Gerät eines anderen Endbenutzers.
Die durch das Token bereitgestellten Berechtigungen sind Ressourcen innerhalb der Plattform zugeordnet. Betrachten Sie das Beispiel von Lights Incorporated. Wenn ein Endbenutzer den Ablauf der Kontoverknüpfung mit seinem Connector startet, wird er auf die Anmeldeseite von Lights Incorporated umgeleitet, auf der sein Autorisierungsserver angezeigt wird. Sobald sie sich angemeldet und dem Client Berechtigungen erteilt haben, stellen sie ein Token bereit, das dem Connector Zugriff auf Ressourcen innerhalb seines Lights Incorporated-Kontos gewährt.

Eigentümer der Ressource (Endbenutzer)

Als Eigentümer der Ressource gewähren Sie einem Kunden mit verwalteten Integrationen für AWS IoT Device Management den Zugriff auf Ressourcen, die mit Ihrem Konto verknüpft sind, indem Sie eine Kontoverknüpfung durchführen. Stellen Sie sich zum Beispiel die intelligente Glühbirne vor, die ein Endbenutzer in die mobile Anwendung von Lights Incorporated integriert hat. Der Ressourcenbesitzer bezieht sich auf das Endbenutzerkonto, das das Gerät gekauft und integriert hat. In unserem Beispiel wird der Ressourcenbesitzer als Lights Incorporated OAuth2 2.0-Konto modelliert. Als Besitzer der Ressource bietet dieses Konto die Erlaubnis, Befehle zu erteilen und das Gerät zu verwalten.

Ressourcenserver

Dies ist der Server, auf dem geschützte Ressourcen gehostet werden, für deren Zugriff eine Autorisierung erforderlich ist (Gerätedaten). Der AWS Kunde muss im Namen eines Endbenutzers auf geschützte Ressourcen zugreifen, und zwar über verwaltete Integrationen für AWS IoT Device Management-Konnektoren nach der Kontoverknüpfung. Betrachtet man die intelligente Glühbirne von früher als Beispiel: Der Ressourcenserver ist ein cloudbasierter Dienst von Lights Incorporated, der die Glühbirne verwaltet, nachdem sie integriert wurde. Über den Ressourcenserver kann der Ressourcenbesitzer Befehle an die intelligente Glühbirne senden, z. B. sie ein- und ausschalten. Die geschützte Ressource gewährt nur Berechtigungen für das Konto des Endbenutzers und für andere Benutzer, für die accounts/entities er möglicherweise Berechtigungen erteilt hat.

Client

In diesem Zusammenhang ist der Client Ihr C2C-Connector. Ein Client ist definiert als eine Anwendung, der im Namen des Endbenutzers Zugriff auf Ressourcen innerhalb eines Ressourcenservers gewährt wird. Der Prozess der Kontoverknüpfung stellt den Konnektor, den Client, dar, der Zugriff auf die Ressourcen eines Endbenutzers in der Drittanbieter-Cloud anfordert.

Obwohl der Connector der OAuth Client ist, führt Managed Integrations for AWS IoT Device Management Operationen im Namen des Connectors durch. Beispielsweise stellen verwaltete Integrationen für AWS IoT Device Management Anfragen an den Autorisierungsserver, um ein Zugriffstoken zu erhalten. Der Connector wird immer noch als Client betrachtet, da er die einzige Komponente ist, die jemals auf die geschützte Ressource (Gerätedaten) im Ressourcenserver zugreift.

Stellen Sie sich die intelligente Glühbirne vor, die von einem Endbenutzer integriert wurde. Nachdem die Kontoverknüpfung zwischen der Kundenplattform und dem Autorisierungsserver von Lights Incorporated abgeschlossen ist, kommuniziert der Connector selbst mit dem Ressourcenserver, um Informationen über die intelligente Glühbirne des Endbenutzers abzurufen. Der Connector kann dann Befehle vom Endbenutzer empfangen. Dazu gehört das Ein- oder Ausschalten des Lichts in ihrem Namen über den Ressourcenserver von Lights Incorporated. Daher bezeichnen wir den Connector als Client.

Workflow zur Kontoverknüpfung

Damit die verwalteten Integrationen für die AWS IoT Device Management-Plattform eines Kunden über Ihren C2C-Connector mit den Geräten eines Endbenutzers auf Ihrer Drittanbieterplattform interagieren können, erhält er das Zugriffstoken über den folgenden Workflow:

1. Wenn ein Benutzer das Onboarding von Geräten von Drittanbietern über die Kundenanwendung initiiert, gibt Managed Integrations for AWS IoT Device Management die Autorisierungs-URI sowie den zurück. AssociationId
2. Das Anwendungs-Frontend speichert den AssociationId und leitet den Endbenutzer zur Anmeldeseite der Drittanbieterplattform weiter.
    - Der Endbenutzer meldet sich an. Der Endbenutzer gewährt dem Client Zugriff auf seine Gerätedaten.

3. Die Drittanbieterplattform erstellt einen Autorisierungscode. Der Endbenutzer wird zu der Callback-URI für verwaltete Integrationen für die AWS IoT Device Management-Plattform weitergeleitet, einschließlich des Codes, der an die Umleitungsanfrage angehängt ist.
4. Managed Integrations tauscht diesen Code mit der Token-URI der Drittanbieter-Plattform aus.
5. Die Token-URI validiert den Autorisierungscode und gibt ein OAuth2 0.0-Zugriffstoken und ein Aktualisierungstoken zurück, die dem Endbenutzer zugeordnet sind.
6. Bei verwalteten Integrationen wird der C2C-Konnektor aufgerufen, um den `AWS.ActivateUser` Vorgang zur Kontoverknüpfung abzuschließen und abzurufen. UserId
7. Bei verwalteten Integrationen wird OAuth RedirectUrl (von der Connector-Richtlinienkonfiguration) die Seite mit der erfolgreichen Authentifizierung an die Kundenanwendung zurückgegeben.

   > **Note**
   >
   > Bei Ausfällen fügt Managed Integrations for AWS IoT Device Management die Abfrageparameter error und error_description an die URL an, die Fehlerdetails für die Kundenanwendung bereitstellt.

8. Die Kundenanwendung leitet den Endbenutzer an die weiter. OAuth RedirectUrl Zu diesem Zeitpunkt kennt das Front-End AssociationId der Anwendung die Zuordnung vom ersten Schritt an.

   Alle nachfolgenden Anfragen, die von verwalteten Integrationen für AWS IoT Device Management über den C2C-Connector an die Cloud-Plattform eines Drittanbieters gestellt werden, wie Befehle zum Erkennen von Geräten und zum Senden von Befehlen, enthalten das Zugriffstoken OAuth2 .0.

Das folgende Diagramm zeigt die Beziehung zwischen den wichtigsten Komponenten der Kontoverknüpfung:

Implementieren Sie Operationen an der C2C-Anschlussschnittstelle

Managed Integrations for AWS IoT Device Management definiert vier Vorgänge, die Sie ausführen AWS Lambda müssen, um sich als Konnektor zu qualifizieren. Ihr C2C-Konnektor muss jeden der folgenden Vorgänge implementieren:

1. `AWS.ActivateUser`- Verwaltete Integrationen für AWS IoT Device Management Dienste rufen diese API auf, um eine weltweit eindeutige Benutzerkennung abzurufen, die dem bereitgestellten OAuth2 0.0-Token zugeordnet ist. Dieser Vorgang kann optional verwendet werden, um zusätzliche Anforderungen für den Kontoverknüpfungsprozess zu erfüllen.
2. `AWS.DiscoverDevices`- verwaltete Integrationen für den AWS IoT Device Management Service rufen diese API an Ihren Connector auf, um Benutzergeräte zu erkennen
3. `AWS.SendCommand`- verwaltete Integrationen für den AWS IoT Device Management Service rufen diese API an Ihren Connector auf, um Befehle für Benutzergeräte zu senden
4. `AWS.DeactivateUser`- Managed Integrations for AWS IoT Device Management Service ruft diese API an Ihren Connector auf, um das Zugriffstoken des Benutzers zu deaktivieren, um die Verbindung zu Ihrem Autorisierungsserver zu trennen.

Managed Integrations for ruft AWS IoT Device Management während der Aktion immer die Lambda-Funktion mit einer JSON-Zeichenfolgennutzlast auf. AWS Lambda `invokeFunction` Anforderungsoperationen müssen in jeder Anforderungsnutzlast ein `operationName` Feld enthalten. Weitere Informationen finden Sie unter Invoke in der AWS Lambda API-Referenz.

Jeder Aufruf-Timeout ist auf zwei Sekunden festgelegt. Schlägt der Aufruf fehl, wird er fünfmal wiederholt.

Das Lambda, das Sie für Ihren Connector implementieren, analysiert eine `operationName` aus der Payload der Anfrage und implementiert die entsprechende Funktionalität für die Zuordnung zur Drittanbieter-Cloud:

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

    return switch (operation) {
        case ActivateUser -> activateUserManager.activateUser(request);
        case DiscoverDevices -> deviceDiscoveryManager.listDevices(request);
```

```
        case SendCommand -> sendCommandManager.sendCommand(request);
        case DeactivateUser -> deactivateUser.deactivateUser(request);
    };
}
```

> **Note**
>
> Der Entwickler des Connectors muss die im vorherigen `activateUserManager.activateUser(request)` Beispiel aufgeführten `deactivateUser.deactivateUser` Operationen `deviceDiscoveryManager.listDevices(request)sendCommandManager.sendCommand(re` und implementieren.

Das folgende Beispiel beschreibt eine generische Konnektoranforderung von verwalteten Integrationen, in der gemeinsame Felder für alle erforderlichen Schnittstellen vorhanden sind. Aus dem Beispiel können Sie ersehen, dass es sowohl einen Anforderungsheader als auch eine Anforderungs-Payload gibt. Anforderungsheader sind auf jeder Bedienoberfläche üblich.

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

Standard-Anforderungsheader

Die Standard-Header-Felder lauten wie folgt.

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

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

Jede API, die von einem Connector gehostet wird, muss die folgenden Header-Parameter verarbeiten:

Standard-Header und -felder

Feld	Erforderlich/optional	Beschreibung
`header:auth`	Ja	Autorisierungsinformationen, die vom C2C-Steckverbinder hersteller bei der Registrierung des Connectors bereitgestellt wurden.
`header:auth:token`	Ja	Autorisierungstoken des Benutzers, das vom Cloud-Drittanbieter generiert und mit dem verknüpft wurde. `connectorAssociationID`
`header:auth:type`	Ja	Die Art der erforderlichen Autorisierung.

> **Note**
>
> Allen Anfragen an Ihren Connector wird das Zugriffstoken des Endbenutzers angehängt. Sie können davon ausgehen, dass die Kontoverknüpfung zwischen dem Endbenutzer und dem Kunden für verwaltete Integrationen bereits erfolgt ist.

Anforderungs-Nutzlast

Zusätzlich zu den allgemeinen Headern hat jede Anfrage eine Nutzlast. Diese Payload wird zwar eindeutige Felder für jeden Operationstyp haben, aber jede Payload hat eine Reihe von Standardfeldern, die immer vorhanden sein werden.

Payload-Felder anfordern:

- `operationName`: Der Vorgang einer bestimmten Anfrage, der einem der folgenden Werte entspricht:`AWS.ActivateUser`,, `AWS.SendCommandAWS.DiscoverDevices`,`AWS.DeactivateUser`.
- `operationVersion`: Jeder Vorgang ist versioniert, um seine Weiterentwicklung im Laufe der Zeit zu ermöglichen und eine stabile Schnittstellendefinition für Konnektoren von Drittanbietern bereitzustellen. Managed Integrations übergibt ein Versionsfeld in der Nutzlast aller Anfragen.
- `connectorId`: Die ID des Connectors, an den die Anfrage gesendet wurde.

Standard-Antwort-Header

Jeder Vorgang reagiert mit einer Antwort ACK auf verwaltete Integrationen für AWS IoT Device Management, die bestätigt, dass Ihr C2C-Connector die Anfrage empfangen und mit der Bearbeitung begonnen hat. Im Folgenden finden Sie ein allgemeines Beispiel für diese Antwort:

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

Jede Operationsantwort muss den folgenden gemeinsamen Header haben:

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

In der folgenden Tabelle ist der Standard-Antwort-Header aufgeführt:

Standard-Antwort-Header und -Feld

Feld	Erforderlich/optional	Kommentar
`header:responseCode`	Ja	ENUM von Werten, die den Ausführungsstatus der Anfrage angeben.

In den verschiedenen Connector-Schnittstellen und API-Schemas, die in diesem Dokument beschrieben werden, gibt es ein `responseMessage` Oder-Feld. `Message` Dies ist ein optionales Feld, das für den C2C-Konnektor Lambda verwendet wird, um mit jedem Kontext bezüglich der Anfrage und ihrer Ausführung zu antworten. Vorzugsweise `200` sollten alle Fehler, die zu einem anderen Statuscode führen, einen Nachrichtenwert enthalten, der den Fehler beschreibt.

Beantworten Sie Anfragen zum Betrieb des C2C-Connectors mit der API SendConnectorEvent

Managed Integrations for AWS IoT Device Management erwartet, dass sich Ihr Connector bei jedem AND-Vorgang asynchron verhält. `AWS.SendCommand` `AWS.DiscoverDevices` Das bedeutet, dass die erste Antwort auf diese Operationen lediglich „bestätigt“, dass Ihr C2C-Connector die Anfrage erhalten hat.

Mithilfe der `SendConnectorEvent` API wird von Ihrem Connector erwartet, dass er die Ereignistypen aus der folgenden Liste an für `AWS.DiscoverDevices` und `AWS.SendCommand` Operationen sowie an proaktive Geräteereignisse (z. B. manuelles Ein- und Ausschalten eines Lichts) sendet. Eine ausführliche Erläuterung dieser Ereignistypen und ihrer Anwendungsfälle finden Sie unter Implementieren Sie das AWS. DiscoverDevices BetriebImplementieren Sie das AWS. SendCommand Betrieb, undGeräteereignisse mit der API senden SendConnectorEvent.

Wenn Ihr C2C-Connector beispielsweise eine `DiscoverDevices` Anfrage empfängt, erwartet Managed Integrations for AWS IoT Device Management, dass er synchron mit dem oben definierten Antwortformat reagiert. Anschließend müssen Sie die `SendConnectorEvent` API mit der in definierten Anforderungsstruktur für ein Implementieren Sie das AWS. DiscoverDevices Betrieb DEVICE_DISCOVERY-Ereignis aufrufen. Der `SendConnectorEvent` On-API-Aufruf kann überall dort erfolgen, wo Sie Zugriff auf Ihre AWS-Konto Lambda-Anmeldeinformationen für den C2C-

Connector haben. Die Geräteerkennung ist erst erfolgreich, wenn Managed Integrations for AWS IoT Device Management dieses Ereignis empfängt.

> **Note**
>
> Alternativ kann der `SendConnectorEvent` API-Aufruf bei Bedarf vor der Lambda-Aufrufreaktion des C2C-Konnektors erfolgen. Dieser Ablauf widerspricht jedoch dem asynchronen Modell für die Softwareentwicklung.

- SendConnectorEvent- Ihr Connector nennt dies verwaltete Integrationen für die AWS-IoT-Gerätemanagement-API, um Geräteereignisse an verwaltete Integrationen für AWS IoT Device Management zu senden. Nur 3 Arten von Ereignissen werden von verwalteten Integrationen akzeptiert:
  - "DEVICE_DISCOVERY" — Dieser Ereignisvorgang wird verwendet, um eine Liste der erkannten Geräte in der Drittanbieter-Cloud für ein bestimmtes Zugriffstoken zu senden.
  - „DEVICE_COMMAND_RESPONSE" — Diese Ereignisoperation soll verwendet werden, um ein bestimmtes Geräteereignis als Ergebnis der Befehlsausführung zu senden.
  - „DEVICE_EVENT" — Diese Ereignisoperation muss für jedes Ereignis verwendet werden, das vom Gerät ausgeht und nicht das direkte Ergebnis eines benutzerbasierten Befehls ist. Dies kann als allgemeiner Ereignistyp dienen, um proaktiv Gerätestatusänderungen oder Benachrichtigungen zu melden.

Implementieren Sie das AWS. ActivateUser Betrieb

Der `AWS.ActivateUser` Vorgang ist für verwaltete Integrationen für AWS IoT Device Management erforderlich, um eine Benutzerkennung aus dem OAuth2 0.0-Token eines Endbenutzers abzurufen. Verwaltete Integrationen für übergeben AWS IoT Device Management das OAuth Token im Anforderungsheader und erwartet, dass Ihr Connector die weltweit eindeutige Benutzerkennung in die Antwortnutzlast einbezieht. Dieser Vorgang erfolgt nach einer erfolgreichen Kontoverknüpfung.

In der folgenden Liste werden die Anforderungen für Ihren Connector beschrieben, um einen erfolgreichen `AWS.Activate` Benutzerfluss zu ermöglichen.

- Ihr C2C-Connector Lambda kann eine `AWS.ActivateUser` Betriebsanforderungsnachricht von verwalteten Integrationen für AWS IoT Device Management verarbeiten.

- Ihr C2C-Connector-Lambda kann aus einem bereitgestellten OAuth2 0.0-Token eine eindeutige Benutzerkennung ermitteln. Normalerweise kann es entweder aus dem Token selbst extrahiert werden, wenn es sich um ein JWT-Token handelt, oder durch das Token vom Autorisierungsserver angefordert werden.

`AWS.ActivateUser`-Workflow

1. Verwaltete Integrationen für AWS IoT Device Management Aufrufe Ihres C2C-Connectors Lambda mit der folgenden Nutzlast:

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

2. Der C2C-Connector bestimmt die Benutzer-ID entweder anhand des Tokens oder durch eine Abfrage Ihres Drittanbieter-Ressourcenservers, die in die Antwort aufgenommen werden soll. `AWS.ActivateUser`
3. Der C2C-Konnektor reagiert auf den Aufruf der `AWS.ActivateUser` Operation Lambda, einschließlich der Standardnutzlast sowie der entsprechenden Benutzer-ID innerhalb des Felds. `userId`

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

```
}
```

Implementieren Sie das AWS. DiscoverDevices Betrieb

Die Geräteerkennung gleicht die Liste der physischen Geräte, die dem Endbenutzer gehören, mit den digitalen Darstellungen dieser Endbenutzergeräte ab, die in verwalteten Integrationen für AWS IoT Device Management verwaltet werden. Sie wird von einem AWS Kunden erst auf Geräten ausgeführt, die dem Endbenutzer gehören, nachdem die Kontoverknüpfung zwischen dem Benutzer und den verwalteten Integrationen für AWS IoT Device Management abgeschlossen ist. Die Geräteerkennung ist ein asynchroner Prozess, bei dem verwaltete Integrationen für AWS IoT Device Management einen Connector aufrufen, um die Geräteerkennungsanfrage zu initiieren. Ein C2C-Connector gibt asynchron eine Liste der erkannten Endbenutzergeräte mit einer Referenz-ID (als`deviceDiscoveryId`) zurück, die von verwalteten Integrationen generiert wurde.

Das folgende Diagramm veranschaulicht den Workflow zur Geräteerkennung zwischen dem Endbenutzer und verwalteten Integrationen für AWS IoT Device Management:

AWS. DiscoverDevices Arbeitsablauf

1. Der Kunde leitet den Geräteerkennungsprozess im Namen des Endbenutzers ein.
2. Managed Integrations for AWS IoT Device Management generiert eine Referenz-ID, die `deviceDiscoveryId` für die vom Kunden generierte Anfrage zur Geräteerkennung aufgerufen wird. AWS
3. Verwaltete Integrationen für AWS IoT Device Management sendet über die `AWS.DiscoverDevices` Bedienoberfläche eine Anfrage zur Geräteerkennung an den C2C-Anschluss, einschließlich einer gültigen OAuth `accessToken` Anfrage für den Endbenutzer sowie für die. `deviceDiscoveryId`
4. Ihr Connector wird gespeichert`deviceDiscoveryId`, um in das `DEVICE_DISCOVERY` Ereignis aufgenommen zu werden. Dieses Ereignis enthält auch eine Liste der erkannten Endbenutzergeräte und muss als `DEVICE_DISCOVERY` Ereignis an verwaltete Integrationen für AWS IoT Device Management mit der `SendConnectorEvent` API gesendet werden.
5. Ihr C2C-Connector ruft den Ressourcenserver auf, um alle Geräte abzurufen, die dem Endbenutzer gehören.
6. Ihr C2C-Connector-Lambda reagiert auf den Lambda-Aufruf (`invokeFunction`) mit der ACK-Antwort zurück auf verwaltete Integrationen für AWS IoT Device Management, die als erste

Antwort für den Vorgang fungiert. `AWS.DiscoverDevices` Managed Integrations benachrichtigt den Kunden mit einem ACK über den von ihm eingeleiteten Geräteerkennungsprozess.

7. Ihr Ressourcenserver sendet Ihnen eine Liste der Geräte, die dem Endbenutzer gehören und von diesem betrieben werden.
8. Ihr Connector konvertiert jedes Endbenutzergerät in das für verwaltete Integrationen für AWS IoT Device Management erforderliche Geräteformat`ConnectorDeviceId`, `ConnectorDeviceName` einschließlich eines Fähigkeitsberichts für jedes Gerät.
9. Der C2C-Connector liefert auch Informationen über den `UserId` Besitzer des erkannten Geräts. Es kann von Ihrem Ressourcenserver entweder als Teil der Geräteliste oder in einem separaten Aufruf abgerufen werden, abhängig von Ihrer Ressourcenserver-Implementierung.
10. Als Nächstes ruft Ihr C2C-Connector die verwalteten Integrationen für die AWS IoT Device Management API über Sigv4 unter Verwendung von AWS-Konto Anmeldeinformationen und mit dem Betriebsparameter „DEVICE_DISCOVERY“ auf. `SendConnectorEvent` Jedes Gerät in der Liste der Geräte, die an verwaltete Integrationen für AWS IoT Device Management gesendet werden, wird durch gerätespezifische Parameter wie `connectorDeviceIdconnectorDeviceName`, und a dargestellt. `capabilityReport`

    - Basierend auf Ihrer Antwort auf den Ressourcenserver müssen Sie verwaltete Integrationen für AWS IoT Device Management entsprechend benachrichtigen.

      Wenn Ihr Ressourcenserver beispielsweise eine paginierte Antwort auf die Liste der erkannten Geräte für einen Endbenutzer erhält, können Sie für jede Abfrage ein individuelles `DEVICE_DISCOVERY` Betriebsereignis mit dem `statusCode` Parameter senden. `3xx` Wenn Ihre Geräteerkennung noch läuft, wiederholen Sie die Schritte 5, 6 und 7.
11. Managed Integrations for AWS IoT Device Management sendet eine Benachrichtigung an den Kunden darüber, dass die Geräte des Endbenutzers entdeckt wurden.
12. Wenn Ihr C2C-Connector ein `DEVICE_DISCOVERY` Betriebsereignis sendet, bei dem der `statusCode` Parameter auf den Wert 200 aktualisiert wird, werden verwaltete Integrationen den Kunden über den Abschluss des Workflows zur Geräteerkennung informieren.

> **Important**
>
> Falls gewünscht, können die Schritte 7 bis 11 vor Schritt 6 erfolgen. Wenn Ihre Drittanbieterplattform beispielsweise über eine API zum Auflisten der Geräte eines Endbenutzers verfügt, kann das DEVICE_DISCOVERY-Ereignis mit gesendet werden,

`SendConnectorEvent` bevor der C2C-Connector Lambda mit dem typischen ACK antwortet.

Anforderungen an den C2C-Anschluss für die Geräteerkennung

In der folgenden Liste sind die Anforderungen für Ihren C2C-Anschluss aufgeführt, um eine erfolgreiche Geräteerkennung zu ermöglichen.

- Der C2C-Konnektor Lambda a kann eine Geräteerkennungsanfrage von verwalteten Integrationen für AWS IoT Device Management verarbeiten und den Vorgang abwickeln. `AWS.DiscoverDevices`
- Ihr C2C-Connector kann die verwalteten Integrationen für AWS IoT Device Management APIs über SigV4 aufrufen und verwendet dabei die Anmeldeinformationen des Benutzers, der für die Registrierung des Connectors AWS-Konto verwendet wurde.

Prozess der Geräteerkennung

In den folgenden Schritten wird der Geräteerkennungsprozess mit Ihrem C2C-Connector und verwalteten Integrationen für AWS IoT Device Management beschrieben.

Prozess der Geräteerkennung

1. Verwaltete Integrationen lösen die Geräteerkennung aus:
    - Senden Sie eine POST-Anfrage `DiscoverDevices` mit der folgenden JSON-Nutzlast an:

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

```
   }
}
```

2. Der Connector bestätigt die Entdeckung:

   - Der Connector sendet eine Bestätigung mit der folgenden JSON-Antwort:

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

3. Der Connector sendet ein Geräteerkennungsereignis:

   - Senden Sie eine POST-Anfrage `/connector-event/`*`{your_connector_id}`* mit der folgenden JSON-Nutzlast an:

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

```
      "id":"0x0006",
      "revision":1,
      "attributes":[{
       "id":"0x0000",
      }],
      "commands":["0x00","0x01"],
      "events":["0x00"]
     }]
    }]
   }
        }
    ]
}
```

Konstruieren Sie eine CapabilityReport für das DISCOVER_DEVICES-Ereignis

Wie aus der oben definierten Ereignisstruktur ersichtlich, benötigt jedes in einem DISCOVER_DEVICES-Ereignis gemeldete Gerät, das als Reaktion auf eine `AWS.DiscoverDevices` Operation dient, eine, CapbilityReport um die Fähigkeiten des entsprechenden Geräts zu beschreiben. Ein `CapabilityReport` teilt verwalteten Integrationen für AWS IoT Device Management-Gerätefunktionen in einem Matter-konformen Format mit. Die folgenden Felder müssen im ` CapabilityReport` angegeben werden:

- `nodeId`, Zeichenfolge: Bezeichner für den Geräteknoten, der Folgendes enthält `endpoints`
- `version`, Zeichenfolge: Version dieses Geräteknotens, die vom Connector-Entwickler festgelegt wurde
- `endpoints`, Liste<Cluster>: Liste der AWS Implementierungen des Matter Data Model, die von diesem Geräteendpunkt unterstützt werden.
  - `id`, Zeichenfolge: Vom Connector-Entwickler festgelegte Endpunkt-ID
  - `deviceTypes`, Liste<String>: Liste der Gerätetypen, die dieser Endpunkt erfasst, z. B. „Kamera“.
  - `clusters`, Liste<Cluster>: Liste der AWS Implementierungen des Matter Data Model, das dieser Endpunkt unterstützt.
    - `id`, Zeichenfolge: Cluster-ID, wie im Matter-Standard definiert.
    - `revision`, Integer: Cluster-Revisionsnummer, wie im Matter-Standard definiert.

- `attributes`,<String, Object> Map: Übersicht der Attributkennungen und der entsprechenden aktuellen Gerätestatuswerte mit Kennungen und gültigen Werten, die durch den Matter-Standard definiert sind.
  - `id`, Zeichenfolge: Attribut-ID, wie sie durch AWS Implementierungen des Matter Data Model definiert wurde.
  - `value`, Objekt: Der aktuelle Wert des durch die Attribut-ID definierten Attributs. Der Typ des „Werts“ kann sich je nach Attribut ändern. Das `value` Feld ist für jedes Attribut optional und sollte nur enthalten sein, wenn Ihr Connector-Lambda den aktuellen Status während der Erkennung ermitteln kann.
- `commands`, Liste<String>: Liste der Befehle, die diesen Cluster gemäß der Definition im Matter-Standard IDs unterstützt haben.
- `events`, Liste<String>: Liste der Ereignisse, die diesen Cluster gemäß der Definition im Matter-Standard IDs unterstützt haben.

Die aktuelle Liste der unterstützten Funktionen und der entsprechenden AWS Implementierungen des Matter-Datenmodells finden Sie in der neuesten Version der Datenmodell-Dokumentation.

Implementieren Sie das AWS. SendCommand Betrieb

Dieser `AWS.SendCommand` Vorgang ermöglicht verwalteten Integrationen für AWS IoT Device Management, Befehle, die vom Endbenutzer initiiert wurden, über den AWS Kunden an Ihren Ressourcenserver zu senden. Ihr Ressourcenserver unterstützt möglicherweise mehrere Gerätetypen, wobei jeder Typ sein eigenes Antwortmodell hat. Die Befehlsausführung ist ein asynchroner Prozess, bei dem verwaltete Integrationen für AWS IoT Device Management eine Anfrage zur Befehlsausführung mit einer `TraceID` senden, die Ihr Connector in eine Befehlsantwort einfügt, die über die ``SendConnectorEventAPI an verwaltete Integrationen zurückgesendet wird. Verwaltete Integrationen für AWS IoT Device Management erwarten, dass der Ressourcenserver eine Antwort zurückgibt, die bestätigt, dass der Befehl empfangen wurde, aber nicht unbedingt angibt, dass der Befehl ausgeführt wurde.

Das folgende Diagramm veranschaulicht den Ablauf der Befehlsausführung anhand eines Beispiels, bei dem der Endbenutzer versucht, die Beleuchtung seines Hauses einzuschalten:

Arbeitsablauf für die Ausführung von Gerätebefehlen

1. Ein Endbenutzer sendet mithilfe der AWS Kundenanwendung einen Befehl zum Einschalten eines Lichts.
2. Der Kunde leitet die Befehlsinformationen zusammen mit den Geräteinformationen des Endbenutzers an verwaltete Integrationen für AWS IoT Device Management weiter.
3. Managed Integrations generiert eine „traceId“, die Ihr Connector verwendet, wenn er Befehlsantworten zurück an den Service sendet.
4. Managed Integrations for AWS IoT Device Management sendet die Befehlsanforderung über die `AWS.SendCommand` Bedienoberfläche an Ihren Connector.

    - Die durch diese Schnittstelle definierte Nutzlast besteht aus der Gerätekennung, als Matter formulierten Gerätebefehlenendpoints/clusters/commands, dem Zugriffstoken des Endbenutzers und anderen erforderlichen Parametern.
5. Ihr Connector speichert die Daten`traceId`, die in die Befehlsantwort aufgenommen werden sollen.

    - Ihr Connector übersetzt die Befehlsanforderung für verwaltete Integrationen in das entsprechende Format Ihres Ressourcenservers.
6. Ihr Connector ruft das Zugriffstoken des bereitgestellten Endbenutzers `UserId` ab und ordnet es dem Befehl zu.

    a. Sie `UserId` können entweder mit einem separaten Aufruf von Ihrem Ressourcenserver abgerufen oder im Fall von JWT und ähnlichen Token aus dem Zugriffstoken extrahiert werden.

    b. Die Implementierung hängt von Ihrem Ressourcenserver und den Details des Zugriffstokens ab.
7. Ihr Connector fordert den Ressourcenserver auf, das Licht des Endbenutzers einzuschalten.
8. Der Ressourcenserver interagiert mit dem Gerät.

    a. Der Konnektor leitet an verwaltete Integrationen für AWS IoT Device Management weiter, dass der Ressourcenserver den Befehl übermittelt hat, und antwortet mit einem ACK als erste, synchrone Befehlsantwort.

    b. Verwaltete Integrationen leiten ihn dann zurück an die Kundenanwendung.
9. Wenn das Gerät das Licht einschaltet, wird dieses Geräteereignis von Ihrem Ressourcenserver erfasst.

10. Ihr Ressourcenserver sendet das Geräteereignis an den Connector.
11. Ihr Connector wandelt das vom Ressourcenserver generierte Geräteereignis in den Vorgangstyp DEVICE_COMMAND_RESPONSE für verwaltete Integrationen um.
12. Ihr Connector ruft die `SendConnectorEvent` API mit der Operation „DEVICE_COMMAND_RESPONSE“ auf.
    - Es fügt der ersten Anfrage die von Managed Integrations für AWS IoT Device Management `traceId` bereitgestellten Integrationen hinzu.
13. Managed Integrations benachrichtigt den Kunden über die Änderung des Gerätestatus des Endbenutzers.
14. Der Kunde informiert den Endbenutzer darüber, dass das Licht des Geräts eingeschaltet ist.

> **Note**
>
> Ihre Ressourcenserverkonfiguration bestimmt die Logik für die Behandlung fehlgeschlagener Gerätebefehle und Antwortnachrichten. Dies schließt Versuche ein, Nachrichten erneut zu versuchen, wobei dieselbe ReferenceID für den Befehl verwendet wurde.

Anforderungen an den C2C-Anschluss für die Ausführung von Gerätebefehlen

In der folgenden Liste sind die Anforderungen für Ihren C2C-Anschluss aufgeführt, um eine erfolgreiche Ausführung von Gerätebefehlen zu ermöglichen.

- Der C2C-Konnektor Lambda kann `AWS.SendCommand` Betriebsanforderungsnachrichten von verwalteten Integrationen für AWS IoT Device Management verarbeiten.
- Ihr C2C-Konnektor muss die an Ihren Ressourcenserver gesendeten Befehle verfolgen und ihnen die entsprechende `TraceId` zuordnen.
- Sie können verwaltete Integrationen für AWS IoT Device Management Service APIs über Sigv4 aufrufen, indem Sie die AWS Anmeldeinformationen verwenden, die für die Registrierung des C2C-Connectors AWS-Konto verwendet wurden.

1. Managed Integrations sendet einen Befehl an den Connector (siehe Schritt 4 im vorherigen Diagramm).

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

2. ACK-Befehl für den C2C-Anschluss (siehe Schritt 7 im vorherigen Diagramm, in dem der Connector ACK an die verwalteten Integrationen für AWS IoT Device Management Service sendet).

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

```
  }
```

3. Der Connector sendet das Device Command Response-Ereignis (siehe Schritt 11 im vorherigen Diagramm).

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

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

> **Note**
>
> Änderungen des Gerätestatus als Ergebnis einer Befehlsausführung werden erst in verwalteten Integrationen für AWS IoT Device Management berücksichtigt, wenn das entsprechende DEVICE_COMMAND_RESPONSE-Ereignis über die API empfangen wurde. SendConnectorEvent Das bedeutet, dass der Gerätestatus erst aktualisiert
  ```

wird, wenn Managed Integrations das Ereignis aus dem vorherigen Schritt 3 empfängt, unabhängig davon, ob Ihre Connector-Aufrufantwort erfolgreich ist oder nicht.

Interpretieren von „Endpunkten“, die in AWS enthalten sind. SendCommand Anfrage

Verwaltete Integrationen verwenden die bei der Geräteerkennung gemeldeten Gerätefunktionen, um zu ermitteln, welche Befehle ein Gerät annehmen kann. Jede Gerätefunktion wird durch AWS Implementierungen des Matter Data Model modelliert. Somit werden alle eingehenden Befehle aus dem Feld „Befehle“ innerhalb eines bestimmten Clusters abgeleitet. Es liegt in der Verantwortung Ihres Konnektors, das Feld „Endpunkte“ zu analysieren, den entsprechenden Matter-Befehl zu ermitteln und ihn so zu übersetzen, dass der richtige Befehl das Gerät erreicht. In der Regel bedeutet dies, das Matter-Datenmodell in die entsprechenden API-Anfragen zu übersetzen.

Nachdem der Befehl ausgeführt wurde, ermittelt Ihr Konnektor, welche `Attribute`, die durch die AWS Implementierungen des Matter-Datenmodells definiert wurden, sich dadurch geändert haben. Diese Änderungen werden dann über API-DEVICE_COMMAND_RESPONSE-Ereignisse, die mit der API gesendet werden, an verwaltete Integrationen für AWS IoT Device Management gemeldet. `SendConnectorEvent`

Betrachten Sie das Feld `endpoints`, das in der folgenden Beispielnutzlast enthalten ist: `AWS.SendCommand`

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

Aus diesem Objekt kann der Konnektor Folgendes ermitteln:

1. Legen Sie die Endpunkt- und Clusterinformationen fest:

a. Setzen Sie den Endpunkt `id` auf „1".

> **Note**
>
> Wenn ein Gerät mehrere Endpunkte definiert, also einen einzelnen Cluster (z. B.On/Off) can control multiple capabilities (i.e. turn a light on/off as well as turning a strobe on/off), wird diese ID verwendet, um den Befehl an die richtige Funktion weiterzuleiten.

b. Stellen Sie den Cluster `id` auf „0x0202" (Fan Control Cluster) ein.

2. Stellen Sie die Befehlsinformationen ein:

   a. Setzen Sie die Befehlskennung auf „0xff01" (Befehl „Status aktualisieren“, definiert von). AWS

   b. Aktualisieren Sie die enthaltenen Attributbezeichner mit den in der Anfrage angegebenen Werten.

3. Aktualisieren Sie das Attribut:

   a. Setzen Sie die Attribut-ID auf „0x0000" (FanMode Attribut des Fan Control Clusters).

   b. Setzen Sie den Attributwert auf „3" (Hohe Lüftergeschwindigkeit).

Managed Integrations hat zwei „benutzerdefinierte“ Befehlstypen definiert, die nicht unbedingt durch AWS Implementierungen des Matterdatenmodells definiert sind: Die Befehle ReadState und UpdateState . Um vom Thema definierte Clusterattribute abzurufen und festzulegen, senden verwaltete Integrationen Ihrem Connector eine `AWS.SendCommand` Anfrage mit einem Befehl, der sich auf UpdateState (id: 0xff01) oder ReadState (id: 0xff02) IDs bezieht, mit entsprechenden Attributparametern, die entweder aktualisiert oder gelesen werden müssen. Diese Befehle können für JEDEN Gerätetyp für Attribute aufgerufen werden, die in der entsprechenden Implementierung des Matter-Datenmodells als veränderbar (aktualisierbar) oder abrufbar (lesbar) festgelegt sind. AWS

Geräteereignisse mit der API senden SendConnectorEvent

Übersicht über vom Gerät ausgelöste Ereignisse

Die `SendConnectorEvent` API wird zwar verwendet, um asynchron auf `AWS.DiscoverDevices` Vorgänge zu `AWS.SendCommand` reagieren, sie wird aber auch verwendet, um verwaltete Integrationen über alle vom Gerät ausgelösten Ereignisse zu benachrichtigen. Geräteinitiierte

Ereignisse können als jedes Ereignis definiert werden, das von einem Gerät ohne einen vom Benutzer initiierten Befehl generiert wird. Zu diesen Geräteereignissen können unter anderem Änderungen des Gerätestatus, Bewegungserkennung, Akkuladestand und mehr gehören. Sie können diese Ereignisse mithilfe der `SendConnectorEvent` API mit der Operation DEVICE_EVENT an verwaltete Integrationen zurücksenden.

Im folgenden Abschnitt wird anhand einer Smart-Kamera, die zu Hause installiert ist, als Beispiel verwendet, um den Ablauf dieser Ereignisse näher zu erläutern:

Arbeitsablauf bei Geräteereignissen

1. Ihre Kamera erkennt Bewegungen und generiert daraufhin ein Ereignis, das an Ihren Ressourcenserver gesendet wird.
2. Ihr Ressourcenserver verarbeitet das Ereignis und sendet es an Ihren C2C-Anschluss.
3. Ihr Connector übersetzt dieses Ereignis in die verwaltete Integrationsschnittstelle für AWS IoT Device Management`DEVICE_EVENT`.
4. Ihr C2C-Connector sendet dieses Geräteereignis mithilfe der `SendConnectorEvent` API an verwaltete Integrationen, wobei der Vorgang auf „DEVICE_EVENT“ eingestellt ist.
5. Managed Integrations identifiziert den relevanten Kunden und leitet dieses Ereignis an den Kunden weiter.
6. Der Kunde erhält dieses Ereignis und zeigt es dem Benutzer anhand einer Benutzerkennung an.

Weitere Informationen zum `SendConnectorEvent` API-Betrieb finden Sie `SendConnectorEvent` im Referenzhandbuch für verwaltete Integrationen für AWS IoT Device Management API.

Anforderungen für vom Gerät ausgelöste Ereignisse

Im Folgenden sind einige Anforderungen für vom Gerät ausgelöste Ereignisse aufgeführt.

- Ihre C2C-Connector-Ressource sollte in der Lage sein, asynchrone Geräteereignisse von Ihrem Ressourcenserver zu empfangen
- Ihre C2C-Connector-Ressource sollte in der Lage sein, verwaltete Integrationen für AWS IoT Device Management Service-APIs über Sigv4 aufzurufen, wobei die AWS Anmeldeinformationen der für die Registrierung des C2C-Connectors AWS-Konto verwendeten Personen verwendet werden.

Das folgende Beispiel zeigt einen Connector, der ein vom Gerät ausgelöstes Ereignis über die API sendet: SendConnectorEvent

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

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

Aus dem folgenden Beispiel geht Folgendes hervor:

- Dies kommt vom Geräteendpunkt mit der ID gleich 1.
- Die Gerätefunktion, auf die sich dieses Ereignis bezieht, hat eine Cluster-ID von 0x0202 und bezieht sich auf den Themencluster Fan Control.
- Das Attribut, das geändert wurde, hat die ID 0x000 und bezieht sich auf das Fan Mode Enum innerhalb des Clusters. Es wurde auf den Wert 3 aktualisiert, was dem Wert High entspricht.
- Da `connectorId` es sich um einen Parameter handelt, der vom Cloud-Dienst bei der Erstellung zurückgegeben wird, müssen Connectors Abfragen verwenden GetCloudConnector und danach filtern`lambdaARN`. Der eigene Lambda `ARN` wird mithilfe der API abgefragt. `Lambda.get_function_url_config` Dadurch kann im `CloudConnectorId` Lambda dynamisch auf die zugegriffen und nicht wie zuvor statisch konfiguriert werden.

Implementieren Sie das AWS. DeactivateUser Betrieb

Übersicht über die Deaktivierung von Benutzern

Die Deaktivierung der bereitgestellten Benutzerzugriffstoken ist erforderlich, wenn ein Kunde sein AWS Kundenkonto löscht oder wenn ein Endbenutzer sein Konto im System vom AWS System des Kunden trennen möchte. In beiden Anwendungsfällen müssen verwaltete Integrationen diesen Arbeitsablauf mithilfe des C2C-Connectors erleichtern.

Die Abbildung unten zeigt, wie ein Endbenutzerkonto vom System getrennt wird

Arbeitsablauf zur Benutzerdeaktivierung

1. Der Benutzer leitet den Prozess der Trennung zwischen dem AWS Kundenkonto und dem Autorisierungsserver eines Drittanbieters ein, der dem C2C-Connector zugeordnet ist.
2. Der Kunde initiiert das Löschen der Benutzerverknüpfung über verwaltete Integrationen für AWS IoT Device Management.
3. Managed Integrations initiiert den Deaktivierungsprozess über eine Anfrage an Ihren Connector über die Bedienoberfläche. `AWS.DeactivateUser`
   - Das Zugriffstoken /user ist im Header der Anfrage enthalten.
4. Ihr C2C-Connector akzeptiert die Anfrage und ruft Ihren Autorisierungsserver auf, um das Token und jeden damit gewährten Zugriff zu widerrufen.
   - Beispielsweise sollten Ereignisse von einem nicht verknüpften Benutzerkonto nach der Ausführung nicht mehr an verwaltete Integrationen gesendet werden. `AWS.DeactivateUser`
5. Ihr Autorisierungsserver widerruft den Zugriff und sendet eine Antwort zurück an Ihren C2C-Connector.
6. Ihr C2C-Connector sendet verwalteten Integrationen für AWS IoT Device Management eine Bestätigung, dass das Zugriffstoken des Benutzers gesperrt wurde.
7. Managed Integrations löscht alle Ressourcen, die dem Endbenutzer gehören und mit Ihrem Ressourcenserver verknüpft waren.
8. Managed Integrations sendet eine ACK an den Kunden, in der angegeben wird, dass alle Verknüpfungen, die sich auf Ihr System beziehen, gelöscht wurden.

9. Der Kunde benachrichtigt den Endbenutzer darüber, dass sein Konto von Ihrer Plattform getrennt wurde.

AWS. DeactivateUser Anforderungen

- Die Lambda-Funktion des C2C-Konnektors empfängt eine Anforderungsnachricht von verwalteten Integrationen, um den Vorgang abzuwickeln. `AWS.DeactivateUser`
- Der C2C-Konnektor muss das bereitgestellte OAuth20.0-Token und das entsprechende Aktualisierungstoken des Benutzers auf Ihrem Autorisierungsserver widerrufen.

Im Folgenden finden Sie eine `AWS.DeactivateUser` Beispielanforderung, die Ihr Connector erhält:

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

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

Rufen Sie Ihren C2C-Connector auf

AWS Lambda ermöglicht ressourcenbasierte Richtlinien, um zu autorisieren, wer ein Lambda aufrufen kann. Da es sich bei verwalteten Integrationen für AWS IoT Device Management um eine handelt AWS-Service, müssen Sie verwalteten Integrationen erlauben, Ihren C2C-Connector-Lambda über die Ressourcenrichtlinie aufzurufen.

Fügen Sie Ihrem C2C-Connector Lambda eine Ressourcenrichtlinie mit mindestens den folgenden Mindestberechtigungen hinzu. Dies bietet verwaltete Integrationen mit Rechten zum Aufrufen von Lambda-Funktionen. Diese Richtlinie enthält einen `Condition` Schlüssel, mit dem Sie die Nutzbarkeit Ihrer Daten `connectorId` auf bestimmte Benutzer beschränken können.

JSON

```
{
  "Version":"2012-10-17",
  "Id": "default",
  "Statement": [
    {
      "Sid": "Your-Desired-Policy-ID",
      "Effect": "Allow",
      "Principal": {
        "Service": "iotmanagedintegrations.amazonaws.com"
      },
      "Action": "lambda:InvokeFunction",
      "Resource": "arn:aws:lambda:ca-central-1:444455556666:function:connector-
lambda-name",
      "Condition": {
        "StringEquals": {
          "aws:SourceArn": "arn:aws:iotmanagedintegrations:ca-
central-1:444455556666:account-association/account-association-id"
        }
      }
    }
  ]
}
```

Fügen Sie Ihrer IAM-Rolle Berechtigungen hinzu

Für den Aufruf aller verwalteten Integrationen APIs ist eine AWS SigV4-Authentifizierung erforderlich. Sigv4 ist ein Signaturprotokoll zur Authentifizierung von AWS API-Anfragen mit Ihren Anmeldeinformationen. AWS-Konto Die IAM-Rolle, die Sie zum Aufrufen der verwalteten Integrationen verwenden, APIs muss über die folgenden Berechtigungen verfügen, um die erfolgreich aufrufen zu können: APIs

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

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

Weitere Informationen zum Hinzufügen dieser Berechtigungen erhalten Sie von. Support

Weitere Ressourcen

Um Ihren C2C-Anschluss zu registrieren, benötigen Sie Folgendes:

- Der Lambda-ARN, der den Connector bezeichnet, den Sie registrieren möchten.

Testen Sie Ihren C2C-Anschluss manuell

Um Ihren C2C-Stecker manuell zu testen end-to-end, müssen Sie sowohl den Kunden als auch den Endbenutzer simulieren.

Sie benötigen die folgenden Ressourcen:

- Ein AWS Lambda ARN, der den Connector bezeichnet, den Sie testen möchten.
- Ein Testing OAuth 2.0-Benutzerkonto von Ihrer Cloud-Plattform.
- Ein Connector, der bei verwalteten Integrationen für AWS IoT Device Management registriert ist. Weitere Informationen finden Sie unter Verwenden Sie einen C2C-Connector (Cloud-to-Cloud).

Verwenden Sie einen C2C-Connector (Cloud-to-Cloud)

Ein C2C-Konnektor verwaltet die Übersetzung von Anfrage- und Antwortnachrichten und ermöglicht die Kommunikation zwischen verwalteten Integrationen und einer Cloud eines Drittanbieters. Er ermöglicht die einheitliche Steuerung verschiedener Gerätetypen, Plattformen und Protokolle, sodass Geräte von Drittanbietern integriert und verwaltet werden können.

Im folgenden Verfahren werden die Schritte zur Verwendung des C2C-Anschlusses aufgeführt.

Schritte zur Verwendung des C2C-Anschlusses:

1. CreateCloudConnector

Konfigurieren Sie einen Konnektor, um die bidirektionale Kommunikation zwischen Ihren verwalteten Integrationen und Clouds von Drittanbietern zu ermöglichen.

Geben Sie bei der Einrichtung des Connectors die folgenden Details an:

- Name: Wählen Sie einen aussagekräftigen Namen für den Connector.
- Beschreibung: Geben Sie eine kurze Zusammenfassung des Zwecks und der Funktionen des Connectors an.
- AWS Lambda ARN: Geben Sie den Amazon-Ressourcennamen (ARN) der AWS Lambda Funktion an, die den Connector mit Strom versorgt.

Erstellen und implementieren Sie eine AWS Lambda Funktion, die mit einem Drittanbieter kommuniziert APIs , um einen Konnektor zu erstellen. Rufen Sie als Nächstes die CreateCloudConnectorAPI in verwalteten Integrationen auf und stellen Sie die AWS Lambda Funktion ARN für die Registrierung bereit. Stellen Sie sicher, dass die AWS Lambda Funktion in demselben AWS Konto bereitgestellt wird, in dem Sie den Connector in verwalteten Integrationen erstellen. Ihnen wird eine eindeutige Connector-ID zugewiesen, um die Integration zu identifizieren.

Beispiel für eine CreateCloudConnector API-Anfrage und -Antwort:

```
Request:

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

Response:

{
    "Id": "string"
```

```
}
```

Ablauf der Erstellung:

> **Note**
>
> Verwenden Sie für dieses Verfahren nach ListCloudConnectors APIs Bedarf DeleteCloudConnector,, und. GetCloudConnectorUpdateCloudConnector

2. CreateConnectorDestination

Konfigurieren Sie Destinations so, dass sie die Einstellungen und Authentifizierungsdaten bereitstellen, die Connectors benötigen, um sichere Verbindungen mit Clouds von Drittanbietern herzustellen. Verwenden Sie Destinations, um Ihre Authentifizierungsdaten von Drittanbietern bei verwalteten Integrationen zu registrieren, z. B. OAuth 2.0-Autorisierungsdetails, einschließlich der Autorisierungs-URL, des Authentifizierungsschemas und des Speicherorts der Anmeldeinformationen darin AWS Secrets Manager.

Voraussetzungen

Bevor Sie eine erstellen ConnectorDestination, müssen Sie:

- Rufen Sie die CreateCloudConnectorAPI auf, um einen Connector zu erstellen. Die ID, die die Funktion zurückgibt, wird im CreateConnectorDestinationAPI-API-Aufruf verwendet.
- Rufen Sie die `tokenUrl` für die 3P-Plattform des Konnektors ab. (Sie können einen AuthCode gegen ein AccessToken austauschen).
- Rufen Sie die AuthURL für die 3P-Plattform des Connectors ab. (Endbenutzer können sich mit ihrem Benutzernamen und Passwort authentifizieren).
- Verwenden Sie das `clientId` und `clientSecret` (von der 3P-Plattform) im Secret Manager Ihres Kontos.

Beispiel für eine CreateConnectorDestination API-Anfrage und -Antwort:

```
Request:

{
    "Name": "CreateConnectorDestination",
```

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

Response:

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

Ablauf bei der Erstellung von Cloud-Zielen:

> **Note**
>
> Verwenden Sie GetCloudConnector, UpdateCloudConnectorDeleteCloudConnector, und nach ListCloudConnectors APIs Bedarf für dieses Verfahren.

3. CreateAccountAssociation

Zuordnungen stellen die Beziehungen zwischen Cloud-Konten von Drittanbietern von Endbenutzern und einem Connector-Ziel dar. Nachdem Sie eine Zuordnung erstellt und

Endbenutzer mit verwalteten Integrationen verknüpft haben, ist der Zugriff auf ihre Geräte über eine eindeutige Zuordnungs-ID möglich. Diese Integration ermöglicht drei Hauptfunktionen: das Erkennen von Geräten, das Senden von Befehlen und das Empfangen von Ereignissen.

Voraussetzungen

Bevor AccountAssociationSie eine erstellen, müssen Sie die folgenden Schritte ausführen:

- Rufen Sie die CreateConnectorDestinationAPI auf, um ein Ziel zu erstellen. Die ID, die die Funktion zurückgibt, wird im CreateAccountAssociationAPI-Aufruf verwendet.
- Rufen Sie die CreateAccountAssociationAPI auf.

Beispiel für eine CreateAccountAssociation API-Anfrage und -Antwort:

```
Request:

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

Response:

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

> **Note**
>
> Verwenden Sie für dieses Verfahren nach ListCloudConnectors APIs Bedarf DeleteCloudConnector,, und. GetCloudConnectorUpdateCloudConnector

An AccountAssociationhat einen Status, der von GetAccountAssociationund abgefragt wird. ListAccountAssociations APIs Dies APIs zeigt den Status des

Vereins. Die StartAccountAssociationRefreshAPI ermöglicht die Aktualisierung eines AccountAssociationStatus, wenn sein Aktualisierungstoken abläuft.

4. Erkennung von Geräten

   Jedes verwaltete Objekt ist mit gerätespezifischen Details wie der Seriennummer und einem Datenmodell verknüpft. Das Datenmodell beschreibt die Funktionalität des Geräts und gibt an, ob es sich um eine Glühbirne, einen Schalter, einen Thermostat oder einen anderen Gerätetyp handelt. Um ein 3P-Gerät zu erkennen und ein ManagedThing für das 3P-Gerät zu erstellen, müssen Sie die folgenden Schritte nacheinander ausführen.

   a. Rufen Sie die StartDeviceDiscoveryAPI auf, um den Geräteerkennungsprozess zu starten.

      Beispiel für eine StartDeviceDiscovery API-Anfrage und -Antwort:

      ```
      Request:

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

      Response:

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

   b. Rufen Sie die GetDeviceDiscoveryAPI auf, um den Status des Erkennungsprozesses zu überprüfen.

   c. Rufen Sie die ListDiscoveredDevicesAPI auf, um die erkannten Geräte aufzulisten.

      Beispiel für eine ListDiscoveredDevices API-Anfrage und -Antwort:

      ```
      Request:

      //Empty body

      Response:
      ```

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

d. Rufen Sie die CreateManagedThingAPI auf, um die Geräte aus der Discovery-Liste auszuwählen, die in verwaltete Integrationen importiert werden sollen.

   Beispiel für eine CreateManagedThing API-Anfrage und -Antwort:

```
Request:

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

Response:

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

e. Rufen Sie die GetManagedThingAPI auf, um diese neu erstellte `managedThing` Datei anzuzeigen. Der Status wird sein. `UNASSOCIATED`

f. Rufen Sie die RegisterAccountAssociationAPI auf, um dies einer bestimmten `managedThing` `accountAssociation` Person zuzuordnen. Am Ende einer erfolgreichen RegisterAccountAssociationAPI `managedThing` ändert sich der `ASSOCIATED` Status.

Beispiel für eine RegisterAccountAssociation API-Anfrage und -Antwort:

```
Request:

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

Response:

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

5. Senden Sie einen Befehl an das 3P-Gerät

Um ein neu integriertes Gerät zu steuern, verwenden Sie die SendManagedThingCommandAPI mit der zuvor erstellten Zuordnungs-ID und einer Steueraktion, die auf der vom Gerät unterstützten Funktion basiert. Der Connector verwendet gespeicherte Anmeldeinformationen aus dem Kontoverknüpfungsprozess, um sich bei der Drittanbieter-Cloud zu authentifizieren und den entsprechenden API-Aufruf für den Vorgang aufzurufen.

Beispiel für eine SendManagedThingCommand API-Anfrage und -Antwort:

```
Request:

{
    "AccountAssociationId": "string",
    "ConnectorAssociationId": "string",
    "Endpoints": [
       {
          "capabilities": [
             {
                "actions": [
```

```
                    {
                       "actionTraceId": "string",
                       "name": "string",
                       "parameters": JSON value,
                       "ref": "string"
                    }
                ],
                "id": "string",
                "name": "string",
                "version": "string"
             }
          ],
          "endpointId": "string"
       }
    ]
}

Response:

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

Befehl an den 3P-Gerätefluss senden:

6. Connector sendet Ereignisse an verwaltete Integrationen

   Die SendConnectorEventAPI erfasst vier Ereignistypen vom Connector bis hin zu verwalteten Integrationen, die durch die folgenden Aufzählungswerte für den Parameter Operation Type dargestellt werden:

   - DEVICE_COMMAND_RESPONSE: Die asynchrone Antwort, die der Connector als Antwort auf einen Befehl sendet.
   - DEVICE_DISCOVERY: Als Reaktion auf einen Geräteerkennungsprozess sendet der Connector die Liste der erkannten Geräte an verwaltete Integrationen und verwendet die API. SendConnectorEvent
   - DEVICE_EVENT: Sendet die empfangenen Geräteereignisse.
   - DEVICE_COMMAND_REQUEST: Vom Gerät initiierte Befehlsanfragen. Zum Beispiel WebRTC-Workflows.

Der Connector kann mit einem optionalen `userId` Parameter auch Geräteereignisse mithilfe der SendConnectorEventAPI weiterleiten.

- Für Geräteereignisse mit einem`userId`:

  Beispiel für eine SendConnectorEvent API-Anfrage und -Antwort:

```
Request:

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

Response:

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

- Für Geräteereignisse ohne`userId`:

  Beispiel für eine SendConnectorEvent API-Anfrage und -Antwort:

```
Request:

{
    "Operation": "DEVICE_EVENT",
    "OperationVersion": "1.0",
```

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

Response:

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

Verwenden Sie den Abmeldemechanismus, um die Verknüpfung zwischen einer bestimmten `managedThing` und einer Kontoverknüpfung zu entfernen:

Beispiel für eine DeregisterAccountAssociation API-Anfrage und -Antwort:

```
Request:

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

Response:

HTTP/1.1 200 // Empty body
```

Ereignisablauf senden:

7. Aktualisieren Sie den Connector-Status auf „Gelistet“, um ihn für andere Kunden mit verwalteten Integrationen sichtbar zu machen

Standardmäßig sind Konnektoren privat und nur für das AWS Konto sichtbar, mit dem sie erstellt wurden. Sie können wählen, ob ein Connector für andere Kunden mit verwalteten Integrationen sichtbar sein soll.

Um Ihren Connector mit anderen Benutzern zu teilen, verwenden Sie die Option Sichtbar machen AWS-Managementkonsole auf der Seite mit den Connector-Details, um Ihre Connector-ID AWS zur Überprüfung einzureichen. Nach der Genehmigung steht der Connector allen Benutzern verwalteter Integrationen in derselben AWS-Region Version zur Verfügung. Darüber hinaus können Sie den Zugriff auf ein bestimmtes AWS Konto einschränken, IDs indem Sie die Zugriffsrichtlinie für die dem Connector zugeordnete AWS Lambda Funktion ändern. Um sicherzustellen, dass Ihr Connector von anderen Kunden verwendet werden kann, verwalten Sie die IAM-Zugriffsberechtigungen für Ihre Lambda-Funktion von anderen AWS Konten auf Ihren sichtbaren Connector.

Lesen Sie die AWS-Service Bedingungen und Richtlinien Ihrer Organisation, die die gemeinsame Nutzung von Connectoren und die Zugriffsberechtigungen regeln, bevor Sie Connectors für andere Kunden mit verwalteten Integrationen sichtbar machen.

Hub SDK für verwaltete Integrationen

In den Themen in diesem Abschnitt erfahren Sie, wie Sie IoT-Hub-Geräte mithilfe des Managed Integrations Hub SDK einbinden und steuern. Weitere Informationen zum Endgeräte-SDK für verwaltete Integrationen finden Sie im. Verwaltete Integrationen Endgeräte-SDK

Hub-SDK-Architektur

Onboarding von Geräten

Informieren Sie sich, wie die Hub SDK-Komponenten das Onboarding von Geräten unterstützen, bevor Sie mit der Arbeit mit verwalteten Integrationen beginnen. In diesem Abschnitt werden die wesentlichen Architekturkomponenten behandelt, die Sie für das Onboarding von Geräten benötigen, einschließlich der Art und Weise, wie der Core Provisioner und die protokollspezifischen Plug-ins zusammenarbeiten, um die Geräteauthentifizierung, Kommunikation und Benutzereinrichtung zu handhaben.

Hub SDK-Komponenten für das Onboarding von Geräten

SDK-Komponenten

- Core-Provisioner
- Protokollspezifische Provisioner-Plugins
- Protokollspezifische Middleware

Core-Provisioner

Der Core Provisioner ist die zentrale Komponente, die das Onboarding von Geräten in Ihrer IoT-Hub-Bereitstellung orchestriert. Er koordiniert die gesamte Kommunikation zwischen verwalteten Integrationen und Ihren protokollspezifischen Provisioner-Plugins und gewährleistet so ein sicheres und zuverlässiges Geräte-Onboarding. Wenn Sie ein Gerät einbinden, übernimmt der Core-Provisioner den Authentifizierungsablauf, verwaltet das MQTT-Messaging und verarbeitet Geräteanfragen über folgende Funktionen:

MQTT-Verbindung

Stellt Verbindungen mit dem MQTT-Broker für die Veröffentlichung und das Abonnieren von Cloud-Themen her.

Nachrichtenwarteschlange und Handler

Verarbeitet eingehende Anfragen zum Hinzufügen und Entfernen von Geräten nacheinander.

Schnittstelle für das Protokoll-Plugin

Funktioniert mit protokollspezifischen Provisioner-Plugins für das Onboarding von Geräten, indem die Authentifizierungs- und Funkverbindungsmodi verwaltet werden.

Hub-SDK-Client APIs

Empfangen Sie Berichte zur Gerätefähigkeit von protokollspezifischen CDMB-Plugins und leiten Sie sie an verwaltete Integrationen weiter.

Protokollspezifische Provisioner-Plugins

Protokollspezifische Provisioner-Plugins sind Bibliotheken, die das Onboarding von Geräten für verschiedene Kommunikationsprotokolle verwalten. Jedes Plugin übersetzt Befehle vom Core-Provisioner in protokollspezifische Aktionen für Ihre IoT-Geräte. Diese Plugins führen Folgendes aus:

- Protokollspezifische Middleware-Initialisierung
- Konfiguration des Funkverbindungsmodus auf der Grundlage von Core-Provisioner-Anfragen
- Entfernung von Geräten durch Middleware-API-Aufrufe

Protokollspezifische Middleware

Protokollspezifische Middleware fungiert als Übersetzungsschicht zwischen Ihren Geräteprotokollen und verwalteten Integrationen. Diese Komponente verarbeitet die Kommunikation in beide Richtungen — sie empfängt Befehle von den Provisioner-Plug-ins und sendet sie an Protokollstapel, sammelt aber auch Antworten von Geräten und leitet sie zurück durch das System.

Abläufe beim Onboarding von Geräten

Überprüfen Sie die Reihenfolge der Vorgänge, die beim Onboarding von Geräten mithilfe des Hub-SDK ausgeführt werden. In diesem Abschnitt wird gezeigt, wie die Komponenten während

des Onboarding-Prozesses interagieren, und es werden die unterstützten Onboarding-Methoden beschrieben.

Onboarding-Abläufe

- Einfache Einrichtung (SS)
- Einrichtung ohne Benutzereingriff (ZTS)
- Benutzergeführte Einrichtung (UGS)

Einfache Einrichtung (SS)

Der Endbenutzer schaltet das IoT-Gerät ein und scannt seinen QR-Code mithilfe der Anwendung des Geräteherstellers. Das Gerät wird dann in der Managed Integrations Cloud registriert und stellt eine Verbindung zum IoT-Hub her.

Einrichtung ohne Benutzereingriff (ZTS)

Das Zero-Touch-Setup (ZTS) optimiert das Onboarding von Geräten, indem das Gerät vorab in der Lieferkette zugewiesen wird. Anstatt dass Endbenutzer beispielsweise den QR-Code des Geräts scannen, wird dieser Schritt früher abgeschlossen, um Geräte vorab mit Kundenkonten zu verknüpfen. Dieser Schritt kann beispielsweise im Versandzentrum abgeschlossen werden.

Wenn der Endbenutzer das Gerät empfängt und einschaltet, registriert es sich automatisch in der Managed Integrations Cloud und stellt eine Verbindung zum IoT-Hub her, ohne dass zusätzliche Einrichtungsaktionen erforderlich sind.

Benutzergeführte Einrichtung (UGS)

Der Endbenutzer schaltet das Gerät ein und folgt interaktiven Schritten, um es in verwaltete Integrationen zu integrieren. Dies kann das Drücken einer Taste am IoT-Hub, die Verwendung einer App des Geräteherstellers oder das Drücken von Tasten sowohl am Hub als auch am Gerät beinhalten. Sie können diese Methode verwenden, wenn die einfache Einrichtung fehlschlägt.

Steuerung der Geräte

Verwaltete Integrationen kümmern sich um die Geräteregistrierung, Befehlsausführung und Steuerung. Mithilfe der herstellerunabhängigen und protokollunabhängigen Geräteverwaltung können Sie Endbenutzererlebnisse schaffen, ohne sich mit gerätespezifischen Protokollen auskennen zu müssen.

Mit der Gerätesteuerung können Sie Gerätestatus wie die Helligkeit von Glühbirnen oder die Position der Tür einsehen und ändern. Die Funktion gibt Ereignisse für Statusänderungen aus, die Sie für Analysen, Regeln und Überwachung verwenden können.

Schlüsselfeatures

Gerätestatus ändern oder lesen

Geräteattribute basierend auf Gerätetypen anzeigen und ändern. Sie können auf Folgendes zugreifen:

- Gerätestatus: Aktuelle Geräteattributwerte
- Konnektivitätsstatus: Status der Erreichbarkeit des Geräts
- Gesundheitsstatus: Systemwerte wie Akkuladestand und Signalstärke (RSSI)

Benachrichtigung über eine Änderung des Zustands

Lassen Sie sich über Ereignisse informieren, wenn sich Geräteattribute oder Verbindungsstatus ändern, z. B. bei Helligkeitsanpassungen bei Glühbirnen oder beim Status von Türschlössern.

Offlinemodus

Geräte kommunizieren auch ohne Internetverbindung mit anderen Geräten auf demselben IoT-Hub. Der Gerätestatus wird mit der Cloud synchronisiert, wenn die Konnektivität wieder hergestellt wird.

Statussynchronisierung

Verfolgen Sie Statusänderungen aus verschiedenen Quellen, Apps von Geräteherstellern und manuelle Geräteanpassungen.

Informieren Sie sich über die Komponenten und Prozesse des Hub SDK, die Sie zur Steuerung von Geräten über verwaltete Integrationen benötigen. In diesem Thema wird beschrieben, wie der Edge Agent, Common Data Model Bridge (CDMB) und protokollspezifische Plug-ins zusammenarbeiten,

um Gerätebefehle zu verarbeiten, Gerätestatus zu verwalten und Antworten protokollübergreifend zu verarbeiten.

Abläufe der Gerätesteuerung

Das folgende Diagramm veranschaulicht den Ablauf der end-to-end Gerätesteuerung, indem es beschreibt, wie ein Endbenutzer einen ZigBee-Smart Plug einschaltet.

Hub-SDK-Komponenten für die Gerätesteuerung

Die Hub SDK-Architektur verwendet die folgenden Komponenten, um Gerätesteuerungsbefehle in Ihrer IoT-Implementierung zu verarbeiten und weiterzuleiten. Jede Komponente spielt eine bestimmte Rolle bei der Übersetzung von Cloud-Befehlen in Geräteaktionen, der Verwaltung von Gerätestatus und der Verarbeitung von Antworten. In den folgenden Abschnitten wird detailliert beschrieben, wie diese Komponenten in Ihrer Bereitstellung zusammenarbeiten:

Das Hub-SDK besteht aus den folgenden Komponenten und erleichtert das Onboarding und die Steuerung von Geräten auf IoT-Hubs.

Primäre Komponenten:

Edge-Agent

: Fungiert als Gateway zwischen dem IoT-Hub und verwalteten Integrationen.

Gemeinsame Datenmodellbrücke (CDMB)

: Übersetzt zwischen dem AWS Datenmodell und lokalen Protokolldatenmodellen wie Z-Wave und Zigbee. Es enthält ein Kern-CDMB und protokollspezifische CDMB-Plugins.

Bereitsteller

: Kümmert sich um die Geräteerkennung und das Onboarding. Es umfasst einen Core-Provisioner und protokollspezifische Provisioner-Plugins für protokollspezifische Onboarding-Aufgaben.

Sekundäre Komponenten

Onboarding von Hubs

: Stellt dem Hub Client-Zertifikate und Schlüssel für eine sichere Cloud-Kommunikation zur Verfügung.

MQTT-Proxy

Stellt MQTT-Verbindungen zur Managed Integrations Cloud bereit.

Logger

Schreibt Protokolle lokal oder in die Cloud für verwaltete Integrationen.

Installieren und validieren Sie das Hub-SDK für verwaltete Integrationen

Wählen Sie zwischen den folgenden Bereitstellungsmethoden, um das Managed Integrations Hub SDK auf Ihren Geräten zu installieren —AWS IoT Greengrass für die automatisierte Bereitstellung oder für eine manuelle Skriptinstallation. In diesem Abschnitt werden die Schritte zur Einrichtung und Validierung für beide Ansätze beschrieben.

Bereitstellungsmethoden

- Installieren Sie das Hub-SDK mit AWS IoT Greengrass
- Stellen Sie das Hub-SDK mit einem Skript bereit
- Stellen Sie das Hub-SDK mit systemd bereit

Installieren Sie das Hub-SDK mit AWS IoT Greengrass

Stellen Sie die Hub-SDK-Komponenten für verwaltete Integrationen mithilfe von AWS IoT Greengrass (Java-Version) für Ihre Geräte bereit.

> **Note**
>
> Sie müssen es bereits eingerichtet haben und sich damit AWS IoT Greengrass auskennen. Weitere Informationen finden Sie unter Was ist AWS IoT Greengrass in der Dokumentation des AWS IoT Greengrass Entwicklerhandbuchs enthalten.

Der AWS IoT Greengrass Benutzer muss berechtigt sein, die folgenden Verzeichnisse zu ändern:

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

- `/data/ace/kvstorage`

Themen

- Stellen Sie Komponenten lokal bereit
- Bereitstellung in der Cloud
- Überprüfen Sie die Hub-Bereitstellung
- Überprüfen Sie den CDMB-Betrieb
- Überprüfen Sie den Betrieb von LPW-Provisioner

Stellen Sie Komponenten lokal bereit

Verwenden Sie die CreateDeployment AWS IoT Greengrass API auf Ihrem Gerät, um die Hub SDK-Komponenten bereitzustellen. Die Versionsnummern sind nicht statisch und können je nach der Version, die Sie gerade verwenden, variieren. Verwenden Sie das folgende Format für**`version`**: com.Amazon.io TManagedIntegrationsDevice. 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"
```

Bereitstellung in der Cloud

Folgen Sie den Anweisungen im AWS IoT Greengrass Entwicklerhandbuch, um die folgenden Schritte durchzuführen:

1. Laden Sie Artefakte auf Amazon S3 hoch.
2. Aktualisieren Sie die Rezepte so, dass sie den Speicherort des Amazon S3 S3-Artefakts enthalten.
3. Erstellen Sie eine Cloud-Bereitstellung für das Gerät für die neuen Komponenten.

Überprüfen Sie die Hub-Bereitstellung

Bestätigen Sie die erfolgreiche Bereitstellung, indem Sie Ihre Konfigurationsdatei überprüfen. Öffnen Sie die `/data/aws/iotmi/config/iotmi_config.json` Datei und überprüfen Sie, ob der Status auf `PROVISIONED` eingestellt ist.

Überprüfen Sie den CDMB-Betrieb

Suchen Sie in der Protokolldatei nach CDMB-Startmeldungen und nach erfolgreicher Initialisierung. Der *`logs file`* Speicherort kann je nach Installationsort AWS IoT Greengrass variieren.

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

Beispiel

```
[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
```

Überprüfen Sie den Betrieb von LPW-Provisioner

Suchen Sie in der Protokolldatei nach Startmeldungen von LPW-Provisioner und nach erfolgreicher Initialisierung. Der *`logs file`* Speicherort kann je nach Installationsort variieren. AWS IoT Greengrass

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

Beispiel

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

Stellen Sie das Hub-SDK mit einem Skript bereit

Stellen Sie die Hub-SDK-Komponenten für verwaltete Integrationen manuell mithilfe von Installationsskripten bereit und validieren Sie dann die Bereitstellung. In diesem Abschnitt werden die Schritte zur Ausführung des Skripts und der Überprüfungsprozess beschrieben.

Themen

Bereiten Sie Ihre Umgebung vor

Gehen Sie wie folgt vor, bevor Sie das SDK-Installationsskript ausführen:

1. Erstellen Sie einen Ordner mit dem Namen `middleware` innerhalb des `artifacts` Ordners.
2. Kopieren Sie Ihre Hub-Middleware-Dateien in den `middleware` Ordner.
3. Führen Sie die Initialisierungsbefehle aus, bevor Sie das SDK starten.

> **Important**
>
> Wiederholen Sie die Initialisierungsbefehle nach jedem Hub-Neustart.

```
#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
```

Führen Sie das Hub-SDK-Skript aus

Navigieren Sie zum Artefaktverzeichnis und führen Sie das `start_iotmi_sdk.sh` Skript aus. Dieses Skript startet die Hub-SDK-Komponenten in der richtigen Reihenfolge. Überprüfen Sie die folgenden Beispielprotokolle, um sicherzustellen, dass der Start erfolgreich war:

> **Note**
>
> Protokolle für alle laufenden Komponenten befinden sich im `artifacts/logs` Ordner.

```
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----
```

Überprüfen Sie die Hub-Bereitstellung

Vergewissern Sie sich, dass das `iot_provisioning_state` Feld in auf eingestellt `/data/aws/iotmi/config/iotmi_config.json` ist`PROVISIONED`.

Überprüfen Sie den Betrieb des Agenten

Suchen Sie in der Protokolldatei nach Startmeldungen des Agenten und nach erfolgreicher Initialisierung.

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

Beispiel

```
[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**
>
> Überprüfen Sie, ob die `iotmi.db` Datenbank in Ihrem `artifacts` Verzeichnis vorhanden ist.

Überprüfen Sie den LPW-Provisioner-Betrieb

Überprüfen Sie die Protokolldatei auf `LPW-Provisioner` Startmeldungen und eine erfolgreiche Initialisierung.

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

Der folgende Code zeigt ein Beispiel dafür.

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

Stellen Sie das Hub-SDK mit systemd bereit

> **Important**
>
> Folgen Sie den Anweisungen `readme.md` im `hubSystemdSetup` Verzeichnis der Datei release.tgz für die neuesten Updates.

In diesem Abschnitt werden die Skripts und Prozesse für die Bereitstellung und Konfiguration von Diensten auf einem Linux-basierten Hub-Gerät beschrieben.

Übersicht

Der Bereitstellungsprozess besteht aus zwei Hauptskripten:

- `copy_to_hub.sh`: Wird auf dem Host-Computer ausgeführt, um die erforderlichen Dateien auf den Hub zu kopieren
- `setup_hub.sh`: Wird auf dem Hub ausgeführt, um die Umgebung zu konfigurieren und Dienste bereitzustellen

Verwaltet außerdem `systemd/deploy_iotshd_services_on_hub.sh` die Prozess-Bootstrap-Sequenz und die Verwaltung von Prozessberechtigungen und wird automatisch ausgelöst durch`setup_hub.sh`.

Voraussetzungen

Die aufgeführten Voraussetzungen sind für eine erfolgreiche Bereitstellung erforderlich.

- Der Systemd-Dienst ist auf dem Hub verfügbar
- SSH-Zugriff auf das Hub-Gerät

- Sudo-Rechte auf dem Hub-Gerät
- `scp`Auf dem Host-Computer installiertes Hilfsprogramm
- `sed`Auf dem Host-Computer installiertes Hilfsprogramm
- Auf dem Host-Computer installiertes Hilfsprogramm zum Entpacken

Dateistruktur

Die Dateistruktur ist so konzipiert, dass sie die Organisation und Verwaltung der verschiedenen Komponenten erleichtert und so einen effizienten Zugriff und eine effiziente Navigation auf den Inhalt ermöglicht.

```
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
```

In der TGZ-Datei der SDK-Version lautet die gesamte Dateistruktur wie folgt:

```
IoT-managed-integrations-Hub-SDK-aarch64-v1.0.0.tgz
###package/
    ###greengrass/
        ###artifacts/
        ###recipes/
    ###hubSystemdSetup/
        ### REAME.md
        ### copy_to_hub.sh
        ### setup_hub.sh
        ### iotshd_config.json  # Sample configuration file
        ### local_certs/  # Directory for DHA certificates
        ### systemd/
            ### *.service.template  # Systemd service templates
            ### deploy_iotshd_services_on_hub.sh
```

Erstes Einrichten

Extrahieren Sie das SDK-Paket

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

Navigieren Sie zum entpackten Verzeichnis und bereiten Sie das Paket vor:

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

Fügen Sie Gerätekonfigurationsdateien hinzu

Folgen Sie den beiden aufgeführten Schritten, um die Gerätekonfigurationsdateien zu erstellen, und kopieren Sie sie auf den Hub.

1. Fügen Sie Gerätekonfigurationsdateien hinzu, um die benötigten Gerätekonfigurationsdateien zu erstellen. Das SDK verwendet diese Datei für seine Funktion.
2. Kopieren Sie die Konfigurationsdateien, um die erstellten Konfigurationsdateien auf den Hub zu kopieren.

Kopieren Sie Dateien auf den Hub

Führen Sie das Bereitstellungsskript von Ihrem Host-Computer aus:

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

Example Beispiel

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

Dies kopiert:

- Die Paketdatei (auf dem Hub in package.zip umbenannt)
- Konfigurationsdateien

- Zertifikate
- Systemd-Dienstdateien

Hub einrichten

Nachdem die Dateien kopiert wurden, stellen Sie eine SSH-Verbindung zum Hub her und führen Sie das Setup-Skript aus:

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

Benutzer- und Gruppenkonfigurationen

Standardmäßig verwenden wir den Benutzer-Hub und den Gruppen-Hub für die SDK-Komponenten. Es gibt mehrere Möglichkeiten, sie zu konfigurieren:

- Verwenden Sie einen benutzerdefinierten Benutzer/eine benutzerdefinierte Gruppe:

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

- Erstellen Sie sie manuell, bevor Sie das Setup-Skript ausführen:

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

- Fügen Sie die Befehle hinzu`setup_hub.sh`.

Dienste verwalten

Um alle Dienste neu zu starten, führen Sie das folgende Skript vom Hub aus aus:

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

Das Setup-Skript erstellt die erforderlichen Verzeichnisse, legt die entsprechenden Berechtigungen fest und stellt die Dienste automatisch bereit. Wenn Sie SSH/SCP nicht verwenden, müssen Sie es an Ihre spezifische Bereitstellungsmethode anpassen`copy_to_hub.sh`. Stellen Sie vor der Bereitstellung sicher, dass alle Zertifikatsdateien und Konfigurationen ordnungsgemäß eingerichtet sind.

Integrieren Sie Ihre Hubs in verwaltete Integrationen

Richten Sie Ihre Hub-Geräte für die Kommunikation mit verwalteten Integrationen ein, indem Sie die erforderliche Verzeichnisstruktur, Zertifikate und Gerätekonfigurationsdateien konfigurieren. In diesem Abschnitt wird beschrieben, wie die Komponenten des Hub-Onboarding-Subsystems zusammenarbeiten, wo Zertifikate und Konfigurationsdateien gespeichert werden, wie die Gerätekonfigurationsdatei erstellt und geändert wird und wie der Hub-Bereitstellungsprozess abgeschlossen wird.

Subsystem für das Onboarding des Hubs

Das Hub-Onboarding-Subsystem verwendet diese Kernkomponenten, um die Gerätebereitstellung und -konfiguration zu verwalten:

Hub-Onboarding-Komponente

Verwaltet den Hub-Onboarding-Prozess, indem es den Hub-Status, den Bereitstellungsansatz und die Authentifizierungsmaterialien koordiniert.

Gerätekonfigurationsdatei

Speichert wichtige Hub-Konfigurationsdaten auf dem Gerät, darunter:

- Status der Gerätebereitstellung (bereitgestellt oder nicht bereitgestellt)
- Zertifikate und wichtige Speicherorte
- Authentifizierungsinformationen Andere SDK-Prozesse, wie der MQTT-Proxy, verweisen auf diese Datei, um den Hub-Status und die Verbindungseinstellungen zu ermitteln.

Schnittstelle für den Zertifikatshandler

Stellt eine Utility-Schnittstelle zum Lesen und Schreiben von Gerätezertifikaten und Schlüsseln bereit. Sie können diese Schnittstelle implementieren, um mit folgenden Komponenten zu arbeiten:

- Speicher im Dateisystem
- Hardware-Sicherheitsmodule (HSM)
- Vertrauenswürdige Plattformmodule (TPM)
- Maßgeschneiderte sichere Speicherlösungen

MQTT-Proxykomponente

Verwaltet die device-to-cloud Kommunikation mit:

- Bereitgestellte Client-Zertifikate und Schlüssel
- Informationen zum Gerätestatus aus der Konfigurationsdatei
- MQTT-Verbindungen zu verwalteten Integrationen

Das folgende Diagramm beschreibt die Architektur des Hub-Onboarding-Subsystems und seine Komponenten. Wenn Sie diese Komponente nicht verwenden AWS IoT Greengrass, können Sie diese Komponente des Diagramms ignorieren.

Einrichtung des Hub-Onboardings

Führen Sie diese Einrichtungsschritte für jedes Hub-Gerät durch, bevor Sie mit dem Onboarding-Prozess für die Flottenbereitstellung beginnen. In diesem Abschnitt wird beschrieben, wie verwaltete Dinge erstellt, Verzeichnisstrukturen eingerichtet und die erforderlichen Zertifikate konfiguriert werden.

Einrichtungsschritte

- Schritt 1: Registrieren Sie einen benutzerdefinierten Endpunkt
- Schritt 2: Erstellen Sie ein Bereitstellungsprofil
- Schritt 3: Erstellen Sie eine verwaltete Sache (Flottenbereitstellung)
- Schritt 4: Erstellen Sie die Verzeichnisstruktur
- Schritt 5: Fügen Sie dem Hub-Gerät Authentifizierungsmaterialien hinzu
- Schritt 6: Erstellen Sie die Gerätekonfigurationsdatei
- Schritt 7: Kopieren Sie die Konfigurationsdatei auf Ihren Hub

Schritt 1: Registrieren Sie einen benutzerdefinierten Endpunkt

Erstellen Sie einen dedizierten Kommunikationsendpunkt, über den Ihre Geräte Daten mit verwalteten Integrationen austauschen. Dieser Endpunkt stellt einen sicheren Verbindungspunkt für alle device-to-cloud Nachrichten her, einschließlich Gerätebefehle, Statusaktualisierungen und Benachrichtigungen.

Um einen Endpunkt zu registrieren

- Verwenden Sie die RegisterCustomEndpointAPI, um einen Endpunkt für die device-to-managed Integrationskommunikation zu erstellen.

RegisterCustomEndpointBeispiel anfordern

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

Antwort:

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

> **Note**
>
> Speichern Sie die Endpunktadresse. Sie werden es für die future Gerätekommunikation benötigen.

Verwenden Sie die `GetCustomEndpoint` API, um die Endpunktinformationen zurückzugeben.

Weitere Informationen zur RegisterCustomEndpointAPI und zur API finden Sie im GetCustomEndpointAPI-Referenzhandbuch für verwaltete Integrationen.

Schritt 2: Erstellen Sie ein Bereitstellungsprofil

Ein Bereitstellungsprofil enthält die Sicherheitsanmeldedaten und Konfigurationseinstellungen, die Ihre Geräte benötigen, um eine Verbindung zu verwalteten Integrationen herzustellen.

Um ein Flottenbereitstellungsprofil zu erstellen

- Rufen Sie die CreateProvisioningProfileAPI auf, um Folgendes zu generieren:
  - Eine Bereitstellungsvorlage, die die Verbindungseinstellungen für Geräte definiert
  - Ein Anspruchszertifikat und ein privater Schlüssel für die Geräteauthentifizierung

> **Important**
>
> Bewahren Sie das Anspruchszertifikat, den privaten Schlüssel und die Vorlagen-ID sicher auf. Sie benötigen diese Anmeldeinformationen, um Geräte in verwaltete

Integrationen einzubinden. Wenn Sie diese Anmeldeinformationen verlieren, müssen Sie ein neues Provisioning-Profil erstellen.

`CreateProvisioningProfile`Beispiel für eine Anfrage

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

Antwort:

```
{
"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"
}
```

Schritt 3: Erstellen Sie eine verwaltete Sache (Flottenbereitstellung)

Verwenden Sie die `CreateManagedThing` API, um ein verwaltetes Ding für Ihr Hub-Gerät zu erstellen. Jeder Hub benötigt sein eigenes verwaltetes Ding mit einzigartigen Authentifizierungsmaterialien. Weitere Informationen zur API finden Sie in der CreateManagedThingAPI-Referenz für verwaltete Integrationen.

Wenn Sie ein verwaltetes Ding erstellen, geben Sie die folgenden Parameter an:

- `Role`: Setzen Sie diesen Wert `CONTROLLER` für Hubs, die Command and Control nicht unterstützen, auf, andernfalls auf`DEVICE`.
- `AuthenticationMaterialType`: Setzen Sie diesen Wert auf`WIFI_SETUP_QR_BAR_CODE`.

- `AuthenticationMaterial`: Schließt die folgenden Felder ein. Sie können eines `UPC` oder `EAN` aber nicht beide verwenden.
  - `SN`: Die eindeutige Seriennummer für dieses Gerät
  - `UPC`: Der universelle Produktcode für dieses Gerät
  - `EAN`: Die internationale Artikelnummer für dieses Gerät

> **Important**
>
> Jedes Gerät muss eine eindeutige Seriennummer (SN) in seinem Authentifizierungsmaterial haben.

`CreateManagedThing`Beispiel für eine Anfrage:

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

Weitere Informationen finden Sie CreateManagedThingin der API-Referenz für verwaltete Integrationen.

(Optional) Holen Sie sich ein verwaltetes Ding

`PRE_ASSOCIATED`Bevor Sie fortfahren können, müssen Sie zuerst das `ProvisioningStatus` von Ihnen verwaltete Ding haben. Weitere Informationen finden Sie ProvisioningStatus unter Gerätebereitstellung. Verwenden Sie die `GetManagedThing` API, um zu überprüfen, ob Ihr verwaltetes Ding vorhanden ist und für die Bereitstellung bereit ist. Weitere Informationen finden Sie GetManagedThingin der API-Referenz für verwaltete Integrationen.

Schritt 4: Erstellen Sie die Verzeichnisstruktur

Erstellen Sie Verzeichnisse für Ihre Konfigurationsdateien und Zertifikate. Standardmäßig verwendet der Hub-Onboarding-Prozess die`/data/aws/iotmi/config/iotmi_config.json`.

Sie können in der Konfigurationsdatei benutzerdefinierte Pfade für Zertifikate und private Schlüssel angeben. In diesem Handbuch wird der Standardpfad verwendet`/data/aws/iotmi/certs`.

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

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

Schritt 5: Fügen Sie dem Hub-Gerät Authentifizierungsmaterialien hinzu

Kopieren Sie Zertifikate und Schlüssel auf Ihr Hub-Gerät und erstellen Sie dann eine gerätespezifische Konfigurationsdatei. Diese Dateien stellen während des Bereitstellungsprozesses eine sichere Kommunikation zwischen Ihrem Hub und den verwalteten Integrationen her.

Um das Anforderungszertifikat und den Schlüssel zu kopieren

- Kopieren Sie diese Authentifizierungsdateien aus Ihrer `CreateProvisioningProfile` API-Antwort auf Ihr Hub-Gerät:
  - `claim_cert.pem`: Das Antragszertifikat (gilt für alle Geräte)
  - `claim_pk.key`: Der private Schlüssel für das Anspruchszertifikat

  Platzieren Sie beide Dateien im `/data/aws/iotmi/certs` Verzeichnis.

  > **Important**
  >
  > Achten Sie beim Speichern von Zertifikaten und privaten Schlüsseln im PEM-Format auf eine korrekte Formatierung, indem Sie Zeilenumbruchzeichen korrekt behandeln. Bei PEM-codierten Dateien (`\n`) müssen die Zeilenumbruchzeichen durch tatsächliche Zeilentrennzeichen ersetzt werden, da das bloße Speichern von Zeilenumbrüchen mit Escape-Zeichen später nicht korrekt abgerufen werden kann.

> **Note**
>
> Wenn Sie sicheren Speicher verwenden, speichern Sie diese Anmeldeinformationen an Ihrem sicheren Speicherort statt im Dateisystem. Weitere Informationen finden Sie unter Erstellen Sie einen benutzerdefinierten Zertifikatshandler für sicheren Speicher.

Schritt 6: Erstellen Sie die Gerätekonfigurationsdatei

Erstellen Sie eine Konfigurationsdatei, die eindeutige Gerätekennungen, Zertifikatsspeicherorte und Bereitstellungseinstellungen enthält. Das SDK verwendet diese Datei beim Onboarding des Hubs, um Ihr Gerät zu authentifizieren, den Bereitstellungsstatus zu verwalten und Verbindungseinstellungen zu speichern.

> **Note**
>
> Jedes Hub-Gerät benötigt eine eigene Konfigurationsdatei mit eindeutigen gerätespezifischen Werten.

Gehen Sie wie folgt vor, um Ihre Konfigurationsdatei zu erstellen oder zu ändern und sie auf den Hub zu kopieren.

- Erstellen oder ändern Sie die Konfigurationsdatei (Fleet Provisioning).

  Konfigurieren Sie diese erforderlichen Felder in der Gerätekonfigurationsdatei:

  - Zertifikatspfade
    1. `iot_claim_cert_path`: Speicherort Ihres Antragszertifikats (`claim_cert.pem`)
    2. `iot_claim_pk_path`: Ort Ihres privaten Schlüssels (`claim_pk.key`)
    3. Wird `SECURE_STORAGE` für beide Felder verwendet, wenn Sie den Secure Storage Cert Handler implementieren
  - Verbindungseinstellungen
    1. `fp_template_name`: Der `ProvisioningProfile` Name von früher.
    2. `endpoint_url`: Die URL Ihres Endpunkts für verwaltete Integrationen aus der `RegisterCustomEndpoint` API-Antwort (gilt für alle Geräte in einer Region).

- Gerätekennungen
  1. SN: Geräteseriennummer, die Ihrem `CreateManagedThing` API-Aufruf entspricht (einmalig pro Gerät)
  2. UPCUniverseller Produktcode aus Ihrem `CreateManagedThing` API-Aufruf (gilt für alle Geräte dieses Produkts)

```
{
    "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"
    }
}
```

Inhalt der Konfigurationsdatei

Überprüfen Sie den Inhalt der `iotmi_config.json` Datei.

Inhalt

Key (Schlüssel)	Werte	Vom Kunden hinzugefügt?	Hinweise
`iot_provisioning_method`	`FLEET_PROVISIONING`	Ja	Geben Sie die Bereitstellungsmethode an, die Sie verwenden möchten.

Key (Schlüssel)	Werte	Vom Kunden hinzugefügt?	Hinweise
`iot_claim_cert_path`	Der Dateipfad, den Sie angeben oder`SECURE_STORAGE` . Beispiel: `/data/aws/iotmi/certs/claim_cert.pem`	Ja	Geben Sie den Dateipfad an, den Sie verwenden möchten oder`SECURE_STORAGE` .
`iot_claim_pk_path`	Der Dateipfad, den Sie angeben oder`SECURE_STORAGE` . Beispiel: `/data/aws/iotmi/certs/claim_pk.pem`	Ja	Geben Sie den Dateipfad an, den Sie verwenden möchten oder`SECURE_STORAGE` .
`fp_template_name`	Der Name der Vorlage für die Flottenbereitstellung sollte dem Namen der Vorlage entsprechen`ProvisioningProfile` , die zuvor verwendet wurde.	Ja	Entspricht dem Namen der Datei`ProvisioningProfile` , die zuvor verwendet wurde
`endpoint_url`	Die Endpunkt-URL für verwaltete Integrationen.	Ja	Ihre Geräte verwenden diese URL, um eine Verbindung zur Cloud für verwaltete Integrationen herzustellen. Verwenden Sie die RegisterCustomEndpointAPI, um diese Informationen zu erhalten.

Key (Schlüssel)	Werte	Vom Kunden hinzugefügt?	Hinweise
`SN`	Die Seriennummer des Geräts. Beispiel, `AIDACKCEVSQ6C2EXAMPLE` .	Ja	Sie müssen diese eindeutigen Informationen für jedes Gerät angeben.
`UPC`	Universeller Produktcode des Geräts. Beispiel, `841667145075` .	Ja	Sie müssen diese Informationen für das Gerät angeben.
`managed_thing_id`	Die ID des verwalteten Objekts.	Nein	Diese Informationen werden später im Rahmen des Onboarding-Prozesses nach der Hub-Bereitstellung hinzugefügt.
`iot_provisioning_state`	Der Bereitstellungsstatus.	Ja	Der Bereitstellungsstatus muss auf festgelegt werden. `NOT_PROVISIONED`
`iot_permanent_cert_path`	Der IoT-Zertifikatspfad. Beispiel, `/data/aws/iotmi/iot_cert.pem` .	Nein	Diese Informationen werden später im Rahmen des Onboarding-Prozesses nach der Hub-Bereitstellung hinzugefügt.
`iot_permanent_pk_path`	Der Dateipfad für den privaten IoT-Schlüssel. Beispiel, `/data/aws/iotmi/iot_pk.pem` .	Nein	Diese Informationen werden später im Rahmen des Onboarding-Prozesses nach der Hub-Bereitstellung hinzugefügt.

Key (Schlüssel)	Werte	Vom Kunden hinzugefügt?	Hinweise
`client_id`	Die Client-ID, die für MQTT-Verbindungen verwendet wird.	Nein	Diese Informationen werden später im Rahmen des Onboarding-Prozesses nach der Hub-Bereitstellung hinzugefügt, sodass sie von anderen Komponenten genutzt werden können.
`mqtt_keep_alive_interval`	Der Bereich liegt zwischen 30 und 1200, und die Einheiten sind in Sekunden angegeben. Der Standardwert ist 300.	Ja	Verwenden Sie dies, um ein Keep-Alive-Intervall für MQTT-Verbindungen festzulegen.
`event_manager_upper_bound`	Der Standardwert lautet 500.	Nein	Diese Informationen werden später im Rahmen des Onboarding-Prozesses nach der Hub-Bereitstellung hinzugefügt, sodass sie von anderen Komponenten genutzt werden können.

Schritt 7: Kopieren Sie die Konfigurationsdatei auf Ihren Hub

Kopieren Sie Ihre Konfigurationsdatei in `/data/aws/iotmi/config` oder Ihren benutzerdefinierten Verzeichnispfad. Sie geben diesen Pfad zur `Hub0nboarding` Binärdatei während des Onboarding-Prozesses an.

Für die Bereitstellung von Flotten

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

```
            config/
                iotmi_config.json
            certs/
                claim_cert.pem
                claim_pk.key
```

Geräte einbinden und im Hub bedienen

Richten Sie Ihre Geräte so ein, dass sie in Ihren Hub für verwaltete Integrationen integriert werden, indem Sie ein verwaltetes Objekt erstellen und es mit Ihrem Hub verbinden. Geräte können entweder durch eine einfache Einrichtung oder durch eine benutzergeführte Einrichtung in einen Hub integriert werden.

Themen

- Einfache Einrichtung für das Onboarding und den Betrieb von Geräten
- Benutzergeführte Einrichtung zur Einbindung und Bedienung von Geräten

Einfache Einrichtung für das Onboarding und den Betrieb von Geräten

Richten Sie Ihre Geräte so ein, dass sie in Ihren Hub für verwaltete Integrationen integriert werden, indem Sie ein verwaltetes Objekt erstellen und es mit Ihrem Hub verbinden. In diesem Abschnitt werden die Schritte beschrieben, um den Onboarding-Prozess für Geräte mithilfe einer einfachen Einrichtung abzuschließen.

Voraussetzungen

Gehen Sie wie folgt vor, bevor Sie versuchen, ein Gerät an Bord zu nehmen:

- Integrieren Sie ein Hub-Gerät in den Managed Integrations Hub.
- Installieren Sie die neueste Version von AWS CLI aus der Managed Integrations AWS CLI Command Reference
- Abonnieren Sie DEVICE_LIFE_CYCLE-Ereignisbenachrichtigungen.

Einrichtungsschritte

- Schritt 1: Erstellen Sie einen Anmeldeinformationsspeicher
- Schritt 2: Fügen Sie Ihrem Hub den Credential Locker hinzu

Schritt 1: Erstellen Sie einen Anmeldeinformationsspeicher

Erstellen Sie einen Anmeldeinformationsspeicher für Ihr Gerät.

Um einen Anmeldeinformationsspeicher zu erstellen

- Verwenden Sie den Befehl `create-credential-locker`. Die Ausführung dieses Befehls löst die Erstellung aller Fertigungsressourcen aus, einschließlich des Wi-Fi-Setup-Schlüsselpaars und des Gerätezertifikats.

  create-credential-locker-Beispiel

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

  Antwort:

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

  Weitere Informationen finden Sie in der create-credential-lockerBefehlsreferenz für verwaltete Integrationen zu diesem AWS CLI Befehl.

Schritt 2: Fügen Sie Ihrem Hub den Credential Locker hinzu

Fügen Sie das Credential Locker zu Ihrem Hub hinzu.

Um Ihrem Hub ein Schließfach für Anmeldeinformationen hinzuzufügen

- Verwenden Sie den folgenden Befehl, um Ihrem Hub einen Anmeldeinformationsspeicher hinzuzufügen.

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

Schritt 3: Erstellen Sie ein verwaltetes Ding mit Anmeldeinformationen.

Erstellen Sie ein verwaltetes Ding mit Anmeldeinformationen für Ihr Gerät. Jedes Gerät benötigt sein eigenes verwaltetes Ding.

Um ein verwaltetes Ding zu erstellen

- Verwenden Sie den `create-managed-thing` Befehl, um ein verwaltetes Ding für Ihr Gerät zu erstellen.

  create-managed-thing-Beispiel

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

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

> **Note**
>
> Es gibt separate Befehle für Z-Wave- und ZigBee-Geräte.

  Antwort:

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

Weitere Informationen finden Sie in der create-managed-thingBefehlsreferenz für verwaltete Integrationen AWS CLI zu diesem Befehl.

Schritt 4: Schließen Sie das Gerät an und überprüfen Sie seinen Status.

Schließen Sie das Gerät an und überprüfen Sie seinen Status.

- Verwenden Sie den `get-managed-thing` Befehl, um den Status Ihres Geräts zu überprüfen. Das ProvisioningStatus Ihres verwalteten Objekts muss AKTIVIERT sein. Weitere Informationen finden Sie ProvisioningStatus unter Gerätebereitstellung.

  get-managed-thing-Beispiel

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

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

Antwort:

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

Weitere Informationen finden Sie unter dem get-managed-thingBefehl in der Befehlsreferenz für verwaltete Integrationen AWS CLI .

Schritt 5: Holen Sie sich die Gerätefunktionen

Verwenden Sie den `get-managed-thing-capabilities` Befehl, um Ihre Endpunkt-ID abzurufen und eine Liste möglicher Aktionen für Ihr Gerät anzuzeigen.

Um die Funktionen eines Geräts abzurufen

- Verwenden Sie den `get-managed-thing-capabilities` Befehl und notieren Sie sich die Endpunkt-ID.

  get-managed-thing-capabilties-Beispiel

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

  Antwort:

  ```
  {
      "ManagedThingId": "1e84f61fa79a41219534b6fd57052cbc",
      "CapabilityReport": {
          "version": "1.0.0",
          "nodeId": "zw.FCB10009+06",
          "endpoints": [
              {
                  "id": "ENDPOINT_ID"
  ```

```
                "deviceTypes": [
                    "On/Off Switch"
                ],
                "capabilities": [
                    {
                        "id": "matter.OnOff@1.4",
                        "name": "On/Off",
                        "version": "6",
                        "properties": [
                            "OnOff"
                        ],
                        "actions": [
                            "Off",
                            "On"
                        ],
                        "events": []
                    }
                    ...
}
```

Weitere Informationen finden Sie unter dem get-managed-thing-capabilitiesBefehl in der AWS CLI Befehlsreferenz für verwaltete Integrationen.

Schritt 6: Senden Sie einen Befehl an das verwaltete Ding

Verwenden Sie den `send-managed-thing-command` Befehl, um einen Befehl zum Umschalten einer Aktion an Ihr verwaltetes Ding zu senden.

Um einen Befehl an Ihr verwaltetes Ding zu senden

- Verwenden Sie den `send-managed-thing-command` Befehl, um einen Befehl an Ihr verwaltetes Ding zu senden.

  send-managed-thing-command-Beispiel

```
json=$(jq -cr '.|@json') <<EOF
[
  {
    "endpointId": "1",
    "capabilities": [
      {
        "id": "matter.OnOff@1.4",
```

```
        "name": "On/Off",
        "version": "1",
        "actions": [
          {
            "name": "Toggle",
            "parameters": {}
          }
        ]
      }
    ]
  }
]
EOF
aws iot-managed-integrations send-managed-thing-command \
--managed-thing-id "DEVICE_MANAGED_THING_ID" --endpoints "ENDPOINT_ID"
```

> **Note**
>
> In diesem Beispiel wird jq cli verwendet, aber Sie können auch die gesamte `endpointId` Zeichenfolge übergeben

Antwort:

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

Weitere Informationen finden Sie unter dem send-managed-thing-commandBefehl in der Befehlsreferenz für verwaltete Integrationen AWS CLI .

Schritt 7: Entfernen Sie das verwaltete Ding von Ihrem Hub

Bereinigen Sie Ihren Hub, indem Sie das verwaltete Ding entfernen.

Um ein verwaltetes Ding zu löschen

- Verwenden Sie den `delete-managed-thing` Befehl, um ein verwaltetes Ding von Ihrem Gerätehub zu entfernen.

delete-managed-thing-Beispiel

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

Weitere Informationen finden Sie in der delete-managed-thingBefehlsreferenz für verwaltete Integrationen zu diesem AWS CLI Befehl.

> **Note**
>
> Wenn das Gerät in einem bestimmten `DELETE_IN_PROGRESS` Zustand feststeckt, hängen Sie das `--force` Kennzeichen an. `delete-managed-thing command`

> **Note**
>
> Bei Z-Wave-Geräten müssen Sie das Gerät nach der Ausführung des Befehls in den Pairing-Modus versetzen.

Benutzergeführte Einrichtung zur Einbindung und Bedienung von Geräten

Richten Sie Ihre Geräte so ein, dass sie in Ihren Hub für verwaltete Integrationen integriert werden, indem Sie ein verwaltetes Objekt erstellen und es mit Ihrem Hub verbinden. In diesem Abschnitt werden die Schritte beschrieben, um den Onboarding-Prozess für Geräte mithilfe der benutzergeführten Einrichtung abzuschließen.

Voraussetzungen

Gehen Sie wie folgt vor, bevor Sie versuchen, ein Gerät an Bord zu nehmen:

- Integrieren Sie ein Hub-Gerät in den Managed Integrations Hub.
- Installieren Sie die neueste Version von AWS CLI aus der Managed Integrations AWS CLI Command Reference
- Abonnieren Sie DEVICE_DISCOVERY-STATUS-Ereignisbenachrichtigungen.

Benutzergeführte Einrichtungsschritte

Voraussetzung: Aktivieren Sie den Pairing-Modus auf Ihrem Z Wave-Gerät

Aktivieren Sie den Pairing-Modus auf dem Z-Wave-Gerät. Der Pairing-Modus kann für jedes Z-Wave-Gerät unterschiedlich sein. Lesen Sie daher die Anweisungen des Geräts, um den Pairing-Modus richtig einzurichten. In der Regel handelt es sich um eine Taste, die der Benutzer drücken muss.

Schritt 1: Starten Sie die Geräteerkennung

Starten Sie die Geräteerkennung für Ihren Hub, um eine Discovery-Job-ID zu erhalten, die zum Onboarding Ihres Geräts verwendet wird.

Um die Geräteerkennung zu starten

- Verwenden Sie den start-device-discoveryBefehl, um die Discovery-Job-ID abzurufen.

  start-device-discovery-Beispiel

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

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

  #For Cloud
```

```
aws iot-managed-integrations start-device-discovery --discovery-type CLOUD \
--account-association-id C2C_ASSOCIATION_ID \

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

Antwort:

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

> **Note**
>
> Es gibt separate Befehle für Z-Wave- und Zigbee-Geräte.

Weitere Informationen zur start-device-discoveryAPI finden Sie in der Befehlsreferenz für verwaltete Integrationen. AWS CLI

Schritt 2: Fragen Sie die Discovery-Job-ID ab

Verwenden Sie den `list-discovered-devices` Befehl, um das Authentifizierungsmaterial Ihres Geräts abzurufen.

Um Ihre Discovery-Job-ID abzufragen

- Verwenden Sie die Discovery-Job-ID zusammen mit dem list-discovered-devices Befehl, um das Authentifizierungsmaterial Ihres Geräts abzurufen.

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

Antwort:

```
"Items": [
```

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

Schritt 3: Erstellen Sie ein verwaltetes Objekt für Ihr Gerät

Verwenden Sie den `create-managed-thing` Befehl, um ein verwaltetes Objekt für Ihr Gerät zu erstellen. Jedes Gerät benötigt sein eigenes verwaltetes Ding.

Um ein verwaltetes Ding zu erstellen

- Verwenden Sie den `create-managed-thing` Befehl, um ein verwaltetes Ding für Ihr Gerät zu erstellen.

  create-managed-thing-Beispiel

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

  Antwort:

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

  Weitere Informationen finden Sie in der create-managed-thingBefehlsreferenz für verwaltete Integrationen zu diesem AWS CLI Befehl.

Schritt 4: Fragen Sie das verwaltete Ding ab

Mit dem `get-managed-thing` Befehl können Sie überprüfen, ob ein verwaltetes Ding aktiviert ist.

Um eine verwaltete Sache abzufragen

- Verwenden Sie den `get-managed-thing` Befehl, um zu überprüfen, ob der Bereitstellungsstatus des verwalteten Objekts auf `ACTIVATED` gesetzt ist. Weitere Informationen zum Bereitstellungsstatus finden Sie unter Gerätebereitstellung.

  get-managed-thing-Beispiel

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

  Antwort:

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

  Weitere Informationen finden Sie unter dem get-managed-thingBefehl in der Befehlsreferenz für verwaltete Integrationen. AWS CLI

Schritt 5: Holen Sie sich die Funktionen für verwaltete Dinge

Sie können eine Liste der verfügbaren Aktionen für ein verwaltetes Ding anzeigen, indem Sie den verwendenget-managed-thing-capabiltiies.

Um die Funktionen eines Geräts abzurufen

- Verwenden Sie den `get-managed-thing-capabilities` Befehl, um die Endpunkt-ID abzurufen. Beachten Sie auch die Liste der möglichen Aktionen.

  get-managed-thing-capabilities-Beispiel

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

Antwort:

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

Weitere Informationen finden Sie unter dem get-managed-thing-capabilitiesBefehl in der AWS CLIBefehlsreferenz für verwaltete Integrationen.

Schritt 6: Senden Sie einen Befehl an das verwaltete Ding

Sie können den send-managed-thing-command Befehl verwenden, um einen Befehl zum Umschalten einer Aktion an Ihr verwaltetes Ding zu senden.

Senden Sie mithilfe einer Umschaltaktion einen Befehl an das verwaltete Ding.

- Verwenden Sie den `send-managed-thing-command` Befehl, um einen Befehl zum Umschalten zu senden.

  send-managed-thing-command-Beispiel

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

> **Note**
>
> In diesem Beispiel wird jq cli verwendet, aber Sie können auch die gesamte Zeichenfolge übergeben `endpointId`

Antwort:

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

Weitere Informationen finden Sie unter dem send-managed-thing-commandBefehl in der Befehlsreferenz für verwaltete Integrationen AWS CLI .

Schritt 7: Überprüfen Sie den Status des verwalteten Dings

Überprüfen Sie den Status des verwalteten Dings, um zu überprüfen, ob die Umschaltaktion erfolgreich war.

Um den Gerätestatus eines verwalteten Objekts zu überprüfen

- Verwenden Sie den `get-managed-thing-state` Befehl, um zu überprüfen, ob die Umschaltaktion erfolgreich war.

  get-managed-thing-state-Beispiel

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

  Antwort:

```
{
      "Endpoints": [
          {
              "endpointId": "1",
              "capabilities": [
                  {
                      "id": "matter.OnOff@1.4",
                      "name": "On/Off",
                      "version": "1.4",
                      "properties": [
                          {
                              "name": "OnOff",
```

```
                            "value": {
                                "propertyValue": true,
                                "lastChangedAt": "2025-06-03T21:50:39.886Z"
                            }
                        }
                    ]
                }
            ]
        }
    ]
}
```

Weitere Informationen zu diesem Befehl finden Sie in der get-managed-thing-stateBefehlsreferenz für verwaltete Integrationen AWS CLI.

Schritt 8: Entferne das verwaltete Ding von deinem Hub

Bereinigen Sie Ihren Hub, indem Sie das verwaltete Ding entfernen.

Um ein verwaltetes Ding zu löschen

- Verwenden Sie den delete-managed-thingBefehl, um ein verwaltetes Ding zu entfernen.

  delete-managed-thing-Beispiel

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

  Weitere Informationen finden Sie unter dem delete-managed-thingBefehl in der AWS CLI Befehlsreferenz für verwaltete Integrationen.

  > **Note**
  >
  > Wenn das Gerät in einem bestimmten `DELETE_IN_PROGRESS` Zustand feststeckt, hängen Sie das `--force` Kennzeichen an den `delete-managed-thing` Befehl an.

> **Note**
>
> Bei Z-Wave-Geräten müssen Sie das Gerät nach der Ausführung des Befehls in den Pairing-Modus versetzen.

Erstellen Sie einen benutzerdefinierten Zertifikatshandler für sicheren Speicher

Die Verwaltung von Gerätezertifikaten ist beim Onboarding des Managed Integrations Hubs von entscheidender Bedeutung. Zertifikate werden zwar standardmäßig im Dateisystem gespeichert, Sie können jedoch einen benutzerdefinierten Zertifikatshandler erstellen, um die Sicherheit zu erhöhen und die Anmeldeinformationen flexibel zu verwalten.

Das Endgeräte-SDK für verwaltete Integrationen stellt einen Zertifikatshandler zur sicheren Speicherschnittstelle bereit, den Sie als Bibliothek für gemeinsam genutzte Objekte (.so) implementieren können. Erstellen Sie Ihre sichere Speicherimplementierung zum Lesen und Schreiben von Zertifikaten und verknüpfen Sie dann die Bibliotheksdatei zur Laufzeit mit dem HubOnboarding Prozess.

API-Definition und Komponenten

Sehen Sie sich die folgende `secure_storage_cert_handler_interface.hpp` Datei an, um die API-Komponenten und Anforderungen für Ihre Implementierung zu verstehen

Themen

API-Definition

Inhalt von **`secure_storage_cert_hander_interface.hpp`**

```
/*
    * 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
```

Zentrale Komponenten

- CERT_TYPE_T — verschiedene Arten von Zertifikaten auf dem Hub.
  - CLAIM — Das ursprünglich auf dem Hub befindliche Antragszertifikat wird gegen ein permanentes Zertifikat ausgetauscht.
  - DHA — vorerst unbenutzt.
  - PERMANENT — permanentes Zertifikat für die Verbindung mit dem Endpunkt für verwaltete Integrationen.
- read_cert_and_private_key — (ZU IMPLEMENTIERENDE FUNKTION) Liest Zertifikat und Schlüsselwert in die Referenzeingabe ein. Diese Funktion muss in der Lage sein, sowohl das CLAIM- als auch das PERMANENT-Zertifikat zu lesen, und unterscheidet sich durch den oben genannten Zertifikatstyp.
- write_permanent_cert_and_private_key — (ZU IMPLEMENTIERENDE FUNKTION) schreibt das permanente Zertifikat und den Schlüsselwert an den gewünschten Ort.

Beispiel für einen Build

Trennen Sie Ihre internen Implementierungsheader von der öffentlichen Schnittstelle (`secure_storage_cert_handler_interface.hpp`), um eine saubere Projektstruktur aufrechtzuerhalten. Durch diese Trennung können Sie öffentliche und private Komponenten verwalten und gleichzeitig Ihren Zertifikatshandler erstellen.

> **Note**
>
> `secure_storage_cert_handler_interface.hpp`Als öffentlich deklarieren.

Themen

- Struktur des Projekts
- Erben Sie die Schnittstelle
- Implementierung
- CMakeList.txt

Struktur des Projekts

Erben Sie die Schnittstelle

Erstellen Sie eine konkrete Klasse, die die Schnittstelle erbt. Verstecken Sie diese Header-Datei und andere Dateien in einem separaten Verzeichnis, sodass private und öffentliche Header beim Erstellen leicht unterschieden werden können.

```
#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
```

Implementierung

Implementieren Sie die oben definierte Speicherklasse,`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;
 }
```

Implementieren Sie die in der Schnittstelle definierte Factory-Funktion,`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
    )
```

Verwendung

Nach der Kompilierung verfügen Sie über eine `libSecureStorageCertHandler.so` gemeinsam genutzte Objektbibliotheksdatei und die zugehörigen symbolischen Links. Kopieren Sie sowohl die Bibliotheksdatei als auch die symbolischen Links an den Bibliotheksort, den die HubOnboarding Binärdatei erwartet.

Themen

- Die wichtigsten Überlegungen
- Verwenden Sie sicheren Speicher

Die wichtigsten Überlegungen

- Stellen Sie sicher, dass Ihr Benutzerkonto Lese- und Schreibberechtigungen sowohl für die HubOnboarding Binärdatei als auch für die `libSecureStorageCertHandler.so` Bibliothek besitzt.
- Behalten Sie `secure_storage_cert_handler_interface.hpp` es als Ihre einzige öffentliche Header-Datei bei. Alle anderen Header-Dateien sollten in Ihrer privaten Implementierung verbleiben.

- Überprüfen Sie den Namen Ihrer Shared Object Library. Während des Builds `libSecureStorageCertHandler.so` ist HubOnboarding möglicherweise eine bestimmte Version im Dateinamen erforderlich, z. `libSecureStorageCertHandler.so.1.0` B. Verwenden Sie den `ldd` Befehl, um die Bibliotheksabhängigkeiten zu überprüfen und bei Bedarf symbolische Links zu erstellen.
- Wenn Ihre Implementierung der gemeinsam genutzten Bibliothek externe Abhängigkeiten aufweist, speichern Sie diese in einem Verzeichnis, auf das zugegriffen HubOnboarding werden kann, z. B. `/usr/lib or the iotmi_common` Verzeichnis.

Verwenden Sie sicheren Speicher

Aktualisieren Sie Ihre `iotmi_config.json` Datei, indem Sie `iot_claim_cert_path` sowohl als auch `iot_claim_pk_path` auf einstellen**`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"
  }
}
```

Benutzerdefiniertes Protokoll-Plugin

Sie können ein benutzerdefiniertes Protokoll-Plugin verwenden, um Ihre proprietären IoT-Protokolle in die verwalteten Integrationen für das AWS IoT Device Management Ökosystem zu integrieren. Über klar definierte SDK-Schnittstellen können Sie Geräte einbinden, Funktionen definieren und Kontrollabläufe in Echtzeit abwickeln, während Sie gleichzeitig die volle Kompatibilität mit verwalteten Integrationen und Hub-SDK-Komponenten wahren.

In der folgenden Liste werden die wichtigsten Funktionen des Plug-ins für benutzerdefinierte Protokolle beschrieben.

Anpassung des Datenmodells

Definieren Sie Ihre eigenen AWS Datenmodellschemas und laden Sie sie während des Bereitstellungsprozesses in verwaltete Integrationen hoch. Sie können diese Schemas später in Ihren Workflows verwenden.

Flexible Plugin-Implementierung

- Erstellen Sie Ihre eigene Plugin-Komponente mitHub-SDK-Client.
- Implementieren Sie separate Plugins für verschiedene Funktionen, z. B. Bereitstellung und Steuerung, oder erstellen Sie einen einheitlichen Client für beide.
- Sorgen Sie für eine klare Grenze zwischen verwalteten Integrationsressourcen und Ihren eigenen Ressourcen wie Middleware-Stacks, um eine entkoppelte und entwicklungsfreundliche Implementierung der Codelogik zu erreichen.

Abwärtskompatibilität

Für Bestandskunden können Sie Ihr neues benutzerdefiniertes Protokoll problemlos integrieren und gleichzeitig die vorhandenen Funktypen unverändert weiterverwenden.

Das folgende Diagramm veranschaulicht die Architektur des Plug-ins für benutzerdefinierte Protokolle.

Hub-SDK-Client

Die Hub SDK-Clientbibliothek dient als Schnittstelle zwischen dem Hub-SDK für verwaltete Integrationen und Ihrem eigenen Protokollstapel, der auf demselben Hub ausgeführt wird. Sie macht eine Reihe von öffentlichen Dateien verfügbar APIs , um die Interaktion Ihres Protokollstapels mit den Device Hub SDK-Komponenten zu erleichtern. Zu den Anwendungsfällen gehören die benutzerdefinierte Plugin-Kontrolle, der benutzerdefinierte Plugin-Provisioner und der lokale Controller.

Themen

- Datentypen

Holen Sie sich Ihr Hub-SDK für verwaltete Integrationen

Der Hub SDK Client wird mit dem SDK für verwaltete Integrationen geliefert. Kontaktieren Sie uns über die Managed Integrations Console, um auf das Hub-SDK zuzugreifen.

Über das Hub SDK-Toolkit

Nach dem Herunterladen wird ein `IotMI-DeviceSDK-Toolkit` Ordner angezeigt, der alle öffentlichen Header-Dateien und `.so` Dateien enthält, die Sie in Ihrer Anwendung verwenden können. Das Team für verwaltete Integrationen stellt auch ein Beispiel `main.cpp` für Demo-Zwecke zur Verfügung, zusammen mit der Binärdatei der Demo-Anwendung`bin/`, die Sie direkt ausführen können. Sie können dies optional als Ausgangspunkt für Ihre Anwendung verwenden.

Erstellen Sie Ihre benutzerdefinierte Anwendung mit dem Hub SDK-Client

Gehen Sie wie folgt vor, um Ihre benutzerdefinierte Anwendung zu erstellen.

1. Nehmen Sie die Header-Dateien (.h) und die Dateien mit gemeinsam genutzten Objekten (.so) in Ihre Anwendung auf.

   Sie müssen die öffentlichen Header-Dateien (.h) und Shared Object-Dateien (.so) in Ihre Anwendung aufnehmen. Die .so-Dateien können Sie in einem lib-Ordner ablegen. Das endgültige Layout wird dem folgenden ähneln:

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

2. Erstellen Sie einen Hub-SDK-Client in Ihrer Hauptanwendung.

   a. In Ihrer Hauptanwendung müssen Sie zuerst den Hub SDK-Client initialisieren, bevor er zur Bearbeitung von Anfragen verwendet werden kann. Sie können den Client einfach mit einem `clientId` erstellen.

b. Sobald Sie den Client haben, können Sie ihn mit dem Geräte-SDK für verwaltete Integrationen verbinden.

   Im Folgenden finden Sie ein Beispiel für die Erstellung des Hub SDK-Clients und für die Herstellung einer Verbindung.

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

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

> **Note**
>
> *your_own_clientId*muss mit dem identisch sein, den Sie start-device-discoveryin der benutzergeführten Einrichtung oder create-managed-thingim Bereitstellungsablauf für die einfache Einrichtung angegeben haben.

3. Veröffentlichen und abonnieren Sie, indem Sie die folgenden Schritte ausführen.

   a. Nachdem die Verbindung hergestellt wurde, können Sie jetzt eingehende Aufgaben über das Hub-SDK für verwaltete Integrationen abonnieren. Bei den eingehenden Aufgaben kann es sich um Steuerungs- oder Bereitstellungsaufgaben handeln. Sie müssen auch Ihre eigene Rückruffunktion für eingegangene Aufgaben und Ihren eigenen benutzerdefinierten Kontext für Ihre eigenen Nachverfolgungszwecke definieren.

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

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

   b. Nachdem die Verbindung hergestellt wurde, können Sie nun Anfragen aus Ihrer Anwendung im Hub-SDK für verwaltete Integrationen veröffentlichen. Sie können Ihren eigenen Aufgabennachrichtentyp mit unterschiedlichen Payloads für unterschiedliche Geschäftszwecke definieren. Die Anfragen können sowohl Kontroll- als auch

Bereitstellungsanforderungen enthalten, ähnlich wie beim Abonnement-Flow. Schließlich können Sie Ihnen eine Adresse zuweisen, `rspPayload` um die Antwort vom Hub-SDK für verwaltete Integrationen synchronisiert zu erhalten.

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

  status = client->iotmi_control_publish_request(&api_payload);

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

4. Erstellen Sie Ihre eigene `CMakeLists.txt` und erstellen Sie Ihre Anwendung von dort aus. Die endgültige Ausgabe könnte eine ausführbare Binärdatei sein, wie `MyFirstApplication`

Ausführen Ihrer benutzerdefinierten Anwendung

Bevor Sie Ihre benutzerdefinierte Anwendung ausführen, führen Sie die folgenden Schritte aus, um Ihren Hub einzurichten und das Hub-SDK für verwaltete Integrationen zu starten:

- Folgen Sie den Onboarding-Anweisungen unter. Integrieren Sie Ihre Hubs in verwaltete Integrationen
- Schließen Sie den unter dokumentierten Installationsvorgang ab. Installieren und validieren Sie das Hub-SDK für verwaltete Integrationen

Sobald die Voraussetzungen erfüllt sind, können Sie Ihre benutzerdefinierte Anwendung ausführen. Zum Beispiel:

```
./MyFirstApplication
```

Important

Sie müssen das unter aufgeführte Startskript manuell Stellen Sie das Hub-SDK mit einem Skript bereit mit einem Skript aktualisieren, um Ihre eigene Anwendung zu starten. Die Reihenfolge ist wichtig, ändern Sie die Reihenfolge nicht.
Aktualisiere Folgendes. Änderung

```
./IotMI-DeviceSDK-Toolkit/bin/DeviceSDKClientDemo >> $LOGS_DIR/
logDeviceSDKClientDemo_logs.txt &
```

to

```
./MyFirstApplication >> $LOGS_DIR/MyFirstApplication_logs.txtt &
```

API-Referenz zum Hub-SDK-Client

Der Hub SDK-Client (SDKClient Geräteklasse) bietet eine Schnittstelle für Ihre benutzerdefinierte Anwendung, über die Sie mit dem Geräte-SDK für verwaltete Integrationen interagieren können. Mit diesem Client können Sie Folgendes tun:

- Abonnieren Sie bereitstellungs- und steuerungsbezogene Aufgaben aus den Komponenten für verwaltete Integrationen.
- Veröffentlichen Sie bereitstellungs- und steuerungsbezogene Anfragen an die Komponenten der verwalteten Integrationen.

Informationen zu verwalteten Integrationen für finden Sie unter Was ist. AWS IoT Device Management APIs AWS Lambda

Themen

- Andere APIs

Initialisierung des Clients

Um mit der Verwendung von zu beginnen`DeviceSDKClient`, initialisieren Sie es mit einer Client-ID.

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

Dadurch wird eine neue `DeviceSDKClient` Instanz mit dem angegebenen `clientId` Wert erstellt. Die `clientId` muss mit der übereinstimmen, die Sie bei verwalteten Integrationen registrieren.

Parameter

`clientId`(string) — Die Client-ID für diese Instanz.

```
connect()
```

Verbindet die `DeviceSDKClient` Instanz mit verwalteten Integrationen.

Rückgabewerte

- `IOTMI_STATUS_OK`- Die Verbindung war erfolgreich.
- `IOTMI_STATUS_CUSTOM_PLUGIN_CONNECTION_ERROR`- Beim Herstellen einer Verbindung zu verwalteten Integrationen ist ein Fehler aufgetreten.

Abonnement für Bereitstellungsaufgaben

Verwenden Sie diese Methoden, um bereitstellungsbezogene Aufgaben aus den Komponenten der verwalteten Integrationen zu abonnieren.

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

Abonniert bereitstellungsbezogene Aufgaben, wie z. B. das Onboarding und die Deprovisionierung von Geräten, aus den Komponenten der verwalteten Integrationen.

Parameter

- `callback`(Device SDKClient _SubscriberCallback) — Eine Rückruffunktion, die ausgeführt wird, wenn eine Aufgabe empfangen wird.
- `context`(char*) — Ein benutzerdefinierter Kontext, der an die Callback-Funktion übergeben wird.

Rückgabewerte

- `IOTMI_STATUS_OK`- Das Abonnement war erfolgreich.
- `IOTMI_STATUS_CUSTOM_PLUGIN_CLIENT_NOT_CONNECTED`- Die SDKClient Geräteinstanz ist nicht mit verwalteten Integrationen verbunden.
- `IOTMI_STATUS_CUSTOM_PLUGIN_SUBSCRIBE_ERROR`- Beim Abonnieren der Aufgaben ist ein Fehler aufgetreten.

Veröffentlichung der Aufgabe bereitstellen

Verwenden Sie diese Methoden, um bereitstellungsbezogene Anfragen in den Komponenten der verwalteten Integration zu veröffentlichen.

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

Veröffentlicht eine bereitstellungsbezogene Anfrage an die Komponenten der verwalteten Integrationen. Zum Beispiel ein Ereignis, bei dem das Gerät integriert wurde, oder der Status der Deprovisionierung

Parameter

`request`(DataModel: :iotmi_client_request_t) — Ein Zeiger auf eine Anforderungsstruktur, die die Details enthält.

Rückgabewerte

- `IOTMI_STATUS_OK`— Die Anfrage wurde erfolgreich veröffentlicht.
- `IOTMI_STATUS_CUSTOM_PLUGIN_CLIENT_NOT_CONNECTED`- Die `DeviceSDKClient` Instanz ist nicht mit verwalteten Integrationen verbunden.
- `IOTMI_STATUS_INVALID_PARAMETER`- Ein oder mehrere Parameter in der Anfrage sind ungültig.

- `IOTMI_STATUS_INVALID_JSON_OBJECT`- Die Payload der Anfrage ist kein gültiges JSON-Objekt.
- `IOTMI_STATUS_NO_MEMORY`- Ein Fehler bei der Speicherzuweisung ist aufgetreten.

Steuern Sie das Abonnement für Aufgaben

Verwenden Sie diese Methoden, um steuerungsbezogene Aufgaben aus den Komponenten der verwalteten Integrationen zu abonnieren.

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

Abonniert steuerungsbezogene Aufgaben (z. B. Anfragen zur Gerätesteuerung) aus den Komponenten der verwalteten Integrationen.

Parameter

- `callback`(Device SDKClient _SubscriberCallback) — Eine Callback-Funktion, die ausgeführt wird, wenn eine Aufgabe empfangen wird.
- `context`(char) — Ein benutzerdefinierter Kontext, der an die Callback-Funktion übergeben wurde.

Rückgabewerte

- `IOTMI_STATUS_OK`- Das Abonnement war erfolgreich.
- `IOTMI_STATUS_CUSTOM_PLUGIN_CLIENT_NOT_CONNECTED`- Die `DeviceSDKClient` Instanz ist nicht mit verwalteten Integrationen verbunden.
- `IOTMI_STATUS_CUSTOM_PLUGIN_SUBSCRIBE_ERROR`- Beim Abonnieren der Aufgaben ist ein Fehler aufgetreten.

Steuern Sie die Veröffentlichung der Aufgaben

Verwenden Sie diese Methoden, um Anfragen im Zusammenhang mit der Steuerung an die Komponenten der verwalteten Integrationen zu veröffentlichen.

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

Veröffentlicht eine Anfrage, die sich auf die Steuerung bezieht, an die Komponenten der verwalteten Integrationen. Zum Beispiel unerwünschte Ereignisse, Befehlsanforderungen oder Gerätestatusabfragen.

Parameter

`request`(DataModel: :iotmi_client_request_t) — Ein Zeiger auf eine Anforderungsstruktur, die die Details enthält.

Rückgabewerte

- `IOTMI_STATUS_OK`— Die Anfrage wurde erfolgreich veröffentlicht.
- `IOTMI_STATUS_CUSTOM_PLUGIN_CLIENT_NOT_CONNECTED`- Die SDKClient Geräteinstanz ist nicht mit verwalteten Integrationen verbunden.
- `IOTMI_STATUS_INVALID_PARAMETER`- Ein oder mehrere Parameter in der Anfrage sind ungültig.
- `IOTMI_STATUS_INVALID_JSON_OBJECT`- Die Payload der Anfrage ist kein gültiges JSON-Objekt.
- `IOTMI_STATUS_NO_MEMORY`- Ein Fehler bei der Speicherzuweisung ist aufgetreten.

Funktionen zum Protokollieren

Verwenden Sie diese Methoden, um die Protokollierungsfunktionen zu implementieren, die verwaltete Integrationen bieten.

Initialisierung des Loggers

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

Sie müssen den Logger initialisieren, bevor Sie eine Protokollierungsfunktion verwenden können.

Parameter

`logger_name`- Der von Ihnen angegebene Logger-Name. Der Standardwert ist: `MyApplication`

Makros protokollieren

`LOGGER_LOGD(...)`

Verwenden Sie dieses Makro in Ihrer Anwendung für die Protokollierung auf DEBUG-Ebene.

`LOGGER_LOGI(...)`

Verwenden Sie dieses Makro in Ihrer Anwendung für die Protokollierung auf INFO-Ebene.

`LOGGER_LOGW(...)`

Verwenden Sie dieses Makro in Ihrer Anwendung für die Protokollierung auf WARN-Ebene.

`LOGGER_LOGE(...)`

Verwenden Sie dieses Makro in Ihrer Anwendung für die Protokollierung auf ERROR-Ebene.

> **Note**
>
> Weitere Informationen zu den Protokollierungsfunktionen finden Sie in der Hub-Protokollierungsdokumentation. Plug-ins für benutzerdefinierte Protokolle unterstützen vollständig alle Protokollierungsfunktionen, die verwaltete Integrationen bieten.

Andere APIs

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

Gibt die Client-ID zurück, die der SDKClient Geräteinstanz zugeordnet ist.

Rückgabewerte

Die Client-ID.

Datentypen

In diesem Abschnitt werden die Datentypen definiert, die für das benutzerdefinierte Protokoll-Plugin verwendet werden.

iotmi_client_request_t

Stellt eine Anfrage dar, die in den Komponenten der verwalteten Integration veröffentlicht werden soll.

messageType

Der Typ der Nachricht (CommonTypes: MIMessage :C2 Type). Die folgende Liste zeigt die gültigen Werte.

- `C2MI_DEVICE_ONBOARDED`: Weist auf eine Onboarding-Meldung des Geräts mit zugehöriger Nutzlast hin.
- `C2MI_DE_PROVISIONING_PRE_ASSOCIATED_COMPLETE`: Weist darauf hin, dass für ein bereits zugeordnetes Gerät eine Benachrichtigung über die Beendigung der Bereitstellung der Aufgabe angezeigt wird.
- `C2MI_DE_PROVISIONING_ACTIVATED_COMPLETE`: Zeigt an, dass für ein aktiviertes Gerät eine Benachrichtigung über den Abschluss der Bereitstellung der Aufgabe angezeigt wird.
- `C2MI_DE_PROVISIONING_COMPLETE_RESPONSE`: Zeigt eine Antwort an, dass die Aufgabe zur Deprovisionierung abgeschlossen wurde.
- `C2MI_CONTROL_EVENT`: Weist auf ein Steuerungsereignis mit potenzieller Änderung des Gerätestatus hin.
- `C2MI_CONTROL_SEND_COMMAND`: Weist auf einen Steuerbefehl von einem lokalen Controller hin.
- `C2MI_CONTROL_SEND_DEVICE_STATE_QUERY`: Weist auf eine Statusabfrage eines Steuergeräts von einem lokalen Controller hin.

reqPayload

Die Payload der Anfrage, normalerweise eine Zeichenfolge im JSON-Format.

RSP-Nutzlast

Die Antwortnutzlast, gefüllt mit den Komponenten der verwalteten Integration.

iotmi_client_event_t

Stellt ein Ereignis dar, das von den Komponenten der verwalteten Integration empfangen wurde.

event_id

Die eindeutige Kennung des Ereignisses.

length

Die Länge der Ereignisdaten.

data

Ein Zeiger auf die Ereignisdaten, einschließlich der`messageType`. Die folgende Liste zeigt die möglichen Werte.

- `C2MI_PROVISION_UGS_TASK`: Weist auf eine Bereitstellungsaufgabe für den UGS-Flow hin.
- `C2MI_PROVISION_SS_TASK`: Weist auf eine Bereitstellungsaufgabe für den SimpleSetup Flow hin.
- `C2MI_DE_PROVISION_PRE_ASSOCIATED_TASK`: Weist auf eine Aufgabe zur Aufhebung der Bereitstellung für ein bereits zugeordnetes Gerät hin.
- `C2MI_DE_PROVISION_ACTIVATED_TASK`: Weist auf eine Aufgabe zur Aufhebung der Bereitstellung für ein aktiviertes Gerät hin.
- `C2MI_DEVICE_ONBOARDED_RESPONSE`: Weist auf eine Reaktion beim Onboarding des Geräts hin.
- `C2MI_CONTROL_TASK`: Weist auf eine Kontrollaufgabe hin.
- `C2MI_CONTROL_EVENT_NOTIFICATION`: Weist auf eine Benachrichtigung über ein Steuerungsereignis für einen lokalen Controller hin.

ctx

Ein benutzerdefinierter Kontext, der mit dem Ereignis verknüpft ist.

Hubsteuerung

Hub Control ist eine Erweiterung des Endgeräte-SDK für verwaltete Integrationen, die es ermöglicht, eine Schnittstelle mit der `MQTTProxy` Komponente im Hub-SDK herzustellen. Mit der Hub-Steuerung können Sie Code mithilfe des Endgeräte-SDK implementieren und Ihren Hub über die Managed Integrations Cloud als separates Gerät steuern. Das Hub Control SDK wird als separates Paket innerhalb des Hub-SDK mit der Bezeichnung als bereitgestellt. `iot-managed-integrations-hub-control-x.x.x`

Themen

Voraussetzungen

Um die Hub-Steuerung einzurichten, benötigen Sie Folgendes:

- Ein Hub, der in das Hub-SDK, Version 0.4.0 oder höher, integriert ist.
- Laden Sie die neueste Version des Endgeräte-SDK von der herunter. AWS-Managementkonsole
- Eine MQTT-Proxykomponente, die auf dem Hub läuft, Version 0.5.0 oder höher.

SDK-Komponenten für Endgeräte

Verwenden Sie die folgenden Komponenten aus dem Endgeräte-SDK:

- Codegenerator für das Datenmodell
- Datenmodell-Handler

Da das Hub-SDK bereits über einen Onboarding-Prozess und eine Verbindung zur Cloud verfügt, benötigen Sie die folgenden Komponenten nicht:

- Bereitsteller
- PKCS-Schnittstelle
- Handler für Jobs
- MQTT-Agent

Integrieren Sie es in das Endgeräte-SDK

1. Folgen Sie den Anweisungen im Codegenerator für Datenmodell, um den Low-Level-C-Code zu generieren.
2. Folgen Sie den Anweisungen unter Integrieren des Endgeräte-SDK, um:

a. Richten Sie die Build-Umgebung ein

Erstellen Sie den Code auf Amazon Linux 2023/x86_64 als Entwicklungshost. Installieren Sie die erforderlichen Build-Abhängigkeiten:

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

b. Entwickeln Sie Hardware-Callback-Funktionen

Bevor Sie die Hardware-Callback-Funktionen implementieren, sollten Sie sich mit der Funktionsweise der API vertraut machen. In diesem Beispiel werden der On/Off Cluster und das OnOff Attribut verwendet, um eine Gerätefunktion zu steuern. Einzelheiten zur API finden Sie unterC-Funktion auf niedriger Ebene APIs.

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

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

c. Richten Sie Endpunkte ein und binden Sie Hardware-Callback-Funktionen ein

Nachdem Sie die Funktionen implementiert haben, erstellen Sie Endpunkte und registrieren Sie Ihre Callbacks. Erledigen Sie diese Aufgaben:

i. Erstellen Sie einen Geräteagenten

ii. Füllen Sie die Callback-Funktionspunkte für jede Clusterstruktur aus, die Sie unterstützen möchten

iii. Richten Sie Endpunkte ein und registrieren Sie unterstützte Cluster

```
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);
}
```

Beispiel: Hub-Kontrolle erstellen

Die Hub-Steuerung wird als Teil des Hub-SDK-Pakets bereitgestellt. Das Hub Control-Unterpaket ist mit anderen Bibliotheken gekennzeichnet `iot-managed-integrations-hub-control-x.x.x` und enthält andere Bibliotheken als das unveränderte Geräte-SDK.

1. Verschieben Sie die mit dem Code generierten Dateien in den folgenden Ordner`example`:

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

2. Führen Sie die folgenden Befehle aus, um die Hub-Steuerung zu erstellen:

```
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. Führen Sie die Beispiele mit der `MQTTProxy` Komponente auf dem Hub aus, während die `MQTTProxy` Komponenten `HubOnboarding` und ausgeführt werden.

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

Informationen Datenmodell für verwaltete Integrationen zum Datenmodell finden Sie unter. Folgen Sie Schritt 5 unterBeginnen Sie mit dem End Device SDK, um Endgeräte einzurichten und die Kommunikation zwischen dem Endbenutzer und zu verwalten. iot-managed-integrations

Unterstützte Beispiele

Die folgenden Beispiele wurden erstellt und getestet:

- iotmi_device_dm_air_purifier_demo
- Grundlegende Diagnose für iotmi_device_devices
- iotmi_device_dm_camera_demo

Unterstützte Plattformen

In der folgenden Tabelle sind die unterstützten Plattformen für die Hub-Steuerung aufgeführt.

Architektur	Betriebssystem	GCC-Version	Binutils-Version
X86_64	Linux	10.5.0	2,37
aarch64	Linux	10,5,0	2,37

CloudWatch Logs aktivieren

Das Hub-SDK bietet umfassende Protokollierungsfunktionen. Standardmäßig schreibt das Hub-SDK Protokolle in das lokale Dateisystem. Sie können jedoch die Cloud-API nutzen, um das Log-Streaming in CloudWatch Logs zu konfigurieren, was Folgendes bietet:

- Geräteleistung überwachen: Erfassen Sie detaillierte Laufzeitprotokolle für eine proaktive Geräteverwaltung. Ermöglichen Sie erweiterte Protokollanalysen und -überwachungen für Ihre gesamte Geräteflotte
- Probleme beheben: Generieren Sie detaillierte Protokolleinträge für eine schnelle Diagnoseanalyse. Zeichnen Sie Ereignisse auf System- und Anwendungsebene auf, um sie eingehend zu untersuchen.
- Flexible und zentralisierte Protokollierung: Protokollverwaltung aus der Ferne ohne direkten Gerätezugriff. Aggregieren Sie Protokolle von mehreren Geräten in einem einzigen, durchsuchbaren Repository.

Voraussetzungen

- Integrieren Sie das verwaltete Gerät in die Cloud. Details dazu finden Sie unter Einrichtung des Hub-Onboardings.
- Überprüfen Sie den Start und die erfolgreiche Initialisierung des Hub-Agenten. Details dazu finden Sie unter Installieren und validieren Sie das Hub-SDK für verwaltete Integrationen.

> **Note**
>
> Einzelheiten zum Erstellen von Protokollierungskonfigurationen finden Sie unter PutRuntimeLogConfiguration API.

> **Warning**
>
> Die Aktivierung von Protokollen wird auf die gestaffelte Kontingentmessung angerechnet. Eine Erhöhung der Protokollebenen führt zu einem höheren Nachrichtenvolumen und zusätzlichen Kosten.

Richten Sie die Hub SDK-Protokollkonfigurationen ein

Konfigurieren Sie die Hub-SDK-Protokolleinstellungen, indem Sie die API aufrufen, um die Laufzeitprotokollkonfiguration einzurichten.

Example Beispiel für eine API-Anfrage

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

RuntimeLogConfigurations Attribute

Die folgenden Attribute sind optional und können in der `RuntimeLogConfigurations` API konfiguriert werden.

LogLevel

Legt den Mindestschweregrad für Runtime-Traces fest. Werte: `DEBUG, ERROR, INFO, WARN`

Standard: `WARN` (veröffentlichter Build)

LogFlushLevel

Legt den Schweregrad für die sofortige Übertragung von Daten in den lokalen Speicher fest. Werte: `DEBUG, ERROR, INFO, WARN`

Standard: `DISABLED`

LocalStoreLocation

Gibt den Speicherort für Runtime-Traces an. Standard: `/var/log/awsiotmi`

- Aktives Protokoll: `/var/log/awsiotmi/ManagedIntegrationsDeviceSdkHub.log`

- Rotierte Protokolle: `/var/log/awsiotmi/ManagedIntegrationsDeviceSdkHub.N.log` (N gibt die Reihenfolge der Rotation an)

LocalStoreFileRotationMaxBytes

Löst die Dateirotation aus, wenn die aktuelle Datei die angegebene Größe überschreitet.

> **Important**
>
> Um eine optimale Effizienz zu erzielen, sollten Sie die Dateigröße unter 125 KB halten. Werte über 125 KB werden automatisch begrenzt.

LocalStoreFileRotationMaxFiles,

Legt die maximale Anzahl von Rotationsdateien fest, die vom Log-Daemon zulässig sind.

UploadLog

Steuert die Runtime-Trace-Übertragung in die Cloud. Protokolle werden in der Gruppe `/aws/iotmanagedintegration` CloudWatch Protokolle gespeichert.

Standard: `false`.

UploadPeriodMinutes

Definiert die Häufigkeit von Runtime-Trace-Uploads. Standard: `5`

DeleteLocalStoreAfterUpload

Steuert das Löschen von Dateien nach dem Upload. Standard: `true`

> **Note**
>
> Wenn der Wert auf false gesetzt ist, werden hochgeladene Dateien umbenannt in: `/var/log/awsiotmi/ManagedIntegrationsDeviceSdkHub.uploaded.{uploaded_timestamp}`

Beispiel für eine Protokolldatei

Unten finden Sie ein Beispiel für eine CloudWatch Log-Datei:

Unterstützte Zigbee- und Z-Wave-Gerätetypen

Auf dieser Seite sind die Gerätetypen aufgeführt, die mit Hubs verbunden sind und mit verwalteten Integrationen getestet wurden und unterstützt werden. Managed Integrations unterstützt sowohl als auch Einfache Einrichtung (SS) für diese Geräte. Benutzergeführte Einrichtung (UGS)

In dieser Tabelle sind die unterstützten Zigbee-Geräte aufgeführt.

ZigBee-Gerätetyp	Unterstützte Funktionen
Intelligente Glühbirne/Dimmbares Licht/RGB-Licht	OnOff, LevelControl, ColorControl
Intelligenter Stecker	OnOff
Intelligenter Schalter	OnOff
LED-Streifen	OnOff, LevelControl, ColorControl
Wasserventil	OnOff
Kühlerventil	Thermostat OnOff, Timer
Thermostat	Thermostat, FanControl OnOff, Zeitschaltuhr
Garagentoröffner	WindowCovering, OnOff, LevelControl
Rauchmelder	BooleanState, OnOff TemperatureMeasurement, Timer, Rauch COAlarm
Bewegungssensor	BooleanState
Anwesenheits- und Präsenzsensor	BooleanState, OccupancySensing
Tür- und Fenstersensor	BooleanState
Wasserlecksensor	BooleanState
Schwingungssensor	BooleanState

ZigBee-Gerätetyp	Unterstützte Funktionen
Temperatur- und Feuchtigkeitssensor	TemperatureMeasurement, RelativeHumidityMeasurement

In dieser Tabelle sind die unterstützten Z-Wave-Geräte aufgeführt.

Z-Wave-Gerätetyp	Unterstützte Funktionen
Intelligente Glühbirne/Dimmbares Licht	OnOff, LevelControl
Intelligenter Stecker	OnOff
Steuerung für Garagentore	OnOff, LevelControl
Energiezähler	ElectricalEnergyMeasurement, ElectricalPowerMeasurement
Batterie	LevelControl
Sirene	LevelControl
Bewegungssensor	BooleanState
Tür- und Fenstersensor	BooleanState
Wasserlecksensor	BooleanState
Temperatursensor	TemperatureMeasurement
CO-Sensor	Rauch COAlarm
Rauchsensor	Rauch COAlarm

Verwaltete Integrationen auf Raspberry Pi ausführen

> **Note**
>
> Diese Implementierung von AWS IoT Hub SDK auf Raspberry Pi ist ein Demonstrationsprojekt, das nur zu Lern- und Testzwecken bestimmt ist und nicht für den Einsatz in Produktionsumgebungen vorgesehen ist. Stellen Sie für diese Demo die folgenden Konfigurationen ein, um die Entwicklung zu vereinfachen:
> AWS Speicherung von Anmeldeinformationen: Nur zu Demo-Zwecken werden Anmeldeinformationen und Zertifikate an einem Ort gespeichert, auf den Zugriff zugegriffen werden kann, um Tests und Entwicklung zu erleichtern. Produktionsumgebungen müssen sichere Speicherlösungen wie AWS Secrets Manager Systems Manager Parameter Store verwenden. Sie müssen Verschlüsselung im Ruhezustand implementieren und AWS IoT Sicherheitsrichtlinien einhalten.
> Container-Rechte: Die Demo wird mit erhöhten Rechten ausgeführt, um uneingeschränkten Zugriff auf Host-Ressourcen zu ermöglichen und die Entwicklungsabläufe zu vereinfachen. In der Produktion sollten Container mit minimalen erforderlichen Rechten betrieben werden.
> Konfiguration der Netzwerkbrücke: Die Demo verwendet eine Netzwerkbrückenkonfiguration, die internen Netzwerkverkehr für einfacheres Debuggen und Überwachen verfügbar macht. Implementieren Sie in Produktionsumgebungen eine angemessene Netzwerkisolierung und -segmentierung, um unbefugten Zugriff auf den internen Netzwerkverkehr zu verhindern.
> USB-Geräteberechtigungen: Der uneingeschränkte Zugriff auf USB-Geräte ist aktiviert, um den einfachen Anschluss von Entwicklungsperipheriegeräten und Testgeräten zu ermöglichen. Implementieren Sie für die Produktion strenge Kontrollen und Validierungen von USB-Geräten, um Geräte-Spoofing-Angriffe zu verhindern.
> Diese Konfigurationen ermöglichen unkomplizierte Tests und dürfen nicht in Produktionsumgebungen verwendet werden. Beachten Sie bei der Bereitstellung in der Produktionsumgebung die bewährten Sicherheitsmethoden, um eine Beeinträchtigung des Hostsystems und den unbefugten Zugriff auf Anmeldeinformationen zu verhindern.

Als Voraussetzung müssen Sie den Sonoff Zigbee USB-Dongle einrichten, bevor Sie den Raspberry Pi einrichten.

Flash-Firmware auf den Sonoff ZigBee USB-Dongle

Voraussetzungen

- Sonoff ZigBee USB-Dongle
- Windows: Installieren Sie den universellen CP210x-Windows-Treiber

Flashen Sie die Firmware

1. Laden Sie den Zigbee-Dongle Firmware Build 7.4.1.0 herunter.
2. Öffnen Sie den Silabs Firmware Flasher.
3. Connect den Sonoff Zigbee USB-Dongle mit Ihrem Computer.
4. Scrollen Sie und suchen Sie -E. ZBDongle
5. Wählen Sie Connect aus.
6. Warten Sie, bis das Gerät eine Verbindung hergestellt hat.
7. Wählen Sie Firmware ändern.
8. Wählen Sie Eigene Firmware hochladen aus.
9. Suchen Sie den Speicherort für den Download von Zigbee Dongle Firmware Build 7.4.1.0 und wählen Sie ihn aus.
10. Klicken Sie auf Installieren.
11. Warten Sie, bis die Firmware installiert ist.
12. Wählen Sie Weiter, wenn die Installation abgeschlossen ist.

Der Dongle ist jetzt einsatzbereit.

Wählen Sie zwischen den unten aufgeführten Optionen, um das Managed Integrations Hub SDK auf Ihrem Raspberry Pi auszuführen. Die Schritte zur Einrichtung und Validierung für beide Ansätze sind unten aufgeführt.

Themen

- Hub SDK-Image für verwaltete Integrationen auf Raspberry Pi

- Verwaltete Integrationen Hub SDK Docker-Container auf Raspberry Pi
- Demo-Anwendung für verwaltete Integrationen

Hub SDK-Image für verwaltete Integrationen auf Raspberry Pi

> **Note**
>
> Diese Implementierung des AWS IoT Hub SDK auf Raspberry Pi ist ein Demonstrationsprojekt, das nur zu Lern- und Testzwecken bestimmt ist und nicht für den Einsatz in Produktionsumgebungen vorgesehen ist. Stellen Sie für diese Demo die folgenden Konfigurationen ein, um die Entwicklung zu vereinfachen:
> AWS Speicherung von Anmeldeinformationen: Nur zu Demo-Zwecken werden Anmeldeinformationen und Zertifikate an einem Ort gespeichert, auf den Zugriff zugegriffen werden kann, um Tests und Entwicklung zu erleichtern. Produktionsumgebungen müssen sichere Speicherlösungen wie AWS Secrets Manager Systems Manager Parameter Store verwenden. Sie müssen Verschlüsselung im Ruhezustand implementieren und AWS IoT Sicherheitsrichtlinien einhalten.
> Container-Rechte: Die Demo wird mit erhöhten Rechten ausgeführt, um uneingeschränkten Zugriff auf Host-Ressourcen zu ermöglichen und die Entwicklungsabläufe zu vereinfachen. In der Produktion sollten Container mit minimalen erforderlichen Rechten betrieben werden.
> Konfiguration der Netzwerkbrücke: Die Demo verwendet eine Netzwerkbrückenkonfiguration, die internen Netzwerkverkehr für einfacheres Debuggen und Überwachen verfügbar macht. Implementieren Sie in Produktionsumgebungen eine angemessene Netzwerkisolierung und -segmentierung, um unbefugten Zugriff auf den internen Netzwerkverkehr zu verhindern.
> USB-Geräteberechtigungen: Der uneingeschränkte Zugriff auf USB-Geräte ist aktiviert, um den einfachen Anschluss von Entwicklungsperipheriegeräten und Testgeräten zu ermöglichen. Implementieren Sie für die Produktion strenge Kontrollen und Validierungen von USB-Geräten, um Geräte-Spoofing-Angriffe zu verhindern.
> Diese Konfigurationen ermöglichen unkomplizierte Tests und dürfen nicht in Produktionsumgebungen verwendet werden. Beachten Sie bei der Bereitstellung in der Produktionsumgebung die bewährten Sicherheitsmethoden, um eine Beeinträchtigung des Hostsystems und den unbefugten Zugriff auf Anmeldeinformationen zu verhindern.

Voraussetzungen

Erfüllen Sie die folgenden Anforderungen, bevor Sie das Raspberry Pi-Image bereitstellen:

- Laden Sie den Raspberry Pi Imager herunter und installieren Sie ihn.
- Besorgen Sie sich eine SD-Karte.
- Richten Sie einen Raspberry Pi 5 mit einer 2,4-GHz-64-Bit-Quad-Core-CPU (8 GB RAM) ein.
- Connect einen Sonoff ZigBee USB-Dongle an.
- Flash-Firmware auf den Sonoff ZigBee USB-Dongle.
- Connect einen SLUSB001A-Dongle von Silicon Labs an.
- Eröffnen Sie ein Konto. AWS
- Installieren Sie die neueste Version von AWS CLI aus der AWS CLI Befehlsreferenz für verwaltete Integrationen.

Flashen Sie ein Raspberry Pi-Image auf einer neuen SD-Karte

Flashen Sie das Image der verwalteten Integrationen wie folgt auf Ihre SD-Karte:

1. Laden Sie das Raspberry Pi Hub SDK-Image für verwaltete Integrationen herunter.
2. Starten Sie Raspberry Pi Imager auf Ihrem Desktop.
3. Stecken Sie die SD-Karte in den eingebauten SD-Kartenleser Ihres Computers oder in den externen USB-Kartenleser.
4. Wählen Sie Gerät wählen → Raspberry Pi 5.
5. Wählen Sie Betriebssystem auswählen → Benutzerdefiniert verwenden → Suchen Sie die Datei IotMI-HubSDK-RPi-Image -v1.0.0.img.gz → Öffnen.
6. Wählen Sie Speicher auswählen → Wählen Sie Ihren SD-Kartenleser aus.
7. Stellen Sie sicher, dass Ihre Konfiguration dem folgenden Bildschirm entspricht:

8. Klicken Sie auf Weiter.
9. Konfigurieren Sie die Einstellungen zur Anpassung des Betriebssystems:
   - Hostname: Wählen Sie Raspberrypi.
   - Nutzername und Passwort:
     - Aktivieren Sie „Benutzername und Passwort festlegen“:
     - Geben Sie für Username: ein`hub123456`.
     - Geben Sie für Passwort: ein`sh123456`.
   - Drahtloses LAN:

    - Aktivieren Sie „WLAN konfigurieren“.
    - Geben Sie die SSID und das Passwort Ihres Routers ein.

      Beispieleinstellungen:

      - SSID: `iotmi-tplink`
      - Passwort: `********` (Mindestens 8 Zeichen)
    - Stellen Sie Land ein: auf`US`.
  - Gebietsschemaeinstellungen festlegen:
    - Zeitzone einstellen: auf`America/Los Angeles`.
    - Stellen Sie das Tastaturlayout ein: auf`US`.
  - SSH:
    - Wählen Sie die Registerkarte Dienste.
    - Markieren Sie SSH aktivieren.
    - Wählen Sie Passwortauthentifizierung verwenden.
10. Bestätigen Sie alle Popups zur Betriebssystemanpassung und Datenlöschung.
11. Warten Sie, bis der Schreibvorgang abgeschlossen ist.
12. Überprüfen Sie den erfolgreichen Abschluss mit dem folgenden Bildschirm:

13. Klicken Sie auf Weiter.
14. Entfernen Sie die SD-Karte und legen Sie sie in Ihren Raspberry Pi ein.

Führen Sie das Hub SDK auf dem Raspberry Pi aus

Starten Sie die Hub SDK-Dienste auf Ihrem konfigurierten Raspberry Pi:

1. Legen Sie die vorbereitete SD-Karte in das Raspberry Pi 5-Gerät ein.
2. Connect den Sonoff Zigbee USB-Dongle und den Silicon Labs SLUSB001A-Dongle mit dem Raspberry Pi.
3. Schalten Sie den Raspberry Pi ein.
4. Stellen Sie sicher, dass sich der Raspberry Pi und Ihr Computer (von dem aus Sie SSH verwenden) im selben Netzwerk befinden.
5. Stellen Sie mithilfe der Anmeldeinformationen, die Sie bei der Image-Bereitstellung festgelegt haben, eine SSH-Verbindung zum Raspberry Pi her.

```
ssh username@hostname
```

6. Navigieren Sie zum Hub-SDK-Verzeichnis:

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

7. Schließen Sie das Hub-Onboarding-Setup ab, um Authentifizierungs- und Konfigurationsmaterial hinzuzufügen.

> **Note**
>
> Sie müssen sich in YUL oder in der DUB Region befinden, um diesen Schritt ausführen zu können.

8. Führen Sie das Hub-SDK aus:

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

Das System zeigt die folgende Antwort für einen erfolgreichen Start des Hub-SDK an:

```
-------Stopping SDK running processes---
-------Starting Hub SDK------------
-------Creating logs directory----------
Logs directory created.
-------Verifying Middleware paths-------
All middleware libraries exist
-------Verifying Middleware pre reqs---
AIPC and KVstroage directories exist
-------Starting HubOnboarding-----------
-------Starting MQTT Proxy-----------
-------Staring Log Daemon---
-------Starting Event Manager-----------
-------Starting Zigbee Service----------
--Checking Zigbee network information--
-------Starting Zwave Service----------
/data/aws/iotmi/middleware/AceZwave/bin /data/aws/iotmi
/data/aws/iotmi
-------Starting CDMB--------------------
-------Starting Agent--------------------
```

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

Nächste Schritte

Nachdem Sie das Hub-SDK erfolgreich gestartet haben, fahren Sie mit dem Onboarding und der Verwaltung des Geräts fort unterBenutzergeführte Einrichtung zur Einbindung und Bedienung von Geräten.

Verwaltete Integrationen Hub SDK Docker-Container auf Raspberry Pi

> **Note**
>
> Diese Implementierung des AWS IoT Hub SDK auf Raspberry Pi ist ein Demonstrationsprojekt, das nur zu Lern- und Testzwecken bestimmt ist und nicht für den Einsatz in Produktionsumgebungen vorgesehen ist. Stellen Sie für diese Demo die folgenden Konfigurationen ein, um die Entwicklung zu vereinfachen:
> AWS Speicherung von Anmeldeinformationen: Nur zu Demo-Zwecken werden Anmeldeinformationen und Zertifikate an einem Ort gespeichert, auf den Zugriff zugegriffen werden kann, um das Testen und Entwickeln zu erleichtern. Produktionsumgebungen

müssen sichere Speicherlösungen wie AWS Secrets Manager Systems Manager Parameter Store verwenden. Sie müssen Verschlüsselung im Ruhezustand implementieren und AWS IoT Sicherheitsrichtlinien einhalten.
Container-Rechte: Die Demo wird mit erhöhten Rechten ausgeführt, um uneingeschränkten Zugriff auf Host-Ressourcen zu ermöglichen und die Entwicklungsabläufe zu vereinfachen. In der Produktion sollten Container mit minimalen erforderlichen Rechten betrieben werden.
Konfiguration der Netzwerkbrücke: Die Demo verwendet eine Netzwerkbrückenkonfiguration, die internen Netzwerkverkehr für einfacheres Debuggen und Überwachen verfügbar macht. Implementieren Sie in Produktionsumgebungen eine angemessene Netzwerkisolierung und -segmentierung, um unbefugten Zugriff auf den internen Netzwerkverkehr zu verhindern.
USB-Geräteberechtigungen: Der uneingeschränkte Zugriff auf USB-Geräte ist aktiviert, um den einfachen Anschluss von Entwicklungsperipheriegeräten und Testgeräten zu ermöglichen. Implementieren Sie für die Produktion strenge Kontrollen und Validierungen von USB-Geräten, um Geräte-Spoofing-Angriffe zu verhindern.
Diese Konfigurationen ermöglichen unkomplizierte Tests und dürfen nicht in Produktionsumgebungen verwendet werden. Beachten Sie bei der Bereitstellung in der Produktionsumgebung die bewährten Sicherheitsmethoden, um eine Beeinträchtigung des Hostsystems und den unbefugten Zugriff auf Anmeldeinformationen zu verhindern.

Voraussetzungen

Für den Docker-Container sind die folgenden Voraussetzungen erforderlich.

- Laden Sie den Raspberry Pi Imager herunter und installieren Sie ihn.
- Besorgen Sie sich eine SD-Karte.
- Richten Sie einen Raspberry Pi 5 mit einer 2,4-GHz-64-Bit-Quad-Core-CPU (8 GB RAM) ein.
- Connect einen Sonoff ZigBee USB-Dongle an.
- Flash-Firmware auf den Sonoff ZigBee USB-Dongle.
- Connect einen SLUSB001A-Dongle von Silicon Labs an.
- Eröffnen Sie ein Konto. AWS
- Installieren Sie die neueste Version von AWS CLI aus der AWS CLI Befehlsreferenz für verwaltete Integrationen.
- SSH-Zugriff auf den Raspberry Pi mit IP-Adresse oder Hostname.

Verwenden Sie den Docker-Container Managed Integrations Hub SDK auf dem Raspberry Pi

1. Laden Sie verwaltete Integrationen Raspberry Pi Hub SDK Docker herunter.
2. Kopieren Sie die Datei mit SCP auf den Raspberry Pi:

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

3. Stellen Sie über SSH Connect zum Raspberry Pi her:

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

4. Installieren Sie Docker, falls nicht vorhanden:

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

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

# Log in again
```

5. Installieren Sie Docker Compose, falls nicht vorhanden:

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

6. Extrahieren Sie die Hub SDK-Dateien:

```
# Navigate to the home directory
cd

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

7. Navigieren Sie zum Hub-Docker-Verzeichnis:

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

8. Schließen Sie das Hub-Onboarding-Setup ab, um die Authentifizierung und die Einstellungen zu konfigurieren.

> **Note**
>
> Sie müssen sich in YUL oder in der DUB Region befinden, um diesen Schritt ausführen zu können.

9. Starten Sie den Docker-Container:

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

Erwartete Ausgabe:

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

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

Nachdem Sie das Hub-SDK erfolgreich gestartet haben, fahren Sie mit dem Onboarding und der Verwaltung von Geräten fort unterBenutzergeführte Einrichtung zur Einbindung und Bedienung von Geräten.

> **Note**
>
> - Führen Sie den folgenden Befehl aus, um auf die Docker-Container-Bash-Shell zuzugreifen:

```
docker compose exec hubsdk bash
```

- Führen Sie den folgenden Befehl aus, um den Container nach dem Neustart neu zu starten:

```
docker compose up -d
```

- Um das Hub-SDK zu aktualisieren, ersetzen Sie die Binärdateien im folgenden Ordner:

```
hub-docker/iotmi
```

- Gehen Sie wie folgt vor, um den Container sicher neu zu starten und gleichzeitig die Daten beizubehalten:

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

Demo-Anwendung für verwaltete Integrationen

Note

Diese Implementierung des AWS IoT Hub SDK auf Raspberry Pi ist ein Demonstrationsprojekt, das nur zu Lern- und Testzwecken bestimmt ist und nicht für den Einsatz in Produktionsumgebungen vorgesehen ist. Stellen Sie für diese Demo die folgenden Konfigurationen ein, um die Entwicklung zu vereinfachen:
AWS Speicherung von Anmeldeinformationen: Nur zu Demo-Zwecken werden Anmeldeinformationen und Zertifikate an einem Ort gespeichert, auf den Zugriff zugegriffen werden kann, um das Testen und Entwickeln zu erleichtern. Produktionsumgebungen müssen sichere Speicherlösungen wie AWS Secrets Manager Systems Manager Parameter Store verwenden. Sie müssen Verschlüsselung im Ruhezustand implementieren und AWS IoT Sicherheitsrichtlinien einhalten.
Container-Rechte: Die Demo wird mit erhöhten Rechten ausgeführt, um uneingeschränkten Zugriff auf Host-Ressourcen zu ermöglichen und die Entwicklungsabläufe zu vereinfachen. In der Produktion sollten Container mit den minimal erforderlichen Rechten betrieben werden.

Konfiguration der Netzwerkbrücke: Die Demo verwendet eine Netzwerkbrückenkonfiguration, die internen Netzwerkverkehr für einfacheres Debuggen und Überwachen verfügbar macht. Implementieren Sie in Produktionsumgebungen eine angemessene Netzwerkisolierung und -segmentierung, um unbefugten Zugriff auf den internen Netzwerkverkehr zu verhindern.
USB-Geräteberechtigungen: Der uneingeschränkte Zugriff auf USB-Geräte ist aktiviert, um den einfachen Anschluss von Entwicklungsperipheriegeräten und Testgeräten zu ermöglichen. Implementieren Sie für die Produktion strenge Kontrollen und Validierungen von USB-Geräten, um Geräte-Spoofing-Angriffe zu verhindern.
Diese Konfigurationen ermöglichen unkomplizierte Tests und dürfen nicht in Produktionsumgebungen verwendet werden. Beachten Sie bei der Bereitstellung in der Produktionsumgebung die bewährten Sicherheitsmethoden, um eine Beeinträchtigung des Hostsystems und den unbefugten Zugriff auf Anmeldeinformationen zu verhindern.

Bei der Demo-Anwendung handelt es sich um eine auf React basierende Demo-Anwendung, die Funktionen von Managed Integrations für die Verwaltung von Smart-Home-Geräten vorstellt. Diese Anwendung demonstriert das Onboarding, die Steuerung und die Überwachung von Geräten für Z-Wave- und Zigbee-Geräte über eine moderne Weboberfläche.

Voraussetzungen

- Eröffnen Sie ein AWS Konto.
- Erstellen Sie einen Anmeldeinformationsspeicher und fügen Sie den Anmeldeinformationsspeicher zu Ihrem Hub hinzu.
- Schließen Sie das Hub-Onboarding-Setup ab.
- Node.js 18+ und npm.
- Installieren Sie die neueste Version von AWS CLI aus der Befehlsreferenz für verwaltete Integrationen. AWS CLI
- Moderner Webbrowser (Chrome, Firefox, Safari, Edge)

Installieren und konfigurieren Sie die Anwendung

1. Laden Sie die Demo-Anwendung Managed Integrations herunter.
2. Extrahieren Sie das Paket:

```
cd ~/Downloads
```

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

3. Installieren Sie die Abhängigkeiten:

```
npm install
```

4. Erstellen Sie eine `.env` Datei im Stammverzeichnis:

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

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

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

5. Erstellen und starten Sie die Anwendung:

```
npm start
```

6. Greifen Sie auf die Anwendung zu unter:

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

Preisinformationen finden Sie im Abschnitt Verwaltete Integrationen auf der Preisseite für AWS IoT Gerätemanagement.

Offboard-Hub für verwaltete Integrationen

Überblick über den Offboard-Prozess des Hub SDK

Der Hub-Offboarding-Prozess entfernt einen Hub aus dem AWS Cloud Verwaltungssystem. Wenn die Cloud eine DeleteManagedThingAnfrage sendet, werden mit dem Prozess zwei Hauptziele erreicht:

Geräteseitige Aktionen:

- Setzen Sie den internen Status des Hubs zurück
- Löschen Sie alle lokal gespeicherten Daten
- Bereiten Sie das Gerät auf ein mögliches future Re-Onboarding vor

Aktionen auf der Cloud-Seite:

- Entfernen Sie alle mit dem Hub verknüpften Cloud-Ressourcen
- Vollständige Trennung vom vorherigen Konto

Kunden initiieren das Offboarding von Hubs in der Regel, wenn:

- Das dem Hub zugeordnete Konto ändern
- Ersetzen eines vorhandenen Hubs durch ein neues Gerät

Der Prozess gewährleistet einen sauberen, sicheren Übergang zwischen Hub-Konfigurationen und ermöglicht so eine nahtlose Geräteverwaltung und Kontoflexibilität.

Voraussetzungen

- Sie müssen über einen integrierten Hub verfügen. Anweisungen finden Sie unter Einrichtung des Hub-Onboardings.
- Vergewissern Sie sich, dass in der `iotmi_config.json` Datei unter data/aws/iotmi/config//Folgendes `iot_provisioning_state` angezeigt wird`PROVISIONED`.
- Stellen Sie sicher, dass die permanenten Zertifikate und Schlüssel, auf die in der verwiesen wird, in den angegebenen Pfaden `iotmi_config.json` vorhanden sind.
- Stellen Sie sicher HubOnboarding, dass Agent, Provisioner und MQTT-Proxy korrekt konfiguriert sind und ausgeführt werden.
- Stellen Sie sicher, dass der Hub keine untergeordneten Geräte hat. Verwenden Sie die DeleteManagedThingAPI, um alle untergeordneten Geräte zu entfernen, bevor Sie fortfahren.

Offboard-Prozess für das Hub-SDK

Gehen Sie wie folgt vor, um den Hub zu entfernen:

Rufen Sie die hub_managed_thing-ID ab

Die `iotmi_config.json` Datei wird verwendet, um die verwaltete Ding-ID für einen Managed Integrations Hub zu speichern. Diese Kennung ist eine wichtige Information, die es dem Hub ermöglicht, mit dem AWS IoT Managed Integrations Service zu kommunizieren. Die ID des verwalteten Objekts wird im Abschnitt rw (Lese-/Schreibzugriff) der JSON-Datei unter dem Feld gespeichert. `managed_thing_id` Dies ist in der folgenden Beispielkonfiguration zu sehen:

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

Befehl an den Offboard-Hub senden

Verwenden Sie Ihre Kontoanmeldeinformationen und führen Sie den Befehl mit den im vorherigen Abschnitt `managed_thing_id` abgerufenen aus:

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

Stellen Sie sicher, dass der Hub ausgelagert wurde

Verwenden Sie Ihre Kontoanmeldeinformationen und führen Sie den Befehl mit den im vorherigen `managed_thing_id` Abschnitt abgerufenen aus:

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

Erfolgs- und Misserfolgszenarien

Erfolgsszenario

Wenn der Befehl zum Offboarden des Hubs erfolgreich war, wird die folgende Beispielantwort erwartet:

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

Außerdem `iotmi_config.json` würde das folgende Beispiel beobachtet werden, wenn der Befehl zum Offboarding des Hubs erfolgreich war. Stellen Sie sicher, dass der Abschnitt rw nur `iot_provisioning_state` und optional Metadaten enthält. Das Fehlen von Metadaten ist akzeptabel. `iot_provisioning_state`muss NOT_PROVISIONED sein.

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

Fehlerszenario

Wenn der Befehl zum Offboarding des Hubs nicht erfolgreich war, wird die folgende Beispielantwort erwartet:

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

- Falls ja ProvisioningStatus`DELETE_IN_PROGRESS`, folgen Sie den Anweisungen unter Hub-Wiederherstellung.
- ProvisioningStatusIst dies nicht der Fall`DELETE_IN_PROGRESS`, ist der Befehl zum Offboarding des Hubs entweder in der Managed Integrations Cloud fehlgeschlagen oder wurde von der Managed Integrations Cloud nicht empfangen. Folgen Sie den Anweisungen unter Hub-Wiederherstellung.
- Wenn das Offboarding nicht erfolgreich war, sieht Ihre `iotmi_config.json` Datei wie die folgende Beispieldatei aus.

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

```
            "iot_permanent_cert_path": "PATH",
            "iot_permanent_pk_path": "PATH",
            "metadata": {
                "last_updated_epoch_time": 1747766125
            }
        }
}
```

(Optional) Nach dem Offboarding des Hub-SDK

> **Important**
>
> In den folgenden Szenarien sind optionale Maßnahmen aufgeführt, die Sie ergreifen können, wenn das Offboarding des Hub-SDK fehlgeschlagen ist oder wenn Sie Ihren Hub nach dem Offboarding wieder einbinden möchten.

Erneut einsteigen

Wenn das Offboarding erfolgreich war, führen Sie das Onboarding Ihres Hub-SDK wie folgt aus Schritt 3: Erstellen eines verwalteten Objekts (Flottenbereitstellung) und dem Rest des Onboard-Prozesses aus.

Hub-Wiederherstellung

Das Offboarding des Geräte-Hubs war erfolgreich und das Cloud-Offboarding schlägt fehl

Wenn der GetManagedThingAPI-Aufruf keine `Managed Thing resource not found` Nachricht zurückgibt, die Datei aber ausgelagert wurde`iotmi_config.json`. Eine JSON-Beispieldatei finden Sie unter Erfolgsszenario.

Informationen zur Wiederherstellung nach diesem Szenario finden Sie unter Löschen erzwingen.

Das Offboarding des Geräte-Hubs schlägt fehl

In diesem Szenario wurde die Datei nicht `iotmi_config.json` korrekt ausgelagert. Eine JSON-Beispieldatei finden Sie unter Fehlerszenario.

Informationen zur Wiederherstellung nach diesem Szenario finden Sie unter Löschen erzwingen. Wenn `iotmi_config.json` der Hub immer noch nicht ausgelagert ist, muss er auf die Werkseinstellungen zurückgesetzt werden.

Das Offboarding des Geräte-Hubs und das Cloud-Offboarding schlagen fehl

In diesem Szenario `iotmi_config.json` wird immer noch kein Offboarding durchgeführt, und der Hub-Status lautet entweder oder. `ACTIVATED` `DISCOVERED`

Informationen zur Wiederherstellung nach diesem Szenario finden Sie unter Löschen erzwingen. Wenn das erzwungene Löschen fehlschlägt oder immer noch nicht offline `iotmi_config.json` ist, muss der Hub auf die Werkseinstellungen zurückgesetzt werden.

Der Hub ist offline und der Hub-Status ist DELETE_IN_PROGRESS

In diesem Szenario ist der Hub offline und die Cloud erhält einen Offboarding-Befehl.

Informationen zur Wiederherstellung nach diesem Szenario finden Sie unter Löschen erzwingen.

Löschen erzwingen

Gehen Sie wie folgt vor, um Cloud-Ressourcen ohne erfolgreiches Device-Hub-Offboarding zu löschen. Dieser Vorgang kann zu Inkonsistenzen zwischen dem Cloud- und dem Gerätestatus führen, was möglicherweise zu Problemen bei future Vorgängen führen kann.

Rufen Sie die DeleteManagedThing API mit dem Hub-Parameter `managed_thing_id` und dem Force-Parameter auf:

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

Rufen Sie als Nächstes die GetManagedThing API auf und überprüfen Sie, ob sie zurückkehrt`Managed Thing resource not found`. Dies bestätigt, dass die Cloud-Ressourcen gelöscht wurden.

> **Note**
>
> Dieser Ansatz wird nicht empfohlen, da er zu Inkonsistenzen zwischen dem Cloud- und dem Gerätestatus führen kann. Im Allgemeinen ist es besser, für ein erfolgreiches Offboarding des Device-Hubs zu sorgen, bevor Sie versuchen, die Cloud-Ressourcen zu löschen.

Protokollspezifische Middleware

> **Important**
>
> Die hier bereitgestellte Dokumentation und der Code beschreiben eine Referenzimplementierung der Middleware. Sie wird Ihnen nicht als Teil des SDK zur Verfügung gestellt.

Die protokollspezifische Middleware spielt eine entscheidende Rolle bei der Interaktion mit den zugrunde liegenden Protokollstapeln. Sowohl die Komponenten zur Geräteeinbindung als auch zur Gerätesteuerung des Hub-SDK für verwaltete Integrationen verwenden es für die Interaktion mit dem Endgerät.

Die Middleware erfüllt die folgenden Funktionen.

- Abstrahiert die Protokollstapel verschiedener Anbieter APIs vom Gerät, indem ein gemeinsamer Satz von Protokollen bereitgestellt wird. APIs
- Stellt die Verwaltung der Softwareausführung wie Thread-Scheduler, Verwaltung von Ereigniswarteschlangen und Datencache bereit.

Middleware-Architektur

Das folgende Blockdiagramm stellt die Architektur der Zigbee-Middleware dar. Die Architektur von Middlewares anderer Protokolle wie Z-Wave ist ebenfalls ähnlich.

Die protokollspezifische Middleware besteht aus drei Hauptkomponenten.

- ACS Zigbee DPK: Das Zigbee Device Porting Kit (DPK) wird verwendet, um eine Abstraktion von der zugrunde liegenden Hardware und dem Betriebssystem zu ermöglichen und so die Portabilität zu ermöglichen. Im Grunde kann dies als Hardware-Abstraktionsschicht (HAL) betrachtet werden, die ein gemeinsames System APIs zur Steuerung und Kommunikation mit Zigbee-Funkgeräten verschiedener Hersteller bereitstellt. Die Zigbee-Middleware enthält die DPK-API-Implementierung für das Zigbee-Anwendungsframework von Silicon Labs.
- ACS Zigbee-Dienst: Der Zigbee-Dienst wird als dedizierter Daemon ausgeführt. Er umfasst einen API-Handler, der die API-Aufrufe von Client-Anwendungen über die IPC-Kanäle bedient. AIPC wird

als IPC-Kanal zwischen dem Zigbee-Adapter und dem Zigbee-Dienst verwendet. Es bietet weitere Funktionen wie die Verarbeitung beider async/sync Befehle, die Verarbeitung von Ereignissen aus der HAL und die Verwendung von ACS Event Manager für die Registrierung/Veröffentlichung von Ereignissen.

- ACS Zigbee-Adapter: Der Zigbee-Adapter ist eine Bibliothek, die innerhalb des Anwendungsprozesses ausgeführt wird (in diesem Fall ist die Anwendung das CDMB-Plugin). Der ZigBee-Adapter stellt eine Reihe davon bereit APIs , die von Client-Anwendungen wie CDMB/ Provisioner Protokoll-Plugins zur Steuerung und Kommunikation mit dem Endgerät verwendet werden.

End-to-end Beispiel für einen Middleware-Befehlsablauf

Hier ist ein Beispiel für den Befehlsfluss durch die Zigbee-Middleware.

Hier ist ein Beispiel für den Befehlsfluss durch die Z-Wave-Middleware.

Protokollspezifische Middleware-Code-Organisation

Dieser Abschnitt enthält Informationen zum Speicherort des Codes für jede Komponente im `IotManagedIntegrationsDeviceSDK-Middleware` Repository. Das Folgende ist ein Beispiel für die Ordnerstruktur in diesem Repository.

```
./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
```

Themen

- Organisation des Zigbee-Middleware-Codes
- Organisation des Z-Wave-Middleware-Codes

Organisation des Zigbee-Middleware-Codes

Im Folgenden wird die Organisation des Zigbee-Referenz-Middleware-Codes dargestellt.

Themen

- ACS Zigbee DPK
- Silicon Labs Zigbee SDK
- ACS ZigBee-Dienst
- ACS ZigBee-Adapter

ACS Zigbee DPK

Der Code für Zigbee DPK befindet sich in dem Verzeichnis, das im folgenden Beispiel aufgeführt ist:

```
./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

Das Silicon Labs SDK befindet sich im `IotManagedIntegrationsDeviceSDK-Middleware/example-iot-ace-z3-gateway` Ordner. Diese ACS ZigBee DPK-Schicht ist für dieses Silicon Labs SDK implementiert.

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

ACS ZigBee-Dienst

Der Code für den Zigbee-Dienst befindet sich im `IotManagedIntegrationsDeviceSDK-Middleware/example-iot-ace-general/middleware/zigbee/` Ordner. Die Ordner `src` und `include` Unterordner an diesem Speicherort enthalten alle Dateien, die sich auf den ACS Zigbee-Dienst beziehen.

```
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-Adapter

Der Code für den ACS ZigBee-Adapter befindet sich im Ordner. `IotManagedIntegrationsDeviceSDK-Middleware/example-iot-ace-general/middleware/zigbee/api` Der Ordner `src` und die `include` Unterordner an diesem Speicherort enthalten alle Dateien, die sich auf die ACS ZigBee Adaptor-Bibliothek beziehen.

```
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
```

Organisation des Z-Wave-Middleware-Codes

Im Folgenden wird die Z-Wave-Referenz-Middleware-Code-Organisation dargestellt.

Themen

- ACS Z-Wave DPK
- Silicon Labs ZWare und Zip Gateway
- ACS Z-Wave-Dienst
- ACS Z-Wave-Adapter

ACS Z-Wave DPK

Der Code für Z-Wave DPK befindet sich im Ordner. `IotManagedIntegrationsDeviceSDK-Middleware/example-iot-ace-dpk/example/dpk/ace_hal/zwave`

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

```
|–       |– fxnZwaveClient
|–       |– include
|–       |– zware
```

Silicon Labs ZWare und Zip Gateway

Der Code für die Silicon Labs ZWare und Zip Gateway befindet sich im `IotManagedIntegrationsDeviceSDK-Middleware/example-iot-ace-z3-gateway` Ordner. Diese ACS Z-Wave DPK-Schicht ist für das Z-Wave C- APIs und Zip-Gateway implementiert.

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

ACS Z-Wave-Dienst

Der Code für den Z-Wave-Dienst befindet sich in dem Ordner, der `IoTmanagedintegrationsMiddlewares/exampleiot-ace-zwave-mw/` im Ordner aufgeführt ist. Die `include` Ordner `src` und an diesem Speicherort enthalten alle Dateien, die sich auf den ACS Z-Wave-Dienst beziehen.

```
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-Adapter

Der Code für den ACS ZigBee-Adapter befindet sich im Ordner. `IotManagedIntegrationsDeviceSDK-Middleware/example-iot-ace-zwave-mw/cli/` Der `include` Ordner `src` und an diesem Speicherort enthält alle Dateien, die sich auf die ACS Z-Wave Adaptor-Bibliothek beziehen.

```
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
```

Integrieren Sie Middleware in das SDK

Die Middleware-Integration auf dem neuen Hub wird in den folgenden Abschnitten erörtert.

Themen

- API-Integration des Device Porting Kit (DPK)
- Referenzimplementierung und Organisation des Codes

API-Integration des Device Porting Kit (DPK)

Um das SDK eines beliebigen Chipsatzherstellers in die Middleware zu integrieren, wird von der DPK-Schicht (Device Porting Kit) in der Mitte eine Standard-API-Schnittstelle bereitgestellt. Die Dienstleister für verwaltete Integrationen oder ODMs müssen diese auf der APIs Grundlage des Hersteller-SDK implementieren, das von den in ihren IoT-Hubs verwendeten Zigbee/Z-wave/Wi Fi-Chipsätzen unterstützt wird.

Referenzimplementierung und Organisation des Codes

Mit Ausnahme der Middleware können alle anderen Device SDK-Komponenten, wie z. B. die verwalteten Integrationen Device Agent und Common Data Model Bridge (CDMB), ohne Änderungen verwendet werden und müssen nur querkompiliert werden.

Die Implementierung der Middleware basiert auf dem Silicon Labs SDK für Zigbee und Z-Wave. Wenn die im neuen Hub verwendeten Z-Wave- und Zigbee-Chipsätze vom in der Middleware vorhandenen Silicon Labs SDK unterstützt werden, kann die Referenz-Middleware ohne Änderungen verwendet werden. Sie müssen nur die Middleware querkompilieren und sie kann dann auf dem neuen Hub ausgeführt werden.

DPK (Device Porting Kit) APIs für Zigbee finden Sie unter`acehal_zigbee.c`, und die Referenzimplementierung des DPK APIs befindet sich im Ordner. `zigbee`

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

```
|—   |—   |— zbd_callbacks.h
|—   |—   |— zbd_common.h
|—   |—   |— zbd_network_creator.h
|—   |—   |— zbd_power_settings.h
|—   |—   |— zbd_zts.h
```

DPK APIs für Z-Wave finden Sie im Ordner, `acehal_zwave.c` und die Referenzimplementierung des DPK APIs befindet sich im Ordner. `zwaved`

```
IotManagedIntegrationsDeviceSDK-Middleware/example-iot-ace-dpk/example/dpk/ace_hal/
zwave/
|— CMakeLists.txt
|— include
|—   |— zwave_log.h
|— src
|—   |— acehal_zwave.c
|— zwaved
|—   |— CMakeLists.txt
|—   |— fxnZwaveClient
|—   |—   |— zwave_client.c
|—   |—   |— zwave_client.h
|—   |— include
|—   |—   |— zwaved_cc_intf_api.h
|—   |—   |— zwaved_common_utils.h
|—   |—   |— zwaved_ctrl_api.h
|—   |— zware
|—   |—   |— ace_zware_cc_intf.c
|—   |—   |— ace_zware_common_utils.c
|—   |—   |— ace_zware_ctrl.c
|—   |—   |— ace_zware_debug.c
|—   |—   |— ace_zware_debug.h
|—   |—   |— ace_zware_internal.h
```

Als Ausgangspunkt für die Implementierung der DPK-Schicht für ein SDK eines anderen Anbieters kann die Referenzimplementierung verwendet und geändert werden. Die folgenden zwei Änderungen sind erforderlich, um ein SDK eines anderen Anbieters zu unterstützen:

1. Ersetzen Sie das aktuelle Hersteller-SDK durch das neue Hersteller-SDK im Repository.
2. Implementieren Sie das Middleware-DPK (Device Porting Kit) APIs gemäß dem SDK des neuen Anbieters.

Verwaltete Integrationen Endgeräte-SDK

Entwickeln Sie eine IoT-Plattform, die intelligente Geräte mit verwalteten Integrationen verbindet und Befehle über eine einheitliche Steuerschnittstelle verarbeitet. Das Endgeräte-SDK lässt sich in Ihre Gerätefirmware integrieren und bietet eine vereinfachte Einrichtung mit den SDK-Edge-Komponenten sowie eine sichere Konnektivität AWS IoT Core und AWS IoT Geräteverwaltung. Laden Sie die neueste Version des Endgeräte-SDK von der AWS-Managementkonsole

In diesem Handbuch wird beschrieben, wie Sie das Endgeräte-SDK in Ihre Firmware implementieren. Lesen Sie sich die Architektur, die Komponenten und die Integrationsschritte durch, um mit dem Aufbau Ihrer Implementierung zu beginnen.

Themen

Was ist das Endgeräte-SDK?

Was ist das Endgeräte-SDK?

Das Endgeräte-SDK ist eine Sammlung von Quellcode, Bibliotheken und Tools, die von bereitgestellt werden AWS IoT. Das SDK wurde für Umgebungen mit begrenzten Ressourcen entwickelt und unterstützt Geräte mit nur 512 KB RAM und 4 MB Flash-Speicher, z. B. Kameras und Luftreiniger, die auf eingebettetem Linux und Echtzeitbetriebssystemen (RTOS) laufen. Laden Sie die neueste Version des Endgeräte-SDK von der Management Console herunter.AWS IoT

Kernkomponenten

Das SDK kombiniert einen MQTT-Agenten für die Cloud-Kommunikation, einen Job-Handler für die Aufgabenverwaltung und einen verwalteten Integrationen, den Data Model Handler. Diese

Komponenten arbeiten zusammen, um sichere Konnektivität und automatisierte Datenübersetzung zwischen Ihren Geräten und verwalteten Integrationen zu gewährleisten.

Detaillierte technische Anforderungen finden Sie in derTechnische Referenz.

Architektur und Komponenten des SDK für Endgeräte

In diesem Abschnitt werden die SDK-Architektur für Endgeräte und die Interaktion ihrer Komponenten mit Ihren C-Funktionen auf niedriger Ebene beschrieben. Das folgende Diagramm veranschaulicht die Kernkomponenten und ihre Beziehungen im SDK-Framework.

SDK-Komponenten für Endgeräte

Die SDK-Architektur für Endgeräte enthält die folgenden Komponenten für die Funktionsintegration verwalteter Integrationen:

Bereitsteller

Erstellt Geräteressourcen in der Managed Integrations Cloud, einschließlich Gerätezertifikaten und privaten Schlüsseln für sichere MQTT-Kommunikation. Diese Anmeldeinformationen stellen vertrauenswürdige Verbindungen zwischen Ihrem Gerät und verwalteten Integrationen her.

MQTT-Agent

Verwaltet MQTT-Verbindungen über eine threadsichere C-Client-Bibliothek. Dieser Hintergrundprozess verarbeitet Befehlswarteschlangen in Multithread-Umgebungen mit konfigurierbaren Warteschlangengrößen für Geräte mit beschränktem Speicherplatz. Nachrichten werden zur Verarbeitung über verwaltete Integrationen weitergeleitet.

Handler für Jobs

Verarbeitet over-the-air (OTA) -Updates für Gerätefirmware, Sicherheitspatches und Dateibereitstellung. Dieser integrierte Dienst verwaltet Softwareupdates für alle registrierten Geräte.

Datenmodell-Handler

Übersetzt Operationen zwischen verwalteten Integrationen und Ihren Low-Level-C-Funktionen mithilfe der Implementierung AWS des Matter Data Model. Weitere Informationen finden Sie in der Matter-Dokumentation unter. GitHub

Schlüssel und Zertifikate

Verwaltet kryptografische Operationen über die PKCS #11 -API und unterstützt sowohl Hardware-Sicherheitsmodule als auch Softwareimplementierungen wie Core. PKCS11 Diese API verarbeitet Zertifikatsoperationen für Komponenten wie Provisionee und MQTT Agent bei TLS-Verbindungen.

Bereitsteller

Der Bereitsteller ist ein Bestandteil verwalteter Integrationen, die die Bereitstellung von Flotten nach Schadensfall ermöglichen. Mit dem Bereitsteller stellen Sie Ihre Geräte sicher bereit. Das SDK erstellt die erforderlichen Ressourcen für die Gerätebereitstellung, einschließlich des Gerätezertifikats und der privaten Schlüssel, die aus der Managed Integrations Cloud abgerufen werden. Wenn Sie Ihre Geräte bereitstellen möchten oder wenn es Änderungen gibt, die eine erneute Bereitstellung Ihrer Geräte erforderlich machen können, können Sie den Bereitsteller verwenden.

Themen

Arbeitsablauf für den Provisionsnehmer

Der Prozess muss sowohl auf der Cloud- als auch auf der Geräteseite eingerichtet werden. Kunden konfigurieren Cloud-Anforderungen wie benutzerdefinierte Endpunkte, Bereitstellungsprofile und verwaltete Dinge. Beim ersten Einschalten des Geräts hat der Bereitsteller:

1. Stellt mithilfe eines Antragszertifikats eine Verbindung zum Endpunkt der verwalteten Integrationen her
2. Überprüft Geräteparameter mithilfe von Fleet Provisioning-Hooks
3. Ruft ein permanentes Zertifikat und einen privaten Schlüssel ab und speichert es auf dem Gerät

4. Das Gerät verwendet das permanente Zertifikat, um die Verbindung wiederherzustellen
5. Erkennt Gerätefunktionen und lädt sie in verwaltete Integrationen hoch

Nach erfolgreicher Bereitstellung kommuniziert das Gerät direkt mit verwalteten Integrationen. Der Bereitsteller wird nur für Aufgaben zur erneuten Bereitstellung aktiviert.

Festlegen von Umgebungsvariablen

Stellen Sie die folgenden AWS Anmeldeinformationen in Ihrer Cloud-Umgebung ein:

```
$ 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
```

Registrieren Sie einen benutzerdefinierten Endpunkt

Verwenden Sie den RegisterCustomEndpointAPI-Befehl in Ihrer Cloud-Umgebung, um einen benutzerdefinierten Endpunkt für die device-to-cloud Kommunikation zu erstellen.

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

Beispielantwort

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

> **Note**
>
> Speichern Sie die Endpunktadresse für die Konfiguration der Bereitstellungsparameter. Verwenden Sie die `GetCustomEndpoint` API, um Endpunktinformationen zurückzugeben. Weitere Informationen finden Sie unter GetCustomEndpointAPI und API im RegisterCustomEndpointAPI-Referenzhandbuch für verwaltete Integrationen.

Erstellen Sie ein Provisioning-Profil

Erstellen Sie ein Bereitstellungsprofil, das Ihre Flottenbereitstellungsmethode definiert. Führen Sie die CreateProvisioningProfileAPI in Ihrer Cloud-Umgebung aus, um ein Antragszertifikat und einen privaten Schlüssel für die Geräteauthentifizierung zurückzugeben:

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

Beispielantwort

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

Sie können die Core PKCS11 Platform Abstraction Library (PAL) implementieren, damit die PKCS11 Kernbibliothek mit Ihrem Gerät funktioniert. Die wichtigsten PKCS11 PAL-Ports müssen einen Speicherort für das Anspruchszertifikat und den privaten Schlüssel bereitstellen. Mit dieser Funktion können Sie den privaten Schlüssel und das Zertifikat des Geräts sicher speichern. Sie können den privaten Schlüssel und das Zertifikat auf einem Hardware-Sicherheitsmodul (HSM) oder einem Trusted Platform Module (TPM) speichern.

Erstellen Sie eine verwaltete Sache

Registrieren Sie Ihr Gerät mithilfe der CreateManagedThingAPI bei der Managed Integrations Cloud. Geben Sie die Seriennummer (SN) und den Universal Product Code (UPC) Ihres Geräts an:

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

Im Folgenden wird ein Beispiel für eine API-Antwort angezeigt.

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

Die API gibt die verwaltete Thing-ID zurück, die für die Überprüfung der Bereitstellung verwendet werden kann. Sie müssen die Geräteseriennummer (SN) und den Universal Product Code (UPC) angeben, die während der Bereitstellungstransaktion mit dem genehmigten verwalteten Objekt abgeglichen werden. Die Transaktion gibt ein Ergebnis zurück, das dem folgenden ähnelt:

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

Wi-Fi-Bereitstellung für SDK-Benutzer

Gerätehersteller und Lösungsanbieter verfügen über ihren eigenen proprietären Wi-Fi-Bereitstellungsdienst für den Empfang und die Konfiguration von Wi-Fi-Anmeldeinformationen. Der Wi-Fi-Bereitstellungsdienst umfasst die Verwendung spezieller mobiler Apps, Bluetooth Low Energy (BLE) -Verbindungen und anderer proprietärer Protokolle, um Wi-Fi-Anmeldeinformationen für den ersten Einrichtungsprozess sicher zu übertragen.

Der Nutzer des Endgeräte-SDK muss den Wi-Fi-Bereitstellungsdienst implementieren und das Gerät kann eine Verbindung zu einem Wi-Fi-Netzwerk herstellen.

Bereitstellung der Flotte nach Schadensfall

Mithilfe des Anbieters kann der Endbenutzer ein einzigartiges Zertifikat bereitstellen und es mithilfe der Bereitstellung nach Anspruch bei verwalteten Integrationen registrieren.

Die Client-ID kann entweder aus der Antwort der Bereitstellungsvorlage oder aus dem Gerätezertifikat abgerufen werden `<common name>“_”<serial number>`

Kapazitäten verwalteter Objekte

Der Bereitsteller erkennt die Funktionen der verwalteten Dinge und lädt die Funktionen dann in verwaltete Integrationen hoch. Es stellt Apps und anderen Diensten Zugriff auf die Funktionen zur Verfügung. Geräte, andere Webclients und Dienste können die Funktionen mithilfe von MQTT und dem reservierten MQTT-Thema oder HTTP mithilfe der REST-API aktualisieren.

Over-the-Air Aktualisierungen

Überblick über die OTA-Architektur

Beim Over-the-Air (OTA-) Aktualisierungsprozess arbeiten mehrere Komponenten zusammen, um Firmware-Updates für Ihre Geräte bereitzustellen. Das folgende Diagramm zeigt, wie eine OTA-Aktualisierungsanforderung durch die Interaktion zwischen dem Endgeräte-SDK, dem Hub-SDK und der Funktion behandelt wird.

Die OTA-Aktualisierungsarchitektur besteht aus den folgenden Komponenten:

- Kunde: Lädt Jobdokumente in einen S3-Bucket hoch und initiiert Updates über die API
- OTA-Service: Kümmert sich um die Erstellung, Validierung und Verwaltung von Jobs
- AWS IoT Jobs: Verwaltet die Ausführung und Lieferung von Jobs an Geräte
- Geräte: Empfangen und Anwenden von Updates mithilfe des Harmony SDK

Voraussetzungen

Bevor Sie OTA-Aufgaben erstellen, müssen Sie die folgenden Voraussetzungen konfigurieren:

Amazon S3 S3-Zugriff konfigurieren

Um OTA-Updates zu aktivieren, müssen Sie Ihre Jobdokumente in einen Amazon S3 S3-Bucket hochladen und die entsprechenden Zugriffsberechtigungen konfigurieren:

1. Laden Sie Ihr OTA-Jobdokument in einen S3-Bucket hoch
2. Fügen Sie eine Amazon S3 S3-Bucket-Richtlinie hinzu, die verwalteten Integrationen Zugriff auf Ihre Jobdokumente gewährt:

JSON

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

```
      "Sid": "PolicyForS3JobDocument",
      "Effect": "Allow",
      "Principal": {
        "Service": "iotmanagedintegrations.amazonaws.com"
      },
      "Action": "s3:GetObject",
      "Resource": [
        "arn:aws:s3:::YOUR_BUCKET/*",
        "arn:aws:s3:::YOUR_BUCKET/ota_job_document.json",
        "arn:aws:s3:::YOUR_BUCKET"
      ]
    }
  ]
}
```

Implementieren Sie Over-the-Air Aufgaben (OTA)

Sie können OTA-Aufgaben auf zwei Arten erstellen, abhängig von Ihren Aktualisierungsanforderungen und Ihrer Strategie für das Geräte-Targeting:

Einmalige Aktualisierungen von OTA-Aufgaben

Eine einmalige OTA-Aufgabe enthält eine statische Liste von Zielen (`ManagedThings`) zur Durchführung von OTA-Updates. Sie können bis zu 100 Ziele gleichzeitig hinzufügen. Der Workflow verwendet AWS IoT Jobs with Fleet Indexing, wobei die Abstraktionsebene für verwaltete Integrationen beibehalten wird.

Verwenden Sie das folgende Beispiel, um eine einmalige OTA-Aufgabe zu erstellen:

```
aws iotmanagedintegrations create-ota-task \
  --description "One-time OTA update" \
  --s3-url "s3://test-job-document-bucket/ota-job-document.json" \
  --protocol HTTP \
  --target ["arn:aws:iotmanagedintegrations:region:account id:managed-thing/managed
 thing id"] \
  --ota-mechanism PUSH \
  --ota-type ONE_TIME \
  --client-token "foo" \
  --tags '{"key1":"foo","key2":"foo"}'
```

Kontinuierliche Aktualisierungen von OTA-Aufgaben

Der OTA-Gruppierungs-Workflow (Over-the-Air) ermöglicht es Ihnen, Firmware-Updates für Gerätegruppen auf der Grundlage bestimmter Attribute bereitzustellen, indem Sie AWS IoT Jobs with Fleet Indexing verwenden und dabei die Abstraktionsebene für verwaltete Integrationen beibehalten. Kontinuierliche OTA-Aufgaben verwenden eine Abfragezeichenfolge anstelle bestimmter Ziele. Alle Geräte, die den Abfragekriterien entsprechen, werden OTA-Updates unterzogen, und die Abfragekriterien werden ständig neu bewertet. Für die entsprechenden Ziele werden Aufträge bereitgestellt.

Konfigurieren Sie die Voraussetzungen

Bevor Sie fortlaufende OTA-Aufgaben erstellen, müssen Sie die folgenden Voraussetzungen erfüllen:

1. Erstellen Sie ein verwaltetes Objekt, indem Sie die CreateManagedThingAPI aufrufen, und führen Sie die Flottenbereitstellung durch.
2. Fügen Sie Metadatenattribute zu Ihren verwalteten Objekten hinzu, um Abfragen auszurichten.

Fügen Sie Attribute und Metadaten zur `ManagedThing` Verwendung der UpdateManagedThingAPI hinzu:

```
aws iotmanagedintegrations update-managed-thing \
  --managed-thing-id "YOUR_MANAGED_THING_ID" \
  --meta-data '{"owner":"managedintegrations","version":"1.0"}'
```

Verwenden Sie das folgende Beispiel, um eine kontinuierliche OTA-Aufgabe zu erstellen:

```
aws iotmanagedintegrations create-ota-task \
  --description "Continuous OTA update" \
  --s3-url "s3://test-job-document-bucket/ota-job-document.json" \
  --protocol HTTP \
  --ota-mechanism PUSH \
  --ota-type CONTINUOUS \
  --client-token "foo" \
  --ota-target-query-string "attributes.owner=managedintegrations" \
  --tags '{"key1":"foo","key2":"foo"}'
```

Verstehen Sie den kontinuierlichen OTA-Workflow

Der kontinuierliche OTA-Update-Workflow folgt diesen Schritten:

1. Mithilfe der UpdateManagedThingAPI aktualisieren Sie verwaltete Dinge mit Attributen.
2. Erstellen Sie einen OTA-Job mit einer Abfragezeichenfolge, die auf bestimmte Geräteattribute abzielt.
3. Der OTA-Dienst erstellt eine dynamische Dinggruppe auf der AWS IoT Core Grundlage von Abfrageattributen
4. IoT Jobs führt Updates auf passenden Geräten aus
5. Sie überwachen den Fortschritt über die ListOtaTaskExecutionsAPI oder OTA-Benachrichtigungen über den Kinesis-Stream (falls aktiviert).

Unterschiede zwischen Managed Integrations OTA und IoT Jobs

Der grundlegende Unterschied zwischen Managed Integrations OTA und IoT Jobs liegt in der Service-Orchestrierung und Automatisierung. Managed Integrations OTA bietet eine Single-Service-Lösung, die die Komplexität der Koordination mehrerer Dienste abstrahiert.

Was Managed Integrations OTA automatisch macht:

- Dynamische Dinggruppenerstellung: Generiert automatisch AWS IoT Core Dinggruppen auf der Grundlage Ihrer Abfragekriterien.
- Zielauflösung: Übersetzt Abfragezeichenfolgen (Beispiel:`attributes.owner=managedintegrations`) in tatsächliche Geräteziele.
- Serviceintegration: Koordiniert nahtlos zwischen AWS IoT Core IoT Jobs und Fleet Indexing Services.
- Lebenszyklusmanagement: Verwaltet den gesamten OTA-Workflow von der Erstellung bis zur Überwachung der Ausführung.

Was MI OTA eliminiert:

- Erstellen von Dinggruppen in AWS IoT Core.
- Dinge zu Gruppen hinzufügen.
- IoT-Arbeitsplätze schaffen.

Verwaltete Integrationen OTA wickelt alle drei Operationen intern auf der Grundlage Ihrer Abfragezeichenfolge ab, erkennt automatisch Geräte, die Ihren Kriterien entsprechen, erstellt IoT-

Jobs unter der Haube und orchestriert den gesamten OTA-Workflow, ohne dass Sie direkt mit mehreren AWS Diensten interagieren müssen.

Einrichtung von OTA-Aufgabenkonfigurationen

Sie können Konfigurationen für OTA-Updates erstellen, um zu steuern, wie Updates auf Geräten bereitgestellt werden, Abbruchbedingungen festzulegen und Timeouts zu konfigurieren.

Beispiel: CreateOtaTaskConfiguration

Verwenden Sie das folgende Beispiel, um eine OTA-Aufgabenkonfiguration zu erstellen:

```
aws iotmanagedintegrations create-ota-task-configuration \
  --description "OTA configuration" \
  --name "MyOtaConfig" \
  --push-config '{
    "AbortConfig": {
      "AbortConfigCriteriaList": [
        {
          "Action": "CANCEL",
          "FailureType": "FAILED",
          "MinNumberOfExecutedThings": 1,
          "ThresholdPercentage": 90.0
        }
      ]
    },
    "RolloutConfig": {
      "ExponentialRolloutRate": {
        "BaseRatePerMinute": 1,
        "IncrementFactor": 3.0,
        "RateIncreaseCriteria": {
          "numberOfNotifiedThings": 1
        }
      },
      "MaximumPerMinute": 1
    },
    "TimeoutConfig": {
      "InProgressTimeoutInMinutes": 100
    }
  }' \
  --client-token "foo"
```

Wenden Sie Konfigurationseinstellungen auf OTA-Aufgaben an

Sobald die Konfiguration erstellt ist, erhalten Sie eine`taskConfigurationId`, die Ihrer `CreateOtaTask` Anfrage zusammen mit weiteren Konfigurationen hinzugefügt wird:

```
aws iotmanagedintegrations create-ota-task \
  --description "OTA with configuration" \
  --s3-url "s3://test-job-document-bucket/ota-job-document.json" \
  --protocol HTTP \
  --target ["arn:aws:iotmanagedintegrations:region:account id:managed-thing/managed
 thing id"] \
  --ota-mechanism PUSH \
  --ota-type ONE_TIME \
  --client-token "foo" \
  --task-configuration-id "ae4f49352c5443369f43ad6c3a7f1580" \
  --ota-scheduling-config '{
    "EndBehavior": "STOP_ROLLOUT",
    "EndTime": "2024-10-23T17:00",
    "StartTime": "2024-10-20T17:00"
  }' \
  --ota-task-execution-retry-config '{
    "RetryConfigCriteria": [
      {
        "FailureType": "FAILED",
        "MinNumberOfRetries": 1
      }
    ]
  }' \
  --tags '{"key1":"foo","key2":"foo"}'
```

Überwachen Sie OTA-Benachrichtigungen

Sie können OTA-Updates mit zwei verschiedenen Methoden überwachen:

Push-Benachrichtigungen über Kinesis Data Streams

Wenn OTA-Benachrichtigungen aktiviert sind, werden Ereignisse zum Aktualisierungsstatus automatisch in Ihren Kinesis-Stream übertragen. Auf diese Weise erhalten Sie in Echtzeit einen Überblick über den Fortschritt der Firmware-Updates auf allen Geräten.

Überwachen Sie mit ListOtaTaskExecutions API

Sie können die ListOtaTaskExecutionsAPI verwenden, um den Status der OTA-Updates für Ihre verwalteten Dinge manuell zu überprüfen:

```
aws iotmanagedintegrations list-ota-task-executions \
  --task-id "task-123456789" \
  --max-results 25
```

Die Antwort enthält einen detaillierten Ausführungsstatus für jedes verwaltete Ding:

```
{
  "taskExecutionSummaries": [
    {
      "taskExecutionSummary": {
        "executionNumber": 1,
        "lastUpdatedAt": 1634567890,
        "queuedAt": 1634567800,
        "startedAt": 1634567830,
        "status": "SUCCEEDED",
        "retryAttempt": 0
      },
      "managedThingId": "device-001"
    },
    {
      "taskExecutionSummary": {
        "executionNumber": 1,
        "lastUpdatedAt": 1634567920,
        "queuedAt": 1634567800,
        "startedAt": 1634567840,
        "status": "IN_PROGRESS",
        "retryAttempt": 0
      },
      "managedThingId": "device-002"
    }
  ],
  "nextToken": "NEXT_TOKEN"
}
```

Mit dieser API können Sie den detaillierten Ausführungsstatus für jedes verwaltete Objekt abrufen, auf das eine bestimmte OTA-Aufgabe abzielt, einschließlich Zeitstempel und aktuellem Status.

Auftragsdokumente verarbeiten

Wenn Sie eine OTA-Aufgabe erstellen, führt der Job-Handler die folgenden Schritte auf Ihrem Gerät aus. Wenn ein Update verfügbar ist, fordert es das Jobdokument über MQTT an.

1. Abonniert die MQTT-Benachrichtigungsthemen.
2. Ruft die StartNextPendingJobExecutionAPI für ausstehende Jobs auf.
3. Empfängt verfügbare Auftragsdokumente.
4. Verarbeitet Aktualisierungen auf der Grundlage der von Ihnen angegebenen Timeouts.

Mithilfe des Job-Handlers kann die Anwendung bestimmen, ob sofort Maßnahmen ergriffen oder bis zu einem bestimmten Timeout-Zeitraum gewartet werden soll.

Implementieren Sie den OTA Agent

Wenn Sie das Jobdokument von Managed Integrations erhalten, benötigen Sie eine Implementierung Ihres eigenen OTA-Agenten, der das Jobdokument verarbeitet, Updates herunterlädt und alle Installationsvorgänge durchführt. Der OTA-Agent muss die folgenden Schritte ausführen:

1. Analysieren Sie Jobdokumente für die Firmware Amazon S3 URLs.
2. Laden Sie Firmware-Updates über HTTP herunter.
3. Überprüfen Sie die digitalen Signaturen.
4. Installieren Sie validierte Updates.
5. Rufen Sie `iotmi\_JobsHandler\_updateJobStatus` mit `SUCCESS` oder `FAILED` Status an.

Wenn Ihr Gerät den OTA-Vorgang erfolgreich abgeschlossen hat, muss es die `iotmi \_JobsHandler\_updateJobStatus` API mit dem Status „Einen `JobSucceeded` erfolgreichen Job melden“ aufrufen.

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

Codegenerator für Datenmodelle

Erfahren Sie, wie Sie den Codegenerator für das Datenmodell verwenden. Der generierte Code kann zur Serialisierung und Deserialisierung der Datenmodelle verwendet werden, die zwischen der Cloud und dem Gerät ausgetauscht werden.

Das Projekt-Repository enthält ein Tool zur Codegenerierung zum Erstellen von C-Code-Datenmodell-Handlern. In den folgenden Themen werden der Codegenerator und der Arbeitsablauf beschrieben.

Themen

- Prozess der Codegenerierung
- Einrichtung der Umgebung
- Generieren Sie Code für Geräte

Prozess der Codegenerierung

Der Codegenerator erstellt C-Quelldateien aus drei Haupteingaben: AWS'Implementierung des Matter-Datenmodells (.matter-Datei) von der Zigbee Cluster Library (ZCL) Advanced Platform, einem Python-Plugin, das die Vorverarbeitung übernimmt, und Jinja2-Vorlagen, die die Codestruktur definieren. Während der Generierung verarbeitet das Python-Plugin Ihre .matter-Dateien, indem es globale Typdefinitionen hinzufügt, Datentypen auf der Grundlage ihrer Abhängigkeiten organisiert und die Informationen für das Rendern von Vorlagen formatiert.

Die folgende Abbildung beschreibt den Codegenerator, der die C-Quelldateien erstellt.

Das Endgeräte-SDK enthält Python-Plugins und Jinja2-Vorlagen, mit denen Sie codegen.pyim connectedhomeipProjekt arbeiten können. Diese Kombination generiert mehrere C-Dateien für jeden Cluster auf der Grundlage Ihrer .matter-Dateieingabe.

In den folgenden Unterthemen werden diese Dateien beschrieben.

- Python-Plugin
- Jinja2-Vorlagen
- (Optional) Benutzerdefiniertes Schema

Python-Plugin

Der Codegenerator analysiert die .matter-Dateien und sendet die Informationen als Python-Objekte an das Plugin. `codegen.py` Die Plugin-Datei `iotmi_data_model.py` verarbeitet diese Daten vor und rendert Quellen mit den bereitgestellten Vorlagen. Die Vorverarbeitung beinhaltet:

1. Hinzufügen von Informationen, die nicht verfügbar sind`codegen.py`, wie z. B. globale Typen
2. Durchführen einer topologischen Sortierung von Datentypen, um die richtige Reihenfolge der Definitionen festzulegen

> **Note**
>
> Die topologische Sortierung stellt sicher, dass abhängige Typen unabhängig von ihrer ursprünglichen Reihenfolge nach ihren Abhängigkeiten definiert werden.

Jinja2-Vorlagen

Das Endgeräte-SDK bietet Jinja2-Vorlagen, die auf Datenmodell-Handler und C-Funktionen auf niedriger Ebene zugeschnitten sind.

Jinja2-Vorlagen

Vorlage	Generierte Quelle	Anmerkungen
`cluster.h.jinja`	`iotmi_device_<cluster>.h`	Erzeugt Header-Dateien für C-Funktionen auf niedriger Ebene.
`cluster.c.jinja`	`iotmi_device_<cluster>.c`	Implementieren und registrieren Sie Callback-Funktionszeiger mit dem Data Model Handler.
`cluster_type_helpers.h.jinja`	`iotmi_device_type_helpers_<cluster>.h`	Definiert Funktionsprototypen für Datentypen.
`cluster_type_helpers.c.jinja`	`iotmi_device_type_helpers_<cluster>.c`	Generiert Prototypen von Datentypfunktionen für clusterspezifische Aufzählungen, Bitmaps, Listen und Strukturen.
`iot_device_dm_types.h.jinja`	`iotmi_device_dm_types.h`	Definiert C-Datentypen für globale Datentypen.
`iot_device_type_helpers_global.h.jinja`	`iotmi_device_type_helpers_global.h`	Definiert C-Datentypen für globale Operationen.
`iot_device_type_helpers_global.c.jinja`	`iotmi_device_type_helpers_global.c`	Deklariert Standarddatentypen wie boolesche Werte, Ganzzahlen, Fließkommazahlen, Zeichenketten,

Vorlage	Generierte Quelle	Anmerkungen
		Bitmaps, Listen und Strukturen.

(Optional) Benutzerdefiniertes Schema

Das SDK für Endgeräte kombiniert den standardisierten Codegenerierungsprozess mit einem benutzerdefinierten Schema. Dies ermöglicht die Erweiterung von Matter Data Model für Ihre Geräte und Gerätesoftware. Benutzerdefinierte Schemas können dabei helfen, die device-to-cloud Kommunikationsmöglichkeiten von Geräten zu beschreiben.

Ausführliche Informationen zu Datenmodellen für verwaltete Integrationen, einschließlich Format, Struktur und Anforderungen, finden Sie unter. Datenmodell für verwaltete Integrationen

Verwenden Sie `codegen.py` das Tool, um C-Quelldateien für ein benutzerdefiniertes Schema wie folgt zu generieren:

> **Note**
>
> Jeder benutzerdefinierte Cluster benötigt dieselbe Cluster-ID für die folgenden drei Dateien.

- Erstellen Sie ein benutzerdefiniertes Schema in einem `JSON` Format, das eine Darstellung von Clustern für die Erstellung neuer benutzerdefinierter Cluster in der Cloud zur Verfügung stellt. Eine Beispieldatei befindet sich unter`codegen/custom_schemas/custom.SimpleLighting@1.0`.
- Erstellen Sie eine ZCL-Definitionsdatei (Zigbee Cluster Library) in einem `XML` Format, das dieselben Informationen wie das benutzerdefinierte Schema enthält. Verwenden Sie das ZAP-Tool, um Ihre Matter-IDL-Dateien aus ZCL-XML zu generieren. Eine Beispieldatei befindet sich unter. `codegen/zcl/custom.SimpleLighting.xml`
- Die Ausgabe des ZAP-Tool ist `Matter IDL File (.matter)` und definiert die Matter-Cluster, die Ihrem benutzerdefinierten Schema entsprechen. Dies ist die Eingabe für das `codegen.py` Tool zum Generieren von C-Quelldateien für das Endgeräte-SDK. Eine Beispieldatei befindet sich unter`codegen/matter_files/custom-light.matter`.

Ausführliche Anweisungen zur Integration von Datenmodellen für benutzerdefinierte verwaltete Integrationen in Ihren Workflow zur Codegenerierung finden Sie unterGenerieren Sie Code für Geräte.

Einrichtung der Umgebung

Erfahren Sie, wie Sie Ihre Umgebung für die Verwendung des `codegen.py` Codegenerators konfigurieren.

Themen

Voraussetzungen

Installieren Sie die folgenden Elemente, bevor Sie Ihre Umgebung konfigurieren:

- Git
- Python 3.10 oder höher
- Poesie 1.2.0 oder höher

Konfiguriere deine Umgebung

Gehen Sie wie folgt vor, um Ihre Umgebung für die Verwendung des Codegenerators codegen.py zu konfigurieren.

1. Laden Sie die neueste Version des Endgeräte-SDK von der herunter AWS-Managementkonsole.
2. Richten Sie die Python-Umgebung ein. Das Codegen-Projekt basiert auf Python und verwendet Poetry für das Abhängigkeitsmanagement.
    - Installieren Sie Projektabhängigkeiten mithilfe von Poetry im Verzeichnis: `codegen`

    ```
    poetry run poetry install --no-root
    ```
3. Richten Sie Ihr Repository ein.

a. Klonen Sie das connectedhomeipRepository. Es verwendet das `codegen.py` Skript, das sich im `connectedhomeip/scripts/` Ordner befindet, für die Codegenerierung. Weitere Informationen finden Sie unter connectedhomeip on. GitHub

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

b. Klonen Sie es auf derselben Ebene wie Ihren Stammordner. `IoT-managed-integrations-End-Device-SDK` Ihre Ordnerstruktur sollte wie folgt aussehen:

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

> **Note**
>
> Sie müssen Submodule nicht rekursiv klonen.

Generieren Sie Code für Geräte

Erstellen Sie mithilfe der Tools zur Codegenerierung für verwaltete Integrationen benutzerdefinierten C-Code für Ihre Geräte. In diesem Abschnitt wird beschrieben, wie Sie Code aus Beispieldateien, die im SDK enthalten sind, oder anhand Ihrer eigenen Spezifikationen generieren. Erfahren Sie, wie Sie die Generierungsskripten verwenden, den Workflow-Prozess verstehen und Code erstellen, der Ihren Geräteanforderungen entspricht.

Themen

Voraussetzungen

1. Python 3.10 oder höher.

2. Beginnen Sie mit einer .matter-Datei für die Codegenerierung. Das Endgeräte-SDK enthält zwei Beispieldateien im`codgen/matter_files folder`:

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

> **Note**
>
> Diese Beispieldateien generieren Code für Demo-Anwendungscluster.

Code generieren

Führen Sie diesen Befehl aus, um Code im Out-Ordner zu generieren:

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

Generieren Sie Code für benutzerdefinierte .matter-Dateien

Führen Sie die folgenden Aufgaben aus, um den Code für eine bestimmte `.matter` `.matter` Datei zu generieren oder eine eigene Datei bereitzustellen.

Um den Code für benutzerdefinierte .matter-Dateien zu generieren

1. Bereiten Sie Ihre .matter-Datei vor
2. Führen Sie den Generierungsbefehl aus:

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

(Optional) Um Code mit einem benutzerdefinierten Schema zu generieren

1. Bereiten Sie Ihr benutzerdefiniertes Schema im `JSON` Format vor
2. Führen Sie den Generierungsbefehl aus:

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

Die obigen Befehle verwenden mehrere Komponenten, um Ihre `.matter` Datei in `C` Code umzuwandeln:

- `codegen.py`aus dem ConnectedHomeIP-Projekt
- Python-Plugin befindet sich unter `codegen/py_scripts/iotmi_data_model.py`
- Jinja2-Vorlagen aus dem Ordner `codegen/py_scripts/templates`

Das Plugin definiert die Variablen, die an die Jinja2-Vorlagen übergeben werden, die dann verwendet werden, um die endgültige C-Code-Ausgabe zu generieren. Durch Hinzufügen des `--format` Flags wird das Clang-Format auf den generierten Code angewendet.

Arbeitsablauf bei der Codegenerierung

Bei der Codegenerierung werden die Datenstrukturen Ihrer .matter-Datei mithilfe von Hilfsfunktionen und topologischer Sortierung organisiert. `topsort.py` Dadurch wird die korrekte Reihenfolge der Datentypen und ihrer Abhängigkeiten gewährleistet.

Das Skript kombiniert dann Ihre .matter-Dateispezifikationen mit der Python-Plugin-Verarbeitung, um die erforderlichen Informationen zu extrahieren und zu formatieren. Schließlich wendet es die Jinja2-Vorlagenformatierung an, um die endgültige C-Code-Ausgabe zu erstellen.

Dieser Workflow stellt sicher, dass Ihre gerätespezifischen Anforderungen aus der .matter-Datei korrekt in funktionalen C-Code übersetzt werden, der sich in das verwaltete Integrationssystem integrieren lässt.

C-Funktion auf niedriger Ebene APIs

Integrieren Sie Ihren gerätespezifischen Code mithilfe der mitgelieferten Low-Level-C-Funktion in verwaltete Integrationen. APIs In diesem Abschnitt werden die API-Operationen beschrieben, die für jeden Cluster im AWS Datenmodell verfügbar sind, um effiziente Interaktionen zwischen Geräten und der Cloud zu gewährleisten. Erfahren Sie, wie Sie Rückruffunktionen implementieren, Ereignisse ausgeben, Attributänderungen benachrichtigen und Cluster für Ihre Geräteendpunkte registrieren.

Zu den wichtigsten API-Komponenten gehören:

1. Strukturen von Zeigern für Callback-Funktionen für Attribute und Befehle
2. Funktionen zur Emission von Ereignissen
3. Funktionen zur Benachrichtigung über Attributänderungen
4. Funktionen zur Cluster-Registrierung

Durch deren APIs Implementierung schaffen Sie eine Brücke zwischen dem physischen Betrieb Ihres Geräts und den Cloud-Funktionen für verwaltete Integrationen und sorgen so für eine reibungslose Kommunikation und Steuerung.

Im folgenden Abschnitt wird die OnOffCluster-API veranschaulicht.

OnOff Cluster-API

Der OnOff.xmlCluster unterstützt diese Attribute und Befehle:.

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

  Für jeden Befehl stellen wir den 1:1 gemappten Funktionszeiger zur Verfügung, mit dem Sie Ihre Implementierung verknüpfen können.

Alle Callbacks für Attribute und Befehle sind in einer C-Struktur definiert, die nach dem Cluster benannt ist.

Beispiel für eine C-Struktur

```
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);
};
```

Zusätzlich zur C-Struktur sind Funktionen zur Meldung von Attributänderungen für alle Attribute definiert.

```
/* 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);
```

Funktionen zur Ereignisberichterstattung sind für alle clusterspezifischen Ereignisse definiert. Da der OnOff Cluster keine Ereignisse definiert, finden Sie im Folgenden ein Beispiel aus dem `CameraAvStreamManagement` Cluster.

```
/* 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);
```

Jeder Cluster hat auch eine Registerfunktion.

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

Der an die Registerfunktion übergebene Benutzerzeiger wird an die Callback-Funktionen übergeben.

Funktions- und Geräteinteraktionen in verwalteten Integrationen

In diesem Abschnitt werden die Rolle der C-Function-Implementierung und die Interaktion zwischen dem Gerät und der Gerätefunktion für verwaltete Integrationen beschrieben.

Themen

- Umgang mit Fernbefehlen
- Umgang mit unaufgeforderten Ereignissen

Umgang mit Fernbefehlen

Fernbefehle werden durch die Interaktion zwischen dem Endgeräte-SDK und der Funktion verarbeitet. Die folgenden Aktionen beschreiben ein Beispiel dafür, wie Sie mithilfe dieser Interaktion eine Glühbirne einschalten können.

Der MQTT-Client empfängt Nutzdaten und leitet sie an Data Model Handler weiter

Wenn Sie einen Remote-Befehl senden, empfängt der MQTT-Client die Nachricht von verwalteten Integrationen im JSON-Format. Anschließend übergibt er die Nutzlast an den Datenmodell-

Handler. Angenommen, Sie möchten verwaltete Integrationen verwenden, um eine Glühbirne einzuschalten. Die Glühbirne hat einen Endpunkt #1, der den OnOff Cluster unterstützt. Wenn Sie in diesem Fall den Befehl zum Einschalten der Glühbirne senden, sendet Managed Integrations eine Anfrage über MQTT an das Gerät, die besagt, dass es den Befehl On auf dem Endpunkt #1 aufrufen möchte.

Data Model Handler sucht nach Callback-Funktionen und ruft sie auf

Der Data Model Handler analysiert die JSON-Anforderung. Wenn die Anfrage Eigenschaften oder Aktionen enthält, findet der Data Model Handler die Endpunkte und ruft sequentiell die entsprechenden Callback-Funktionen auf. Wenn der Data Model Handler beispielsweise die MQTT-Nachricht empfängt, prüft er, ob die Callback-Funktion, die dem im OnOff Cluster definierten On-Befehl entspricht, am Endpunkt #1 registriert ist.

Die Implementierung eines Handlers und einer C-Funktion führt den Befehl aus

Der Data Model Handler ruft die entsprechenden Callback-Funktionen auf, die er gefunden hat, und ruft sie auf. Die Implementierung der C-Funktion ruft dann die entsprechenden Hardwarefunktionen auf, um die physische Hardware zu steuern, und gibt das Ausführungsergebnis zurück. Im Fall der Glühbirne ruft der Data Model Handler beispielsweise die Callback-Funktion auf und speichert das Ausführungsergebnis. Die Callback-Funktion schaltet daraufhin die Glühbirne ein.

Data Model Handler gibt das Ausführungsergebnis zurück

Sobald alle Callback-Funktionen aufgerufen wurden, kombiniert der Data Model Handler alle Ergebnisse. Anschließend packt er die Antwort im JSON-Format und veröffentlicht das Ergebnis mithilfe des MQTT-Clients in der Managed Integrations Cloud. Im Fall der Glühbirne enthält die MQTT-Nachricht in der Antwort das Ergebnis, dass die Glühbirne durch die Callback-Funktion eingeschaltet wurde.

Umgang mit unaufgeforderten Ereignissen

Unerwünschte Ereignisse werden auch durch die Interaktion zwischen dem Endgeräte-SDK und der Funktion behandelt. Die folgenden Aktionen beschreiben, wie das geht.

Das Gerät sendet eine Benachrichtigung an Data Model Handler

Wenn eine Eigenschaftsänderung oder ein Ereignis eintritt, z. B. wenn eine physische Taste am Gerät gedrückt wurde, generiert die Implementierung der C-Funktion eine unaufgeforderte

Ereignisbenachrichtigung und ruft die entsprechende Benachrichtigungsfunktion auf, um die Benachrichtigung an den Data Model Handler zu senden.

Data Model Handler übersetzt die Benachrichtigung

Der Data Model Handler verarbeitet die empfangene Benachrichtigung und übersetzt sie in das AWS Datenmodell.

Data Model Handler veröffentlicht die Benachrichtigung in der Cloud

Der Data Model Handler veröffentlicht dann mithilfe des MQTT-Clients ein unaufgefordertes Ereignis in der Managed Integrations Cloud.

Beginnen Sie mit dem End Device SDK

Gehen Sie wie folgt vor, um das Endgeräte-SDK auf einem Linux-Gerät auszuführen. Dieser Abschnitt führt Sie durch die Einrichtung der Umgebung, die Netzwerkkonfiguration, die Implementierung der Hardwarefunktionen und die Endpunktkonfiguration.

> **Important**
>
> Die Demonstrationsanwendungen im `examples` Verzeichnis und ihre PAL-Implementierung (Platform Abstraction Layer) `platform/posix` dienen nur als Referenz. Verwenden Sie diese nicht in Produktionsumgebungen.

Überprüfen Sie jeden Schritt des folgenden Verfahrens sorgfältig, um eine ordnungsgemäße Geräteintegration mit verwalteten Integrationen sicherzustellen.

Integrieren Sie das Endgeräte-SDK

1. EC2 Amazon-Instanz einrichten

   Melden Sie sich bei der an AWS-Managementkonsole und starten Sie eine EC2 Amazon-Instance mit einem Amazon Linux AMI. Weitere Informationen finden Sie unter Erste Schritte mit Amazon EC2 im Amazon Elastic Container Registry-Benutzerhandbuch.
2. Richten Sie die Build-Umgebung ein

   Erstellen Sie den Code auf Amazon Linux 2023/x86_64 als Entwicklungshost. Installieren Sie die erforderlichen Build-Abhängigkeiten:

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

3. (Optional) Richten Sie das Netzwerk ein

   Das Endgeräte-SDK wird am besten mit physischer Hardware verwendet. Wenn Sie Amazon verwenden EC2, folgen Sie diesem Schritt nicht.

   Wenn Sie Amazon EC2 vor der Verwendung der Beispielanwendung nicht verwenden, initialisieren Sie das Netzwerk und verbinden Sie Ihr Gerät mit einem verfügbaren Wi-Fi-Netzwerk. Schließen Sie das Netzwerk-Setup ab, bevor Sie das Gerät bereitstellen:

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

4. Konfigurieren Sie die Bereitstellungsparameter

   > **Note**
   >
   > Folgen Sie Provisionee, um das Antragszertifikat und den privaten Schlüssel zu erhalten, bevor Sie fortfahren.

   Ändern Sie die Konfigurationsdatei `example/project_name/device_config.sh` mit den folgenden Bereitstellungsparametern:

   Bereitstellungsparameter

Makro-Parameter	Description	Wie erhalte ich diese Informationen
`IOTMI_ROOT_CA_PATH`	Die Root-CA-Zertifikatsdatei.	Sie können diese Datei im Abschnitt Amazon Root CA-Zertifikat herunterladen im AWS IoT Core Entwicklerhandbuch herunterladen.
`IOTMI_CLAIM_CERTIFICATE_PATH`	Der Pfad zur Datei mit dem Anspruchszertifikat.	Um das Anspruchszertifikat und den privaten Schlüssel zu erhalten, erstellen Sie mithilfe der CreateProvisioningProfileAPI ein Bereitstellungsprofil. Detaillierte Anweisung

Makro-Parameter	Description	Wie erhalte ich diese Informationen
`IOTMI_CLAIM_PRIVATE_KEY_PATH`	Der Pfad zur Datei mit dem privaten Claim-Schlüssel.	en finden Sie unter Erstellen Sie ein Provisioning-Profil.
`IOTMI_MANAGEDINTEGRATIONS_ENDPOINT`	Endpunkt-URL für verwaltete Integrationen.	Verwenden Sie die API, um den Endpunkt für verwaltete Integrationen abzurufen. RegisterCustomEndpoint Detaillierte Anweisungen finden Sie unter Registrieren Sie einen benutzerdefinierten Endpunkt.
IOTMI_MANAGEDINTEGRATIONS_ENDPOINT_PORT	Die Portnummer für den Endpunkt der verwalteten Integrationen	Standardmäßig wird der Port 8883 für MQTT-Veröffentlichungs- und Abonnementvorgänge verwendet. Port 443 ist für die TLS-Erweiterung Application Layer Protocol Negotiation (ALPN) festgelegt, die Geräte verwenden.

5. Erstellen Sie die Demo-Anwendungen und führen Sie sie aus

   In diesem Abschnitt werden zwei Linux-Demoanwendungen vorgestellt: eine einfache Sicherheitskamera und ein Luftreiniger, die beide CMake als Build-System verwendet werden.

   a. Einfache Sicherheitskamera-Anwendung

      Führen Sie die folgenden Befehle aus, um die Anwendung zu erstellen und auszuführen:

```
>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
```

In dieser Demo werden einfache C-Funktionen für eine simulierte Kamera mit RTC Session Controller und Recording Clustern implementiert. Schließen Sie den unter beschriebenen Ablauf ab, bevor Sie ihn ausführen. Arbeitsablauf für den Provisionsnehmer

Beispielausgabe der Demo-Anwendung:

```
[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_XXXXXXXXXXXXXXXXXXXXXXX
[2406840540][MAIN][INFO] Managed thing ID : XXXXXXXXXXXXXXXXXXXXXX
[2406840540][MAIN][INFO] ======================== application loop
 ==================
```

```
[2025-01-02 01:43:19.217094828][iotmi_device_sdkLog][INFO] [2406840540]
[MQTT_AGENT][INFO]
[2025-01-02 01:43:19.217124600][iotmi_device_sdkLog][INFO] Establishing a TLS
 session to XXXXXXXXX.account-prefix-ats.iot.region.amazonaws.com:8883
[2025-01-02 01:43:19.217138724][iotmi_device_sdkLog][INFO]
[2406840540][Cluster OnOff][INFO] exampleOnOffInitCluster() for endpoint#1
[2406840540][MAIN][INFO] Press Ctrl+C when you finish testing...
[2406840540][Cluster ActivatedCarbonFilterMonitoring][INFO]
 exampleActivatedCarbonFilterMonitoringInitCluster() for endpoint#1
[2406840540][Cluster AirQuality][INFO] exampleAirQualityInitCluster() for
 endpoint#1
[2406840540][Cluster CarbonDioxideConcentrationMeasurement][INFO]
 exampleCarbonDioxideConcentrationMeasurementInitCluster() for endpoint#1
[2406840540][Cluster FanControl][INFO] exampleFanControlInitCluster() for
 endpoint#1
[2406840540][Cluster HepaFilterMonitoring][INFO]
 exampleHepaFilterMonitoringInitCluster() for endpoint#1
[2406840540][Cluster Pm1ConcentrationMeasurement][INFO]
 examplePm1ConcentrationMeasurementInitCluster() for endpoint#1
[2406840540][Cluster Pm25ConcentrationMeasurement][INFO]
 examplePm25ConcentrationMeasurementInitCluster() for endpoint#1
[2406840540][Cluster TotalVolatileOrganicCompoundsConcentrationMeasurement]
[INFO]
 exampleTotalVolatileOrganicCompoundsConcentrationMeasurementInitCluster() for
 endpoint#1
[2025-01-02 01:43:19.648185488][iotmi_device_sdkLog][INFO] [2406840971]
[MQTT_AGENT][INFO]
[2025-01-02 01:43:19.648211988][iotmi_device_sdkLog][INFO] TLS session
 connected
[2025-01-02 01:43:19.648225583][iotmi_device_sdkLog][INFO]

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

b. Einfache Luftreiniger-Anwendung

Führen Sie die folgenden Befehle aus, um die Anwendung zu erstellen und auszuführen:

```
>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
```

In dieser Demo werden grundlegende C-Funktionen für einen simulierten Luftreiniger mit 2 Endpunkten und den folgenden unterstützten Clustern implementiert:

Unterstützte Cluster für den Luftreiniger-Endpunkt

Endpoint	Cluster
Endpunkt #1: Luftreiniger	OnOff
	Steuerung des Lüfters
	Überwachung von HEPA-Filtern
	Überwachung von Aktivkohlefiltern
Endpunkt #2: Luftqualitätssensor	Luftqualität
	Messung der Kohlendioxidkonzentration
	Messung der Formaldehydkonzentration
	Messung der Pm25-Konzentration
	Pm1-Konzentrationsmessung
	Messung der Gesamtkonzentration flüchtiger organischer Verbindungen

Die Ausgabe ähnelt der der Kamera-Demo-Anwendung, es werden jedoch verschiedene Cluster unterstützt.

6. Nächste Schritte:

Das Managed Integrations End Device SDK und die Demoanwendungen werden jetzt auf Ihren EC2 Amazon-Instances ausgeführt. Auf diese Weise können Sie Ihre Anwendungen auf

Ihrer eigenen physischen Hardware entwickeln und testen. Mit diesem Setup können Sie den Managed Integrations Service nutzen, um Ihre AWS IoT Geräte zu steuern.

a. Entwickeln Sie Hardware-Callback-Funktionen

Bevor Sie die Hardware-Callback-Funktionen implementieren, sollten Sie sich mit der Funktionsweise der API vertraut machen. In diesem Beispiel werden der On/Off Cluster und das OnOff Attribut verwendet, um eine Gerätefunktion zu steuern. Einzelheiten zur API finden Sie unterC-Funktion auf niedriger Ebene APIs.

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

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

b. Richten Sie Endpunkte ein und binden Sie Hardware-Callback-Funktionen ein

Nachdem Sie die Funktionen implementiert haben, erstellen Sie Endpunkte und registrieren Sie Ihre Callbacks. Führen Sie die folgenden Schritte aus:

i. Erstellen Sie einen Geräteagenten.

A. Erstellen Sie einen Geräteagenten, `iotmiDev_Agent_new()` bevor Sie andere SDK-Funktionen aufrufen.

B. Ihre Konfiguration muss mindestens die Parameter thingId und clientId enthalten.

C. Verwenden Sie die `iotmiDev_Agent_initDefaultConfig()` Funktion, um angemessene Standardwerte für Parameter wie Warteschlangengrößen und maximale Endpunkte festzulegen.

D. Wenn Sie die Ressourcen nicht mehr verwenden, geben Sie sie mit der `iotmiDev_Agent_free()` Funktion wieder frei. Dies verhindert Speicherlecks und gewährleistet eine ordnungsgemäße Ressourcenverwaltung in Ihrer Anwendung.

ii. Füllen Sie die Callback-Funktionszeiger für jede Clusterstruktur aus, die Sie unterstützen möchten.

iii. Richten Sie Endpunkte ein und registrieren Sie Ihre unterstützten Cluster.

Erstellen Sie Endgeräte mit`iotmiDev_Agent_addEndpoint()`, was Folgendes erfordert:

A. Eine eindeutige Endpunkt-ID.

B. Ein beschreibender Endpunktname

C. Ein oder mehrere Gerätetypen, die den AWS Datenmodelldefinitionen entsprechen.

D. Nachdem Sie Endpoints erstellt haben, registrieren Sie Cluster mithilfe der entsprechenden clusterspezifischen Registrierungsfunktionen.

E. Jede Clusterregistrierung erfordert Rückruffunktionen für Attribute und Befehle. Das System übergibt Ihren Benutzerkontextzeiger an Callbacks, um den Status zwischen Aufrufen aufrechtzuerhalten.

```
struct DeviceState
{
    struct iotmiDev_Agent * agent;
    struct iotmiDev_Endpoint *endpoint1;

    /* OnOff cluster states*/
    bool hwState;
};


/* This implementation for OnOff getter just reads
   the state from the DeviceState */
iotmiDev_DMStatus exampleGetOnOff( bool * value, void * user )
{
    struct DeviceState * state = ( struct DeviceState * ) ( user );
    *value = state->hwState;
    printf( "%s(): state->hwState: %d\n", __func__, state->hwState );
    return iotmiDev_DMStatusOk;
```

```
}

iotmiDev_DMStatus exampleGetOnTime( uint16_t * value, void * user )
{
    *value = 0;
    printf( "%s(): OnTime is %u\n", __func__, *value );
    return iotmiDev_DMStatusOk;
}

iotmiDev_DMStatus exampleGetStartUpOnOff( iotmiDev_OnOff_StartUpOnOffEnum *
 value, void * user )
{
    *value = iotmiDev_OnOff_StartUpOnOffEnum_Off;
    printf( "%s(): StartUpOnOff is %d\n", __func__, *value );
    return iotmiDev_DMStatusOk;
}

void setupOnOff( struct DeviceState *state )
{
    struct iotmiDev_clusterOnOff clusterOnOff = {
        .getOnOff = exampleGetOnOff,
        .getOnTime = exampleGetOnTime,
        .getStartUpOnOff = exampleGetStartUpOnOff,
    };
    iotmiDev_OnOffRegisterCluster( state->endpoint1,
                                   &clusterOnOff,
                                   ( void * ) state);
}


/* Here is the sample setting up an endpoint 1 with OnOff
   cluster. Note all error handling code is omitted. */
void setupAgent(struct DeviceState *state)
{
    struct iotmiDev_Agent_Config config = {
        .thingId = IOTMI_DEVICE_MANAGED_THING_ID,
        .clientId = IOTMI_DEVICE_CLIENT_ID,
    };
    iotmiDev_Agent_InitDefaultConfig(&config);

    /* Create a device agent before calling other SDK APIs */
    state->agent = iotmiDev_Agent_new(&config);

    /* Create endpoint#1 */
```

```
    state->endpoint1 = iotmiDev_Agent_addEndpoint( state->agent,
                                                   1,
                                                   "Data Model Handler Test
 Device",
                                                   (const char*[])
{ "Camera" },
                                                   1 );
    setupOnOff(state);
}
```

c. Verwenden Sie den Job-Handler, um das Job-Dokument abzurufen

i. Initiieren Sie einen Anruf an Ihre OTA-Anwendung:

```
static iotmi_JobCurrentStatus_t processOTA( iotmi_JobData_t * pJobData )
{
    iotmi_JobCurrentStatus_t jobCurrentStatus = JobSucceeded;

...
    // This function should create OTA tasks
    jobCurrentStatus = YOUR_OTA_FUNCTION(iotmi_JobData_t * pJobData);
...

    return jobCurrentStatus;
}
```

ii. Rufen Sie `iotmi_JobsHandler_start` auf, um den Job-Handler zu initialisieren.

iii. Rufen Sie `iotmi_JobsHandler_getJobDocument` auf, um das Jobdokument aus verwalteten Integrationen abzurufen.

iv. Wenn das Jobs-Dokument erfolgreich abgerufen wurde, schreiben Sie Ihre benutzerdefinierte OTA-Operation in die `processOTA` Funktion und geben Sie einen `JobSucceeded` Status zurück.

```
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." ) );

}
```

Portieren Sie das Endgeräte-SDK auf Ihr Gerät

Portieren Sie das Endgeräte-SDK auf Ihre Geräteplattform. Gehen Sie wie folgt vor, um Ihre Geräte mit AWS IoT Device Management zu verbinden.

Laden Sie das Endgeräte-SDK herunter und überprüfen Sie es

1. Laden Sie die neueste Version des Endgeräte-SDK von der Managed Integrations Console herunter.
2. Vergewissern Sie sich, dass Ihre Plattform in der Liste der unterstützten Plattformen unter aufgeführt ist. Referenz: Unterstützte Plattformen

> **Note**
>
> Das Endgeräte-SDK wurde auf den angegebenen Plattformen getestet. Andere Plattformen funktionieren möglicherweise, wurden aber nicht getestet.

3. Extrahieren (entpacken) Sie die SDK-Dateien in Ihren Workspace.
4. Konfigurieren Sie Ihre Build-Umgebung mit den folgenden Einstellungen:
   - Pfade der Quelldateien
   - Header-Dateiverzeichnisse
   - Benötigte Bibliotheken
   - Compiler- und Linker-Flags
5. Bevor Sie den Platform Abstraction Layer (PAL) portieren, stellen Sie sicher, dass die grundlegenden Funktionen Ihrer Plattform initialisiert sind. Zu den Funktionen gehören:
   - Aufgaben des Betriebssystems
   - Peripheriegeräte
   - Netzwerkschnittstellen
   - Plattformspezifische Anforderungen

Portieren Sie das PAL auf Ihr Gerät

1. Erstellen Sie ein neues Verzeichnis für Ihre plattformspezifischen Implementierungen im vorhandenen Plattformverzeichnis. Wenn Sie beispielsweise FreeRTOS verwenden, erstellen Sie ein Verzeichnis unter. `platform/freertos`

   Example SDK-Verzeichnisstruktur

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

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

2. Kopieren Sie die POSIX-Referenzimplementierungsdateien (.c und .h) aus dem POSIX-Ordner in Ihr neues Plattformverzeichnis. Diese Dateien bieten eine Vorlage für die Funktionen, die Sie implementieren müssen.
   - Flash-Speicherverwaltung für die Speicherung von Anmeldeinformationen
   - PKCS #11 -Implementierung
   - Netzwerk-Transportschnittstelle
   - Zeitsynchronisierung
   - Funktionen zum Neustarten und Zurücksetzen des Systems
   - Mechanismen der Protokollierung
   - Gerätespezifische Konfigurationen
3. Richten Sie die Transport Layer Security (TLS) -Authentifizierung mit MBed TLS ein.
   - Verwenden Sie die bereitgestellte POSIX-Implementierung, wenn Sie bereits über eine MBed TLS-Version verfügen, die der SDK-Version auf Ihrer Plattform entspricht.
   - Mit einer anderen TLS-Version implementieren Sie die Transport-Hooks für Ihren TLS-Stack mit TCP/IP Stack.
4. Vergleichen Sie die MbedTLS-Konfiguration Ihrer Plattform mit den SDK-Anforderungen unter. `platform/posix/mbedtls/mbedtls_config.h` Stellen Sie sicher, dass alle erforderlichen Optionen aktiviert sind.
5. Das SDK ist auf CoreMQTT angewiesen, um mit der Cloud zu interagieren. Daher müssen Sie eine Netzwerktransportschicht implementieren, die die folgende Struktur verwendet:

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

Weitere Informationen finden Sie in der Dokumentation zur Transportschnittstelle auf der FreeRTOS-Website.

6. (Optional) Das SDK verwendet die PCKS #11 -API, um Zertifikatsoperationen abzuwickeln. CorePKCS ist eine nicht hardwarespezifische PKCS #11 -Implementierung für das Prototyping. Wir empfehlen, in Ihrer Produktionsumgebung sichere Kryptoprozessoren wie Trusted Platform Module (TPM), Hardware Security Module (HSM) oder Secure Element zu verwenden:

   - Sehen Sie sich die PKCS #11 -Beispielimplementierung an, die das Linux-Dateisystem für die Verwaltung von Anmeldeinformationen verwendet, unter. `platform/posix/corePKCS11-mbedtls`
   - Implementieren Sie die PAL-Ebene PKCS #11 unter. `commonDependencies/core_pkcs11/corePKCS11/source/include/core_pkcs11.h`
   - Implementieren Sie das Linux-Dateisystem unter`platform/posix/corePKCS11-mbedtls/source/iotmi_pal_Pkcs11Operations.c`.
   - Implementieren Sie die Speicher- und Ladefunktion Ihres Speichertyps unter`platform/include/iotmi_pal_Nvm.h`.
   - Implementieren Sie den standardmäßigen Dateizugriff unter`platform/posix/source/iotmi_pal_Nvm.c`.

   Eine ausführliche Anleitung zur Portierung finden Sie unter Portierung der PKCS11 Kernbibliothek im FreeRTOS-Benutzerhandbuch.
7. Fügen Sie die statischen SDK-Bibliotheken zu Ihrer Build-Umgebung hinzu:

   - Richten Sie die Bibliothekspfade ein, um alle Linker-Probleme oder Symbolkonflikte zu lösen
   - Stellen Sie sicher, dass alle Abhängigkeiten ordnungsgemäß verknüpft sind

Testen Sie Ihren Port

Sie können die vorhandene Beispielanwendung verwenden, um Ihren Port zu testen. Die Kompilierung muss ohne Fehler oder Warnungen abgeschlossen werden.

> **Note**
>
> Wir empfehlen, mit einer möglichst einfachen Multitasking-Anwendung zu beginnen. Die Beispielanwendung bietet ein Multitasking-Äquivalent.

1. Die Beispielanwendung finden Sie unter. `examples/[device_type_sample]`
2. Konvertieren Sie die `main.c` Datei in Ihr Projekt und fügen Sie einen Eintrag hinzu, um die bestehende main () -Funktion aufzurufen.
3. Stellen Sie sicher, dass Sie die Demo-Anwendung erfolgreich kompilieren können.

Technische Referenz

Themen

- Referenz: Unterstützte Plattformen
- Referenz: Technische Anforderungen
- Referenz: Allgemeine API

Referenz: Unterstützte Plattformen

In der folgenden Tabelle sind die unterstützten Plattformen für das SDK aufgeführt.

Unterstützte Plattformen

Plattform	Architektur	Betriebssystem
Linux x86_64	x86_64	Linux
Ambarella	Arm 8 () AArch64	Linux
Ein Mebad	Armv8-M 32-Bit	FreeRTOS
ESP32S3	Xtensa 32-Bit LX7	FreeRTOS

Referenz: Technische Anforderungen

Die folgende Tabelle zeigt die technischen Anforderungen für das SDK, einschließlich des RAM-Speicherplatzes. Das Endgeräte-SDK selbst benötigt etwa 5 bis 10 MB ROM-Speicherplatz, wenn dieselbe Konfiguration verwendet wird.

RAM-Speicherplatz

SDK und Komponenten	Speicheranforderungen (verwendete Byte)
Das SDK des Endgeräts selbst	180 KB
Standard-Befehlswarteschlange für den MQTT-Agenten	480 Byte (kann konfiguriert werden)
Standard-Eingangswarteschlange des MQTT-Agenten	320 Byte (kann konfiguriert werden)

Referenz: Allgemeine API

Dieser Abschnitt enthält eine Liste von API-Vorgängen, die nicht clusterspezifisch sind.

```
/* 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);
```

Sicherheit in verwalteten Integrationen für AWS IoT Device Management

Cloud-Sicherheit hat AWS höchste Priorität. Als AWS Kunde profitieren Sie von Rechenzentren und Netzwerkarchitekturen, die darauf ausgelegt sind, die Anforderungen der sicherheitssensibelsten Unternehmen zu erfüllen.

Sicherheit ist eine gemeinsame AWS Verantwortung von Ihnen und Ihnen. Das Modell der geteilten Verantwortung beschreibt dies als Sicherheit der Cloud selbst und Sicherheit in der Cloud:

- Sicherheit der Cloud — AWS ist verantwortlich für den Schutz der Infrastruktur, auf der AWS Dienste in der ausgeführt AWS Cloud werden. AWS bietet Ihnen auch Dienste, die Sie sicher nutzen können. Externe Prüfer testen und verifizieren regelmäßig die Wirksamkeit unserer Sicherheitsmaßnahmen im Rahmen der AWS . Weitere Informationen zu den Compliance-Programmen, die für verwaltete Integrationen gelten, finden Sie unter AWS Services im Bereich nach Compliance-Programm AWS .
- Sicherheit in der Cloud — Ihre Verantwortung richtet sich nach dem AWS Dienst, den Sie nutzen. Sie sind auch für andere Faktoren verantwortlich, etwa für die Vertraulichkeit Ihrer Daten, für die Anforderungen Ihres Unternehmens und für die geltenden Gesetze und Vorschriften.

Diese Dokumentation hilft Ihnen zu verstehen, wie Sie das Modell der gemeinsamen Verantwortung bei der Verwendung verwalteter Integrationen anwenden können. In den folgenden Themen erfahren Sie, wie Sie verwaltete Integrationen konfigurieren, um Ihre Sicherheits- und Compliance-Ziele zu erreichen. Sie erfahren auch, wie Sie andere AWS Dienste nutzen können, die Sie bei der Überwachung und Sicherung Ihrer Ressourcen für verwaltete Integrationen unterstützen.

Topics

Datenschutz in verwalteten Integrationen

Das Modell der AWS gemeinsamen Verantwortung und gilt für den Datenschutz in verwalteten Integrationen für. AWS IoT Device Management Wie in diesem Modell beschrieben, AWS ist verantwortlich für den Schutz der globalen Infrastruktur, auf der AWS Cloud alle Systeme laufen. Sie sind dafür verantwortlich, die Kontrolle über Ihre in dieser Infrastruktur gehosteten Inhalte zu behalten. Sie sind auch für die Sicherheitskonfiguration und die Verwaltungsaufgaben für die von Ihnen verwendeten AWS-Services verantwortlich. Weitere Informationen zum Datenschutz finden Sie unter Häufig gestellte Fragen zum Datenschutz. Informationen zum Datenschutz in Europa finden Sie im Blog-Beitrag AWS -Modell der geteilten Verantwortung und in der DSGVO im AWS -Sicherheitsblog.

Aus Datenschutzgründen empfehlen wir, dass Sie AWS-Konto Anmeldeinformationen schützen und einzelne Benutzer mit AWS IAM Identity Center oder AWS Identity and Access Management (IAM) einrichten. So erhält jeder Benutzer nur die Berechtigungen, die zum Durchführen seiner Aufgaben erforderlich sind. Außerdem empfehlen wir, die Daten mit folgenden Methoden schützen:

- Verwenden Sie für jedes Konto die Multi-Faktor-Authentifizierung (MFA).
- Wird verwendet SSL/TLS , um mit AWS Ressourcen zu kommunizieren. Wir benötigen TLS 1.2 und empfehlen TLS 1.3.
- Richten Sie die API und die Protokollierung von Benutzeraktivitäten mit ein AWS CloudTrail. Informationen zur Verwendung von CloudTrail Pfaden zur Erfassung von AWS Aktivitäten finden Sie unter Arbeiten mit CloudTrail Pfaden im AWS CloudTrail Benutzerhandbuch.
- Verwenden Sie AWS Verschlüsselungslösungen zusammen mit allen darin enthaltenen Standardsicherheitskontrollen AWS-Services.
- Verwenden Sie erweiterte verwaltete Sicherheitsservices wie Amazon Macie, die dabei helfen, in Amazon S3 gespeicherte persönliche Daten zu erkennen und zu schützen.
- Wenn Sie für den Zugriff AWS über eine Befehlszeilenschnittstelle oder eine API FIPS 140-3-validierte kryptografische Module benötigen, verwenden Sie einen FIPS-Endpunkt. Weitere Informationen über verfügbare FIPS-Endpunkte finden Sie unter Federal Information Processing Standard (FIPS) 140-3.

Wir empfehlen dringend, in Freitextfeldern, z. B. im Feld Name, keine vertraulichen oder sensiblen Informationen wie die E-Mail-Adressen Ihrer Kunden einzugeben. Dies gilt auch, wenn Sie mit verwalteten Integrationen für AWS IoT Device Management oder auf andere Weise AWS-Services über die Konsole, API oder arbeiten. AWS CLI AWS SDKs Alle Daten, die Sie in Tags

oder Freitextfelder eingeben, die für Namen verwendet werden, können für Abrechnungs- oder Diagnoseprotokolle verwendet werden. Wenn Sie eine URL für einen externen Server bereitstellen, empfehlen wir dringend, keine Anmeldeinformationen zur Validierung Ihrer Anforderung an den betreffenden Server in die URL einzuschließen.

Datenverschlüsselung im Ruhezustand für verwaltete Integrationen

Verwaltete Integrationen für die AWS IoT Device Management Verschlüsselung vertraulicher Kundendaten im Speicher standardmäßig mithilfe von Verschlüsselungsschlüsseln.

Es gibt zwei Arten von Verschlüsselungsschlüsseln, die zum Schutz sensibler Daten von Kunden mit verwalteten Integrationen verwendet werden:

Vom Kunden verwaltete Schlüssel (CMK)

Managed Integrations unterstützt die Verwendung von symmetrischen, vom Kunden verwalteten Schlüsseln, die Sie erstellen, besitzen und verwalten können. Sie haben die volle Kontrolle über diese KMS-Schlüssel. Dies gilt auch für die Festlegung und Verwaltung ihrer Schlüsselrichtlinien, IAM-Richtlinien und Erteilungen, ihre Aktivierung und Deaktivierung, die Drehung ihrer Verschlüsselungsinformationen, das Hinzufügen von Tags, das Erstellen von Aliassen, die sich auf die KMS-Schlüssel beziehen, und das Einplanen der KMS-Schlüssel zum Löschen.

AWS eigene Schlüssel

Managed Integrations verwendet diese Schlüssel standardmäßig, um sensible Kundendaten automatisch zu verschlüsseln. Sie können ihre Verwendung nicht einsehen, verwalten oder überprüfen. Sie müssen keine Maßnahmen ergreifen oder Programme ändern, um die Schlüssel zu schützen, mit denen Ihre Daten verschlüsselt werden. Die standardmäßige Verschlüsselung von Daten im Ruhezustand trägt dazu bei, den betrieblichen Aufwand und die Komplexität zu reduzieren, die mit dem Schutz vertraulicher Daten verbunden sind. Gleichzeitig können Sie damit sichere Anwendungen erstellen, die strenge Verschlüsselungsvorschriften und gesetzliche Auflagen erfüllen.

Als Standard-Verschlüsselungsschlüssel werden AWS eigene Schlüssel verwendet.
Alternativ ist die optionale API zur Aktualisierung Ihres Verschlüsselungsschlüssels `PutDefaultEncryptionConfiguration`.

Weitere Informationen zu den Arten von AWS KMS Verschlüsselungsschlüsseln finden Sie unter AWS KMS Schlüssel.

AWS KMS Verwendung für verwaltete Integrationen

Managed Integrations verschlüsselt und entschlüsselt alle Kundendaten mithilfe der Umschlagverschlüsselung. Bei dieser Art der Verschlüsselung werden Ihre Klartextdaten mit einem Datenschlüssel verschlüsselt. Als Nächstes verschlüsselt ein Verschlüsselungsschlüssel, der als Wrapping-Schlüssel bezeichnet wird, den ursprünglichen Datenschlüssel, der zur Verschlüsselung Ihrer Klartextdaten verwendet wurde. Bei der Umschlagverschlüsselung können zusätzliche Umschließungsschlüssel verwendet werden, um vorhandene Umschließungsschlüssel zu verschlüsseln, deren Abstand zum ursprünglichen Datenschlüssel enger ist. Da der ursprüngliche Datenschlüssel durch einen separat gespeicherten Umschließungsschlüssel verschlüsselt wird, können Sie den ursprünglichen Datenschlüssel und verschlüsselte Klartextdaten am selben Ort speichern. Ein Schlüsselbund wird verwendet, um Datenschlüssel zu generieren, zu verschlüsseln und zu entschlüsseln. Darüber hinaus wird der Umschließungsschlüssel zum Verschlüsseln und Entschlüsseln des Datenschlüssels verwendet.

> **Note**
>
> Das AWS Database Encryption SDK bietet Umschlagverschlüsselung für Ihre clientseitige Verschlüsselungsimplementierung. Weitere Informationen zum AWS Database Encryption SDK finden Sie unter Was ist das AWS Database Encryption SDK?

Weitere Informationen zur Umschlagverschlüsselung, zu Datenschlüsseln, Wrappeschlüsseln und Schlüsselbunden finden Sie unter Umschlagverschlüsselung, Datenschlüssel, Wrappeschlüssel und Schlüsselbunde.

Für verwaltete Integrationen müssen die Dienste Ihren vom Kunden verwalteten Schlüssel für die folgenden internen Vorgänge verwenden:

- Senden Sie `DescribeKey` Anfragen an, um AWS KMS zu überprüfen, ob die symmetrische, vom Kunden verwaltete Schlüssel-ID, die bei der Rotation der Datenschlüssel angegeben wurde, angegeben wurde.
- Senden Sie `GenerateDataKeyWithoutPlaintext` Anfragen AWS KMS zur Generierung von Datenschlüsseln, die mit Ihrem vom Kunden verwalteten Schlüssel verschlüsselt sind.
- Senden Sie `ReEncrypt*` Anfragen AWS KMS an, Datenschlüssel mit Ihrem vom Kunden verwalteten Schlüssel erneut zu verschlüsseln.
- Senden Sie `Decrypt` Anfragen AWS KMS an, Daten mit Ihrem vom Kunden verwalteten Schlüssel zu entschlüsseln.

Mit Verschlüsselungsschlüsseln verschlüsselte Datentypen

Verwaltete Integrationen verwenden Verschlüsselungsschlüssel, um mehrere Arten von Daten zu verschlüsseln, die im Ruhezustand gespeichert sind. In der folgenden Liste werden Datentypen beschrieben, die im Ruhezustand mithilfe von Verschlüsselungsschlüsseln verschlüsselt wurden:

- Cloud-to-Cloud (C2C) -Connector-Ereignisse wie Geräteerkennung und Aktualisierung des Gerätestatus.
- Erstellung eines verwalteten Objekts, das das physische Gerät darstellt, und eines Geräteprofils, das die Funktionen für einen bestimmten Gerätetyp enthält. Weitere Informationen zu einem Gerät und einem Geräteprofil finden Sie unter Gerät undGerät.
- Benachrichtigungen über verwaltete Integrationen zu verschiedenen Aspekten Ihrer Geräteimplementierung. Weitere Informationen zu Benachrichtigungen über verwaltete Integrationen finden Sie unter. Richten Sie Benachrichtigungen für verwaltete Integrationen ein
- Persönlich identifizierbare Informationen (PII) eines Endbenutzers wie Geräteauthentifizierungsmaterial, Geräteseriennummer, Name des Endbenutzers, Gerätekennung und Amazon-Ressourcenname (arn) des Geräts.

Wie verwaltete Integrationen wichtige Richtlinien verwenden in AWS KMS

Für die Rotation von Zweigstellenschlüsseln und asynchrone Anrufe ist für verwaltete Integrationen eine Schlüsselrichtlinie zur Verwendung Ihres Verschlüsselungsschlüssels erforderlich. Eine Schlüsselrichtlinie wird aus den folgenden Gründen verwendet:

- Autorisieren Sie programmgesteuert die Verwendung eines Verschlüsselungsschlüssels für andere Prinzipale. AWS

Ein Beispiel für eine Schlüsselrichtlinie, mit der der Zugriff auf Ihren Verschlüsselungsschlüssel in verwalteten Integrationen verwaltet wird, finden Sie unter Erstellen Sie einen Verschlüsselungsschlüssel

> **Note**
>
> Für einen AWS eigenen Schlüssel ist keine Schlüsselrichtlinie erforderlich, da der AWS Eigentümer des Schlüssels ist AWS und Sie ihn nicht anzeigen, verwalten oder verwenden

können. Managed Integrations verwendet standardmäßig den AWS eigenen Schlüssel, um Ihre sensiblen Kundendaten automatisch zu verschlüsseln.

Managed Integrations verwendet nicht nur wichtige Richtlinien für die Verwaltung Ihrer Verschlüsselungskonfiguration mit AWS KMS Schlüsseln, sondern auch IAM-Richtlinien. Weitere Informationen zu IAM-Richtlinien finden Sie unter Richtlinien und Berechtigungen in. AWS Identity and Access Management

Erstellen Sie einen Verschlüsselungsschlüssel

Sie können einen Verschlüsselungsschlüssel erstellen, indem Sie den AWS-Managementkonsole oder den verwenden AWS KMS APIs.

Um einen Verschlüsselungsschlüssel zu erstellen

Folgen Sie den Schritten zum Erstellen eines KMS-Schlüssels im AWS Key Management Service Entwicklerhandbuch.

Schlüsselrichtlinie

Eine wichtige Richtlinienerklärung steuert den Zugriff auf einen AWS KMS Schlüssel. Jeder AWS KMS Schlüssel wird nur eine wichtige Richtlinie enthalten. Diese wichtige Richtlinie legt fest, welche AWS Prinzipale den Schlüssel verwenden dürfen und wie sie ihn verwenden dürfen. Weitere Informationen zur Verwaltung des Zugriffs und der Verwendung von AWS KMS Schlüsseln mithilfe wichtiger Richtlinienerklärungen finden Sie unter Zugriff mithilfe von Richtlinien verwalten.

Im Folgenden finden Sie ein Beispiel für eine wichtige Richtlinienerklärung, mit der Sie den Zugriff und die Verwendung von AWS KMS Schlüsseln verwalten können, die in Ihren vier AWS-Konto verwalteten Integrationen gespeichert sind:

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

Weitere Informationen zu Schlüsselspeichern finden Sie unter Schlüsselspeicher.

Die Verschlüsselungskonfiguration wird aktualisiert

Die Möglichkeit, Ihre Verschlüsselungskonfiguration nahtlos zu aktualisieren, ist entscheidend für die Verwaltung Ihrer Datenverschlüsselungsimplementierung für verwaltete Integrationen. Wenn Sie die verwalteten Integrationen zum ersten Mal nutzen, werden Sie aufgefordert, Ihre Verschlüsselungskonfiguration auszuwählen. Ihre Optionen sind entweder die standardmäßigen eigenen Schlüssel AWS oder Sie können Ihren eigenen AWS KMS Schlüssel erstellen.

AWS-Managementkonsole

Um Ihre Verschlüsselungskonfiguration in zu aktualisieren AWS-Managementkonsole, öffnen Sie die AWS IoT Service-Homepage und navigieren Sie dann zu Managed Integration for Unified Control > Einstellungen > Verschlüsselung. Im Fenster mit den Verschlüsselungseinstellungen können Sie Ihre Verschlüsselungskonfiguration aktualisieren, indem Sie einen neuen AWS KMS Schlüssel für zusätzlichen Verschlüsselungsschutz auswählen. Wählen Sie Verschlüsselungseinstellungen anpassen (erweitert), um einen vorhandenen AWS KMS Schlüssel auszuwählen, oder wählen Sie AWS KMS Schlüssel erstellen, um Ihren eigenen vom Kunden verwalteten Schlüssel zu erstellen.

API-Befehle

Für die Verwaltung Ihrer Verschlüsselungskonfiguration von AWS KMS Schlüsseln in verwalteten Integrationen werden zwei APIs verwendet: `PutDefaultEncryptionConfiuration` und`GetDefaultEncryptionConfiguration`.

Rufen `PutDefaultEncryptionConfiuration` Sie an, um die standardmäßige Verschlüsselungskonfiguration zu aktualisieren. Weitere Informationen zu `PutDefaultEncryptionConfiuration` finden Sie unter PutDefaultEncryptionConfiuration.

Rufen Sie an, um die Standardverschlüsselungskonfiguration einzusehen`GetDefaultEncryptionConfiguration`. Weitere Informationen zu `GetDefaultEncryptionConfiguration` finden Sie unter GetDefaultEncryptionConfiguration.

Identitäts- und Zugriffsmanagement für verwaltete Integrationen

AWS Identity and Access Management (IAM) hilft einem Administrator AWS-Service , den Zugriff auf Ressourcen sicher zu kontrollieren. AWS IAM-Administratoren kontrollieren, wer authentifiziert (angemeldet) und autorisiert werden kann (über Berechtigungen verfügt), um verwaltete Integrationsressourcen zu verwenden. IAM ist ein Programm AWS-Service , das Sie ohne zusätzliche Kosten nutzen können.

Themen

Zielgruppe

Wie Sie AWS Identity and Access Management (IAM) verwenden, hängt von Ihrer Rolle ab:

- Servicebenutzer – fordern Sie von Ihrem Administrator Berechtigungen an, wenn Sie nicht auf Funktionen zugreifen können (siehe Fehlerbehebung bei verwalteten Integrationen, Identität und Zugriff)
- Dienstadministrator – bestimmen Sie den Benutzerzugriff und reichen Sie Berechtigungsanfragen ein (siehe Wie funktionieren verwaltete Integrationen mit IAM)
- IAM-Administrator – Schreiben Sie Richtlinien zur Zugriffsverwaltung (siehe Beispiele für identitätsbasierte Richtlinien für verwaltete Integrationen)

Authentifizierung mit Identitäten

Authentifizierung ist die Art und Weise, wie Sie sich AWS mit Ihren Identitätsdaten anmelden. Sie müssen sich als IAM-Benutzer authentifizieren oder eine IAM-Rolle annehmen. Root-Benutzer des AWS-Kontos

Sie können sich als föderierte Identität anmelden, indem Sie Anmeldeinformationen aus einer Identitätsquelle wie AWS IAM Identity Center (IAM Identity Center), Single Sign-On-Authentifizierung oder Anmeldeinformationen verwenden. Google/Facebook Weitere Informationen zum Anmelden finden Sie unter So melden Sie sich bei Ihrem AWS-Konto an im Benutzerhandbuch für AWS-Anmeldung .

AWS Bietet für den programmatischen Zugriff ein SDK und eine CLI zum kryptografischen Signieren von Anfragen. Weitere Informationen finden Sie unter AWS Signature Version 4 for API requests im IAM-Benutzerhandbuch.

AWS-Konto Root-Benutzer

Wenn Sie einen erstellen AWS-Konto, beginnen Sie mit einer Anmeldeidentität, dem sogenannten AWS-Konto Root-Benutzer, der vollständigen Zugriff auf alle AWS-Services Ressourcen hat. Wir raten ausdrücklich davon ab, den Root-Benutzer für Alltagsaufgaben zu verwenden. Eine Liste der Aufgaben, für die Sie sich als Root-Benutzer anmelden müssen, finden Sie unter Tasks that require root user credentials im IAM-Benutzerhandbuch.

Verbundidentität

Als bewährte Methode sollten menschliche Benutzer für den Zugriff AWS-Services mithilfe temporärer Anmeldeinformationen einen Verbund mit einem Identitätsanbieter verwenden.

Eine föderierte Identität ist ein Benutzer aus Ihrem Unternehmensverzeichnis, Ihrem Directory Service Web-Identitätsanbieter oder der AWS-Services mithilfe von Anmeldeinformationen aus einer Identitätsquelle zugreift. Verbundene Identitäten übernehmen Rollen, die temporäre Anmeldeinformationen bereitstellen.

Für die zentrale Zugriffsverwaltung empfehlen wir AWS IAM Identity Center. Weitere Informationen finden Sie unter Was ist IAM Identity Center? im AWS IAM Identity Center -Benutzerhandbuch.

IAM-Benutzer und -Gruppen

Ein IAM-Benutzer ist eine Identität mit bestimmten Berechtigungen für eine einzelne Person oder Anwendung. Verwenden Sie möglichst temporäre Anmeldeinformationen anstelle von IAM-Benutzern mit langfristigen Anmeldeinformationen. Weitere Informationen finden Sie im IAM-Benutzerhandbuch unter Erfordern, dass menschliche Benutzer den Verbund mit einem Identitätsanbieter verwenden müssen, um AWS mithilfe temporärer Anmeldeinformationen darauf zugreifen zu können.

Eine IAM-Gruppe spezifiziert eine Sammlung von IAM-Benutzern und erleichtert die Verwaltung von Berechtigungen für große Gruppen von Benutzern. Weitere Informationen finden Sie unter Anwendungsfälle für IAM-Benutzer im IAM-Benutzerhandbuch.

IAM-Rollen

Eine IAM-Rolle ist eine Identität mit spezifischen Berechtigungen, die temporäre Anmeldeinformationen bereitstellt. Sie können eine Rolle übernehmen, indem Sie von einer Benutzer-

zu einer IAM-Rolle (Konsole) wechseln AWS CLI oder einen AWS API-Vorgang aufrufen. Weitere Informationen finden Sie unter Methoden, um eine Rolle zu übernehmen im IAM-Benutzerhandbuch.

IAM-Rollen sind nützlich für Verbundbenutzerzugriff, temporäre IAM-Benutzerberechtigungen, kontoübergreifenden Zugriff, dienstübergreifenden Zugriff und Anwendungen, die auf Amazon ausgeführt werden. EC2 Weitere Informationen finden Sie unter Kontoübergreifender Ressourcenzugriff in IAM im IAM-Benutzerhandbuch.

Verwalten des Zugriffs mit Richtlinien

Sie kontrollieren den Zugriff, AWS indem Sie Richtlinien erstellen und diese an Identitäten oder Ressourcen anhängen. AWS Eine Richtlinie definiert Berechtigungen, wenn sie mit einer Identität oder Ressource verknüpft sind. AWS bewertet diese Richtlinien, wenn ein Principal eine Anfrage stellt. Die meisten Richtlinien werden AWS als JSON-Dokumente gespeichert. Weitere Informationen zu JSON-Richtliniendokumenten finden Sie unter Übersicht über JSON-Richtlinien im IAM-Benutzerhandbuch.

Mit Hilfe von Richtlinien legen Administratoren fest, wer Zugriff auf was hat, indem sie definieren, welches Prinzipal welche Aktionen auf welchen Ressourcenund unter welchen Bedingungendurchführen darf.

Standardmäßig haben Benutzer, Gruppen und Rollen keine Berechtigungen. Ein IAM-Administrator erstellt IAM-Richtlinien und fügt sie zu Rollen hinzu, die die Benutzer dann übernehmen können. IAM-Richtlinien definieren Berechtigungen unabhängig von der Methode, die zur Ausführung der Operation verwendet wird.

Identitätsbasierte Richtlinien

Identitätsbasierte Richtlinien sind JSON-Berechtigungsrichtliniendokumente, die Sie einer Identität (Benutzer, Gruppe oder Rolle) anfügen können. Diese Richtlinien steuern, welche Aktionen Identitäten für welche Ressourcen und unter welchen Bedingungen ausführen können. Informationen zum Erstellen identitätsbasierter Richtlinien finden Sie unter Definieren benutzerdefinierter IAM-Berechtigungen mit vom Kunden verwalteten Richtlinien im IAM-Benutzerhandbuch.

Identitätsbasierte Richtlinien können Inline-Richtlinien (direkt in eine einzelne Identität eingebettet) oder verwaltete Richtlinien (eigenständige Richtlinien, die mit mehreren Identitäten verbunden sind) sein. Informationen dazu, wie Sie zwischen verwalteten und Inline-Richtlinien wählen, finden Sie unter Choose between managed policies and inline policies im IAM-Benutzerhandbuch.

Ressourcenbasierte Richtlinien

Ressourcenbasierte Richtlinien sind JSON-Richtliniendokumente, die Sie an eine Ressource anfügen. Beispiele hierfür sind Vertrauensrichtlinien für IAM-Rollen und Amazon S3-Bucket-Richtlinien. In Services, die ressourcenbasierte Richtlinien unterstützen, können Service-Administratoren sie verwenden, um den Zugriff auf eine bestimmte Ressource zu steuern. Sie müssen in einer ressourcenbasierten Richtlinie einen Prinzipal angeben.

Ressourcenbasierte Richtlinien sind Richtlinien innerhalb dieses Diensts. Sie können AWS verwaltete Richtlinien von IAM nicht in einer ressourcenbasierten Richtlinie verwenden.

Weitere Richtlinientypen

AWS unterstützt zusätzliche Richtlinientypen, mit denen die maximalen Berechtigungen festgelegt werden können, die durch gängigere Richtlinientypen gewährt werden:

- Berechtigungsgrenzen – Eine Berechtigungsgrenze legt die maximalen Berechtigungen fest, die eine identitätsbasierte Richtlinie einer IAM-Entität erteilen kann. Weitere Informationen finden Sie unter Berechtigungsgrenzen für IAM-Entitäten im -IAM-Benutzerhandbuch.
- Richtlinien zur Dienstkontrolle (SCPs) — Geben Sie die maximalen Berechtigungen für eine Organisation oder Organisationseinheit in an AWS Organizations. Weitere Informationen finden Sie unter Service-Kontrollrichtlinien im AWS Organizations -Benutzerhandbuch.
- Richtlinien zur Ressourcenkontrolle (RCPs) — Legen Sie die maximal verfügbaren Berechtigungen für Ressourcen in Ihren Konten fest. Weitere Informationen finden Sie im AWS Organizations Benutzerhandbuch unter Richtlinien zur Ressourcenkontrolle (RCPs).
- Sitzungsrichtlinien – Sitzungsrichtlinien sind erweiterte Richtlinien, die als Parameter übergeben werden, wenn Sie eine temporäre Sitzung für eine Rolle oder einen Verbundbenutzer erstellen. Weitere Informationen finden Sie unter Sitzungsrichtlinien im IAM-Benutzerhandbuch.

Mehrere Richtlinientypen

Wenn für eine Anfrage mehrere Arten von Richtlinien gelten, sind die daraus resultierenden Berechtigungen schwieriger zu verstehen. Informationen darüber, wie AWS bestimmt wird, ob eine Anfrage zulässig ist, wenn mehrere Richtlinientypen betroffen sind, finden Sie unter Bewertungslogik für Richtlinien im IAM-Benutzerhandbuch.

AWS verwaltete Richtlinien für verwaltete Integrationen

Um Benutzern, Gruppen und Rollen Berechtigungen hinzuzufügen, ist es einfacher, AWS verwaltete Richtlinien zu verwenden, als Richtlinien selbst zu schreiben. Es erfordert Zeit und Fachwissen, um von Kunden verwaltete IAM-Richtlinien zu erstellen, die Ihrem Team nur die benötigten Berechtigungen bieten. Um schnell loszulegen, können Sie unsere AWS verwalteten Richtlinien verwenden. Diese Richtlinien decken allgemeine Anwendungsfälle ab und sind in Ihrem AWS-Konto verfügbar. Weitere Informationen zu AWS verwalteten Richtlinien finden Sie im IAM-Benutzerhandbuch unter AWS Verwaltete Richtlinien.

AWS Dienste verwalten und aktualisieren AWS verwaltete Richtlinien. Sie können die Berechtigungen in AWS verwalteten Richtlinien nicht ändern. Services fügen einer von AWS verwalteten Richtlinien gelegentlich zusätzliche Berechtigungen hinzu, um neue Features zu unterstützen. Diese Art von Update betrifft alle Identitäten (Benutzer, Gruppen und Rollen), an welche die Richtlinie angehängt ist. Services aktualisieren eine von AWS verwaltete Richtlinie am ehesten, ein neues Feature gestartet wird oder neue Vorgänge verfügbar werden. Dienste entfernen keine Berechtigungen aus einer AWS verwalteten Richtlinie, sodass durch Richtlinienaktualisierungen Ihre bestehenden Berechtigungen nicht beeinträchtigt werden.

AWS Unterstützt außerdem verwaltete Richtlinien für Jobfunktionen, die sich über mehrere Dienste erstrecken. Die ReadOnlyAccess AWS verwaltete Richtlinie bietet beispielsweise schreibgeschützten Zugriff auf alle AWS Dienste und Ressourcen. Wenn ein Dienst eine neue Funktion startet, werden nur Leseberechtigungen für neue Operationen und Ressourcen AWS hinzugefügt. Eine Liste und Beschreibungen der Richtlinien für Auftragsfunktionen finden Sie in Verwaltete AWS -Richtlinien für Auftragsfunktionen im IAM-Leitfaden.

AWS verwaltete Richtlinie: AWSIo TManaged IntegrationsFullAccess

Sie können die `AWSIoTManagedIntegrationsFullAccess`-Richtlinie an Ihre IAM-Identitäten anfügen.

Diese Richtlinie gewährt volle Zugriffsberechtigungen für verwaltete Integrationen und zugehörige Dienste. Informationen zu dieser Richtlinie finden Sie AWS-Managementkonsole unter AWSIoTManagedIntegrationsFullAccess.

Details zu Berechtigungen

Diese Richtlinie umfasst die folgenden Berechtigungen:

- `iotmanagedintegrations`— Bietet vollen Zugriff auf verwaltete Integrationen und zugehörige Dienste für die IAM-Benutzer, -Gruppen und -Rollen, denen Sie diese Richtlinie hinzufügen.
- `iam`— Ermöglicht den zugewiesenen IAM-Benutzern, -Gruppen und -Rollen das Erstellen einer dienstbezogenen Rolle in einem. AWS-Konto

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iotmanagedintegrations:*",
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": "iam:CreateServiceLinkedRole",
      "Resource": "arn:aws:iam::*:role/aws-service-role/
iotmanagedintegrations.amazonaws.com/AWSServiceRoleForIoTManagedIntegrations",
      "Condition": {
        "StringEquals": {
          "iam:AWSServiceName": "iotmanagedintegrations.amazonaws.com"
        }
      }
    }
  ]
}
```

AWS verwaltete Richtlinie: Io AWS TManaged IntegrationsRolePolicy

Sie können die `AWS IoTManagedIntegrationsRolePolicy`-Richtlinie an Ihre IAM-Identitäten anfügen.

Diese Richtlinie gewährt verwalteten Integrationen die Erlaubnis, CloudWatch Amazon-Logs und -Metriken in Ihrem Namen zu veröffentlichen.

Informationen zu dieser Richtlinie finden Sie AWS-Managementkonsole unter AWSIoTManagedIntegrationsRolePolicy.

Details zu Berechtigungen

Diese Richtlinie umfasst die folgenden Berechtigungen.

- `logs`— Bietet die Möglichkeit, CloudWatch Amazon-Protokollgruppen zu erstellen und Protokolle an die Gruppen zu streamen.
- `cloudwatch`— Bietet die Möglichkeit, CloudWatch Amazon-Metriken zu veröffentlichen. Weitere Informationen zu CloudWatch Amazon-Metriken finden Sie unter Metriken in Amazon CloudWatch.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "CloudWatchLogs",
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup"
      ],
      "Resource": [
        "arn:aws:logs:*:*:log-group:/aws/iotmanagedintegrations/*"
      ],
      "Condition": {
        "StringEquals": {
          "aws:PrincipalAccount": "${aws:ResourceAccount}"
        }
      }
    },
    {
      "Sid": "CloudWatchStreams",
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogStream",
        "logs:PutLogEvents"
      ],
      "Resource": [
        "arn:aws:logs:*:*:log-group:/aws/iotmanagedintegrations/*:log-stream:*"
      ],
      "Condition": {
        "StringEquals": {
```

```
          "aws:PrincipalAccount": "${aws:ResourceAccount}"
        }
      }
    },
    {
      "Sid": "CloudWatchMetrics",
      "Effect": "Allow",
      "Action": [
        "cloudwatch:PutMetricData"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "cloudwatch:namespace": [
            "AWS/IoTManagedIntegrations",
            "AWS/Usage"
          ]
        }
      }
    }
  ]
}
```

Verwaltete Integrationen, Aktualisierungen AWS verwalteter Richtlinien

Hier finden Sie Informationen zu Aktualisierungen der AWS verwalteten Richtlinien für verwaltete Integrationen, seit dieser Dienst begonnen hat, diese Änderungen nachzuverfolgen. Wenn Sie automatische Benachrichtigungen über Änderungen an dieser Seite erhalten möchten, abonnieren Sie den RSS-Feed auf der Seite mit dem Dokumentverlauf verwalteter Integrationen.

Änderungen	Beschreibung	Date
Verwaltete Integrationen haben begonnen, Änderungen nachzuverfolgen	Managed Integrations hat damit begonnen, Änderungen an den AWS verwalteten Richtlinien nachzuverfolgen.	03. März 2025

Wie funktionieren verwaltete Integrationen mit IAM

Bevor Sie IAM verwenden, um den Zugriff auf verwaltete Integrationen zu verwalten, sollten Sie sich darüber informieren, welche IAM-Funktionen für verwaltete Integrationen verfügbar sind.

IAM-Funktionen, die Sie mit verwalteten Integrationen verwenden können

IAM-Feature	Unterstützung für verwaltete Integrationen
Identitätsbasierte Richtlinien	Ja
Ressourcenbasierte Richtlinien	Nein
Richtlinienaktionen	Ja
Richtlinienressourcen	Ja
Bedingungsschlüssel für die Richtlinie	Ja
ACLs	Nein
ABAC (Tags in Richtlinien)	Nein
Temporäre Anmeldeinformationen	Ja
Prinzipalberechtigungen	Ja
Servicerollen	Ja
Service-verknüpfte Rollen	Ja

Einen allgemeinen Überblick darüber, wie verwaltete Integrationen und andere AWS Dienste mit den meisten IAM-Funktionen funktionieren, finden Sie im AWS IAM-Benutzerhandbuch unter Dienste, die mit IAM funktionieren.

Identitätsbasierte Richtlinien für verwaltete Integrationen

Unterstützt Richtlinien auf Identitätsbasis: Ja

Identitätsbasierte Richtlinien sind JSON-Berechtigungsrichtliniendokumente, die Sie einer Identität anfügen können, wie z. B. IAM-Benutzern, -Benutzergruppen oder -Rollen. Diese Richtlinien steuern,

welche Aktionen die Benutzer und Rollen für welche Ressourcen und unter welchen Bedingungen ausführen können. Informationen zum Erstellen identitätsbasierter Richtlinien finden Sie unter Definieren benutzerdefinierter IAM-Berechtigungen mit vom Kunden verwalteten Richtlinien im IAM-Benutzerhandbuch.

Mit identitätsbasierten IAM-Richtlinien können Sie angeben, welche Aktionen und Ressourcen zugelassen oder abgelehnt werden. Darüber hinaus können Sie die Bedingungen festlegen, unter denen Aktionen zugelassen oder abgelehnt werden. Informationen zu sämtlichen Elementen, die Sie in einer JSON-Richtlinie verwenden, finden Sie in der IAM-Referenz für JSON-Richtlinienelemente im IAM-Benutzerhandbuch.

Beispiele für identitätsbasierte Richtlinien für verwaltete Integrationen

Beispiele für identitätsbasierte Richtlinien für verwaltete Integrationen finden Sie unter. Beispiele für identitätsbasierte Richtlinien für verwaltete Integrationen

Ressourcenbasierte Richtlinien innerhalb verwalteter Integrationen

Unterstützt ressourcenbasierte Richtlinien: Nein

Ressourcenbasierte Richtlinien sind JSON-Richtliniendokumente, die Sie an eine Ressource anfügen. Beispiele für ressourcenbasierte Richtlinien sind IAM-Rollen-Vertrauensrichtlinien und Amazon-S3-Bucket-Richtlinien. In Services, die ressourcenbasierte Richtlinien unterstützen, können Service-Administratoren sie verwenden, um den Zugriff auf eine bestimmte Ressource zu steuern. Für die Ressource, an welche die Richtlinie angehängt ist, legt die Richtlinie fest, welche Aktionen ein bestimmter Prinzipal unter welchen Bedingungen für diese Ressource ausführen kann. Sie müssen in einer ressourcenbasierten Richtlinie einen Prinzipal angeben. Zu den Prinzipalen können Konten, Benutzer, Rollen, Verbundbenutzer oder gehören. AWS-Services

Um kontoübergreifenden Zugriff zu ermöglichen, können Sie ein gesamtes Konto oder IAM-Entitäten in einem anderen Konto als Prinzipal in einer ressourcenbasierten Richtlinie angeben. Weitere Informationen finden Sie unter Kontoübergreifender Ressourcenzugriff in IAM im IAM-Benutzerhandbuch.

Richtlinienaktionen für verwaltete Integrationen

Unterstützt Richtlinienaktionen: Ja

Administratoren können mithilfe von AWS JSON-Richtlinien angeben, wer Zugriff auf was hat. Das heißt, welcher Prinzipal Aktionen für welche Ressourcen und unter welchen Bedingungen ausführen kann.

Das Element `Action` einer JSON-Richtlinie beschreibt die Aktionen, mit denen Sie den Zugriff in einer Richtlinie zulassen oder verweigern können. Nehmen Sie Aktionen in eine Richtlinie auf, um Berechtigungen zur Ausführung des zugehörigen Vorgangs zu erteilen.

Eine Liste der Aktionen für verwaltete Integrationen finden Sie unter Durch verwaltete Integrationen definierte Aktionen in der Serviceautorisierungsreferenz.

Bei Richtlinienaktionen in verwalteten Integrationen wird vor der Aktion das folgende Präfix verwendet:

```
iot-mi
```

Um mehrere Aktionen in einer einzigen Anweisung anzugeben, trennen Sie sie mit Kommata:

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

Beispiele für identitätsbasierte Richtlinien für verwaltete Integrationen finden Sie unter. Beispiele für identitätsbasierte Richtlinien für verwaltete Integrationen

Richtlinienressourcen für verwaltete Integrationen

Unterstützt Richtlinienressourcen: Ja

Administratoren können mithilfe von AWS JSON-Richtlinien angeben, wer Zugriff auf was hat. Das heißt, welcher Prinzipal Aktionen für welche Ressourcen und unter welchen Bedingungen ausführen kann.

Das JSON-Richtlinienelement `Resource` gibt die Objekte an, auf welche die Aktion angewendet wird. Als Best Practice geben Sie eine Ressource mit dem zugehörigen Amazon-Ressourcennamen (ARN) an. Verwenden Sie für Aktionen, die keine Berechtigungen auf Ressourcenebene unterstützen, einen Platzhalter (*), um anzugeben, dass die Anweisung für alle Ressourcen gilt.

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

Eine Liste der Ressourcentypen und ihrer Ressourcen für verwaltete Integrationen finden Sie unter Ressourcen ARNs, die durch verwaltete Integrationen definiert sind in der Serviceautorisierungsreferenz. Informationen zu den Aktionen, mit denen Sie den ARN jeder Ressource angeben können, finden Sie unter Von verwalteten Integrationen definierte Aktionen.

Beispiele für identitätsbasierte Richtlinien für verwaltete Integrationen finden Sie unter. Beispiele für identitätsbasierte Richtlinien für verwaltete Integrationen

Bedingungsschlüssel für Richtlinien für verwaltete Integrationen

Unterstützt servicespezifische Richtlinienbedingungsschlüssel: Ja

Administratoren können mithilfe von AWS JSON-Richtlinien angeben, wer auf was Zugriff hat. Das heißt, welcher Prinzipal Aktionen für welche Ressourcen und unter welchen Bedingungen ausführen kann.

Das Element `Condition` gibt an, wann Anweisungen auf der Grundlage definierter Kriterien ausgeführt werden. Sie können bedingte Ausdrücke erstellen, die Bedingungsoperatoren verwenden, z. B. ist gleich oder kleiner als, damit die Bedingung in der Richtlinie mit Werten in der Anforderung übereinstimmt. Eine Übersicht aller AWS globalen Bedingungsschlüssel finden Sie unter Kontextschlüssel für AWS globale Bedingungen im IAM-Benutzerhandbuch.

Eine Liste der Bedingungsschlüssel für verwaltete Integrationen finden Sie unter Bedingungsschlüssel für verwaltete Integrationen in der Serviceautorisierungsreferenz. Informationen zu den Aktionen und Ressourcen, mit denen Sie einen Bedingungsschlüssel verwenden können, finden Sie unter Durch verwaltete Integrationen definierte Aktionen.

Beispiele für identitätsbasierte Richtlinien für verwaltete Integrationen finden Sie unter. Beispiele für identitätsbasierte Richtlinien für verwaltete Integrationen

ACLs in verwalteten Integrationen

Unterstützt ACLs: Nein

Zugriffskontrolllisten (ACLs) steuern, welche Principals (Kontomitglieder, Benutzer oder Rollen) über Zugriffsberechtigungen für eine Ressource verfügen. ACLs ähneln ressourcenbasierten Richtlinien, verwenden jedoch nicht das JSON-Richtliniendokumentformat.

ABAC mit verwalteten Integrationen

Unterstützt ABAC (Tags in Richtlinien): Teilweise

Die attributbasierte Zugriffskontrolle (ABAC) ist eine Autorisierungsstrategie, bei der Berechtigungen basierend auf Attributen, auch als Tags bezeichnet, definiert werden. Sie können Tags an IAM-Entitäten und AWS -Ressourcen anhängen und dann ABAC-Richtlinien entwerfen, um Operationen zu ermöglichen, wenn das Tag des Prinzipals mit dem Tag auf der Ressource übereinstimmt.

Um den Zugriff auf der Grundlage von Tags zu steuern, geben Sie im Bedingungselement einer Richtlinie Tag-Informationen an, indem Sie die Schlüssel `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, oder Bedingung `aws:TagKeys` verwenden.

Wenn ein Service alle drei Bedingungsschlüssel für jeden Ressourcentyp unterstützt, lautet der Wert für den Service Ja. Wenn ein Service alle drei Bedingungsschlüssel für nur einige Ressourcentypen unterstützt, lautet der Wert Teilweise.

Weitere Informationen zu ABAC finden Sie unter Definieren von Berechtigungen mit ABAC-Autorisierung im IAM-Benutzerhandbuch. Um ein Tutorial mit Schritten zur Einstellung von ABAC anzuzeigen, siehe Attributbasierte Zugriffskontrolle (ABAC) verwenden im IAM-Benutzerhandbuch.

Verwendung temporärer Anmeldeinformationen bei verwalteten Integrationen

Unterstützt temporäre Anmeldeinformationen: Ja

Temporäre Anmeldeinformationen ermöglichen kurzfristigen Zugriff auf AWS Ressourcen und werden automatisch erstellt, wenn Sie einen Verbund verwenden oder die Rollen wechseln. AWS empfiehlt, temporäre Anmeldeinformationen dynamisch zu generieren, anstatt langfristige Zugriffsschlüssel zu verwenden. Weitere Informationen finden Sie unter Temporäre Anmeldeinformationen in IAM und AWS-Services , die mit IAM funktionieren im IAM-Benutzerhandbuch.

Serviceübergreifende Prinzipalberechtigungen für verwaltete Integrationen

Unterstützt Forward Access Sessions (FAS): Ja

Forward Access Sessions (FAS) verwenden die Berechtigungen des Prinzipals, der einen aufruft AWS-Service, in Kombination mit der Anforderung, Anfragen an AWS-Service nachgelagerte Dienste zu stellen. Einzelheiten zu den Richtlinien für FAS-Anforderungen finden Sie unter Zugriffssitzungen weiterleiten.

Servicerollen für verwaltete Integrationen

Unterstützt Servicerollen: Ja

Eine Servicerolle ist eine IAM-Rolle, die ein Service annimmt, um Aktionen in Ihrem Namen auszuführen. Ein IAM-Administrator kann eine Servicerolle innerhalb von IAM erstellen, ändern und löschen. Weitere Informationen finden Sie unter Erstellen einer Rolle zum Delegieren von Berechtigungen an einen AWS-Service im IAM-Benutzerhandbuch.

> **Warning**
>
> Das Ändern der Berechtigungen für eine Servicerolle kann dazu führen, dass die Funktionalität verwalteter Integrationen beeinträchtigt wird. Bearbeiten Sie Servicerollen nur, wenn verwaltete Integrationen eine Anleitung dazu enthalten.

Mit Diensten verknüpfte Rollen für verwaltete Integrationen

Unterstützt serviceverknüpfte Rollen: Ja

Eine serviceverknüpfte Rolle ist eine Art von Servicerolle, die mit einer verknüpft ist. AWS-Service Der Service kann die Rolle übernehmen, um eine Aktion in Ihrem Namen auszuführen. Dienstbezogene Rollen werden in Ihrem Dienst angezeigt AWS-Konto und gehören dem Dienst. Ein IAM-Administrator kann die Berechtigungen für Service-verknüpfte Rollen anzeigen, aber nicht bearbeiten.

Details zum Erstellen oder Verwalten von serviceverknüpften Rollen finden Sie unter AWS -Services, die mit IAM funktionieren. Suchen Sie in der Tabelle nach einem Service mit einem `Yes` in der Spalte Service-linked role (Serviceverknüpfte Rolle). Wählen Sie den Link Yes (Ja) aus, um die Dokumentation für die serviceverknüpfte Rolle für diesen Service anzuzeigen.

Beispiele für identitätsbasierte Richtlinien für verwaltete Integrationen

Standardmäßig sind Benutzer und Rollen nicht berechtigt, Ressourcen für verwaltete Integrationen zu erstellen oder zu ändern. Ein IAM-Administrator muss IAM-Richtlinien erstellen, die Benutzern die Berechtigung erteilen, Aktionen für die Ressourcen auszuführen, die sie benötigen.

Informationen dazu, wie Sie unter Verwendung dieser beispielhaften JSON-Richtliniendokumente eine identitätsbasierte IAM-Richtlinie erstellen, finden Sie unter Erstellen von IAM-Richtlinien (Konsole) im IAM-Benutzerhandbuch.

Einzelheiten zu Aktionen und Ressourcentypen, die durch verwaltete Integrationen definiert werden, einschließlich des Formats ARNs für die einzelnen Ressourcentypen, finden Sie unter Aktionen, Ressourcen und Bedingungsschlüssel für verwaltete Integrationen in der Referenz zur Serviceautorisierung.

Themen

- Best Practices für Richtlinien
- Verwenden Sie die Konsole für verwaltete Integrationen
- Gewähren der Berechtigung zur Anzeige der eigenen Berechtigungen für Benutzer

Best Practices für Richtlinien

Identitätsbasierte Richtlinien legen fest, ob jemand Ressourcen für verwaltete Integrationen in Ihrem Konto erstellen, darauf zugreifen oder sie löschen kann. Dies kann zusätzliche Kosten für Ihr verursachen AWS-Konto. Beachten Sie beim Erstellen oder Bearbeiten identitätsbasierter Richtlinien die folgenden Richtlinien und Empfehlungen:

- Erste Schritte mit AWS verwalteten Richtlinien und Umstellung auf Berechtigungen mit den geringsten Rechten — Verwenden Sie die AWS verwalteten Richtlinien, die Berechtigungen für viele gängige Anwendungsfälle gewähren, um damit zu beginnen, Ihren Benutzern und Workloads Berechtigungen zu gewähren. Sie sind in Ihrem verfügbar. AWS-Konto Wir empfehlen Ihnen, die Berechtigungen weiter zu reduzieren, indem Sie vom AWS Kunden verwaltete Richtlinien definieren, die speziell auf Ihre Anwendungsfälle zugeschnitten sind. Weitere Informationen finden Sie unter Von AWS verwaltete Richtlinien oder Von AWS verwaltete Richtlinien für Auftragsfunktionen im IAM-Benutzerhandbuch.
- Anwendung von Berechtigungen mit den geringsten Rechten – Wenn Sie mit IAM-Richtlinien Berechtigungen festlegen, gewähren Sie nur die Berechtigungen, die für die Durchführung einer Aufgabe erforderlich sind. Sie tun dies, indem Sie die Aktionen definieren, die für bestimmte Ressourcen unter bestimmten Bedingungen durchgeführt werden können, auch bekannt als die geringsten Berechtigungen. Weitere Informationen zur Verwendung von IAM zum Anwenden von Berechtigungen finden Sie unter Richtlinien und Berechtigungen in IAM im IAM-Benutzerhandbuch.
- Verwenden von Bedingungen in IAM-Richtlinien zur weiteren Einschränkung des Zugriffs – Sie können Ihren Richtlinien eine Bedingung hinzufügen, um den Zugriff auf Aktionen und Ressourcen zu beschränken. Sie können beispielsweise eine Richtlinienbedingung schreiben, um festzulegen, dass alle Anforderungen mithilfe von SSL gesendet werden müssen. Sie können

auch Bedingungen verwenden, um Zugriff auf Serviceaktionen zu gewähren, wenn diese für einen bestimmten Zweck verwendet werden AWS-Service, z. CloudFormation B. Weitere Informationen finden Sie unter IAM-JSON-Richtlinienelemente: Bedingung im IAM-Benutzerhandbuch.

- Verwenden von IAM Access Analyzer zur Validierung Ihrer IAM-Richtlinien, um sichere und funktionale Berechtigungen zu gewährleisten – IAM Access Analyzer validiert neue und vorhandene Richtlinien, damit die Richtlinien der IAM-Richtliniensprache (JSON) und den bewährten IAM-Methoden entsprechen. IAM Access Analyzer stellt mehr als 100 Richtlinienprüfungen und umsetzbare Empfehlungen zur Verfügung, damit Sie sichere und funktionale Richtlinien erstellen können. Weitere Informationen finden Sie unter Richtlinienvalidierung mit IAM Access Analyzer im IAM-Benutzerhandbuch.
- Multi-Faktor-Authentifizierung (MFA) erforderlich — Wenn Sie ein Szenario haben, das IAM-Benutzer oder einen Root-Benutzer in Ihrem System erfordert AWS-Konto, aktivieren Sie MFA für zusätzliche Sicherheit. Um MFA beim Aufrufen von API-Vorgängen anzufordern, fügen Sie Ihren Richtlinien MFA-Bedingungen hinzu. Weitere Informationen finden Sie unter Sicherer API-Zugriff mit MFA im IAM-Benutzerhandbuch.

Weitere Informationen zu bewährten Methoden in IAM finden Sie unter Best Practices für die Sicherheit in IAM im IAM-Benutzerhandbuch.

Verwenden Sie die Konsole für verwaltete Integrationen

Um auf die Konsole für verwaltete Integrationen zugreifen zu können, benötigen Sie ein Mindestmaß an Berechtigungen. Diese Berechtigungen müssen es Ihnen ermöglichen, die Ressourcen für verwaltete Integrationen in Ihrem aufzulisten und Details zu diesen Ressourcen anzuzeigen. AWS-Konto Wenn Sie eine identitätsbasierte Richtlinie erstellen, die strenger ist als die mindestens erforderlichen Berechtigungen, funktioniert die Konsole nicht wie vorgesehen für Entitäten (Benutzer oder Rollen) mit dieser Richtlinie.

Sie müssen Benutzern, die nur die API AWS CLI oder die AWS API aufrufen, keine Mindestberechtigungen für die Konsole gewähren. Stattdessen sollten Sie nur Zugriff auf die Aktionen zulassen, die der API-Operation entsprechen, die die Benutzer ausführen möchten.

Um sicherzustellen, dass Benutzer und Rollen die Konsole für verwaltete Integrationen weiterhin verwenden können, fügen Sie den Entitäten auch die verwalteten Integrationen `ConsoleAccess` oder `ReadOnly` AWS verwalteten Richtlinien hinzu. Weitere Informationen finden Sie unter Hinzufügen von Berechtigungen zu einem Benutzer im IAM-Benutzerhandbuch.

Gewähren der Berechtigung zur Anzeige der eigenen Berechtigungen für Benutzer

In diesem Beispiel wird gezeigt, wie Sie eine Richtlinie erstellen, die IAM-Benutzern die Berechtigung zum Anzeigen der eingebundenen Richtlinien und verwalteten Richtlinien gewährt, die ihrer Benutzeridentität angefügt sind. Diese Richtlinie umfasst Berechtigungen zum Ausführen dieser Aktion auf der Konsole oder programmgesteuert mithilfe der OR-API. AWS CLI AWS

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

Fehlerbehebung bei verwalteten Integrationen, Identität und Zugriff

Verwenden Sie die folgenden Informationen, um häufig auftretende Probleme zu diagnostizieren und zu beheben, die bei der Arbeit mit verwalteten Integrationen und IAM auftreten können.

Themen

Ich bin nicht berechtigt, eine Aktion in verwalteten Integrationen durchzuführen

Wenn Sie eine Fehlermeldung erhalten, dass Sie nicht zur Durchführung einer Aktion berechtigt sind, müssen Ihre Richtlinien aktualisiert werden, damit Sie die Aktion durchführen können.

Der folgende Beispielfehler tritt auf, wenn der IAM-Benutzer `mateojackson` versucht, über die Konsole Details zu einer fiktiven *`my-example-widget`*-Ressource anzuzeigen, jedoch nicht über `iot-mi:`*`GetWidget`*-Berechtigungen verfügt.

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

In diesem Fall muss die Richtlinie für den Benutzer `mateojackson` aktualisiert werden, damit er mit der `iot-mi:`*`GetWidget`*-Aktion auf die *`my-example-widget`*-Ressource zugreifen kann.

Wenn Sie Hilfe benötigen, wenden Sie sich an Ihren AWS Administrator. Ihr Administrator hat Ihnen Ihre Anmeldeinformationen zur Verfügung gestellt.

Ich bin nicht berechtigt, iam auszuführen: PassRole

Wenn Sie die Fehlermeldung erhalten, dass Sie nicht zur Durchführung der `iam:PassRole` Aktion berechtigt sind, müssen Ihre Richtlinien aktualisiert werden, damit Sie eine Rolle an verwaltete Integrationen übergeben können.

Einige AWS-Services ermöglichen es Ihnen, eine bestehende Rolle an diesen Dienst zu übergeben, anstatt eine neue Servicerolle oder eine dienstverknüpfte Rolle zu erstellen. Hierzu benötigen Sie Berechtigungen für die Übergabe der Rolle an den Dienst.

Der folgende Beispielfehler tritt auf, wenn ein IAM-Benutzer mit dem Namen `marymajor` versucht, die Konsole zu verwenden, um eine Aktion in verwalteten Integrationen auszuführen. Die Aktion erfordert jedoch, dass der Service über Berechtigungen verfügt, die durch eine Servicerolle gewährt werden. Mary besitzt keine Berechtigungen für die Übergabe der Rolle an den Dienst.

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

In diesem Fall müssen die Richtlinien von Mary aktualisiert werden, um die Aktion `iam:PassRole` ausführen zu können.

Wenn Sie Hilfe benötigen, wenden Sie sich an Ihren AWS Administrator. Ihr Administrator hat Ihnen Ihre Anmeldeinformationen zur Verfügung gestellt.

Ich möchte Personen außerhalb von mir den Zugriff AWS-Konto auf meine Ressourcen für verwaltete Integrationen ermöglichen

Sie können eine Rolle erstellen, mit der Benutzer in anderen Konten oder Personen außerhalb Ihrer Organisation auf Ihre Ressourcen zugreifen können. Sie können festlegen, wem die Übernahme der Rolle anvertraut wird. Für Dienste, die ressourcenbasierte Richtlinien oder Zugriffskontrolllisten (ACLs) unterstützen, können Sie diese Richtlinien verwenden, um Personen Zugriff auf Ihre Ressourcen zu gewähren.

Weitere Informationen dazu finden Sie hier:

- Informationen darüber, ob verwaltete Integrationen diese Funktionen unterstützen, finden Sie unter. Wie funktionieren verwaltete Integrationen mit IAM
- Informationen dazu, wie Sie Zugriff auf Ihre Ressourcen gewähren können, AWS-Konten die Ihnen gehören, finden Sie im IAM-Benutzerhandbuch unter Gewähren des Zugriffs für einen IAM-Benutzer in einem anderen AWS-Konto , den Sie besitzen.
- Informationen dazu, wie Sie Dritten Zugriff auf Ihre Ressourcen gewähren können AWS-Konten, finden Sie AWS-Konten im IAM-Benutzerhandbuch unter Gewähren des Zugriffs für Dritte.
- Informationen dazu, wie Sie über einen Identitätsverbund Zugriff gewähren, finden Sie unter Gewähren von Zugriff für extern authentifizierte Benutzer (Identitätsverbund) im IAM-Benutzerhandbuch.
- Informationen zum Unterschied zwischen der Verwendung von Rollen und ressourcenbasierten Richtlinien für den kontoübergreifenden Zugriff finden Sie unter Kontoübergreifender Ressourcenzugriff in IAM im IAM-Benutzerhandbuch.

Verwenden von serviceverknüpften Rollen für verwaltete Integrationen

Verwaltete Integrationen für AWS IoT Device Management verwenden dienstgebundene Rollen AWS Identity and Access Management (IAM). Eine dienstverknüpfte Rolle ist eine einzigartige Art von IAM-Rolle, die direkt mit verwalteten Integrationen verknüpft ist. Mit Diensten verknüpfte Rollen sind durch verwaltete Integrationen vordefiniert und enthalten alle Berechtigungen, die der Dienst benötigt, um andere AWS Dienste in Ihrem Namen aufzurufen.

Eine dienstbezogene Rolle erleichtert die Einrichtung verwalteter Integrationen, da Sie die erforderlichen Berechtigungen nicht manuell hinzufügen müssen. Verwaltete Integrationen für die AWS IoT Geräteverwaltung definieren die Berechtigungen ihrer dienstbezogenen Rollen. Sofern nicht anders definiert, können nur verwaltete Integrationen ihre Rollen übernehmen. Die definierten Berechtigungen umfassen die Vertrauens- und Berechtigungsrichtlinie. Diese Berechtigungsrichtlinie kann keinen anderen IAM-Entitäten zugewiesen werden.

Sie können eine serviceverknüpfte Rolle erst löschen, nachdem ihre verwandten Ressourcen gelöscht wurden. Dadurch werden Ihre Ressourcen für verwaltete Integrationen geschützt, da Sie die Zugriffsberechtigung für die Ressourcen nicht versehentlich entziehen können.

Informationen zu anderen Diensten, die dienstverknüpfte Rollen unterstützen, finden Sie unter AWS Dienste, die mit IAM funktionieren. Suchen Sie in der Spalte Dienstverknüpfte Rollen nach den Diensten, für die Ja steht. Wählen Sie über einen Link Ja aus, um die Dokumentation zu einer serviceverknüpften Rolle für diesen Service anzuzeigen.

Dienstbezogene Rollenberechtigungen für verwaltete Integrationen

Verwaltete Integrationen für die AWS IoT Geräteverwaltung verwenden die dienstbezogene Rolle AWSServiceRoleForIoTManagedIntegrations — Ermöglicht verwaltete Integrationen für die AWS IoT Geräteverwaltung, Protokolle und Metriken in Ihrem Namen zu veröffentlichen.

Die mit dem Dienst verknüpfte Rolle „ AWSServiceRoleForIoTManagedIntegrations“ vertraut darauf, dass die folgenden Dienste die Rolle übernehmen:

- `iotmanagedintegrations.amazonaws.com`

Die genannte Richtlinie für Rollenberechtigungen AWSIo TManaged IntegrationsServiceRolePolicy ermöglicht verwalteten Integrationen, die folgenden Aktionen an den angegebenen Ressourcen durchzuführen:

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

JSON

```
{
  "Version":"2012-10-17",
  "Statement" : [
    {
      "Sid" : "CloudWatchLogs",
      "Effect" : "Allow",
      "Action" : [
        "logs:CreateLogGroup",
        "logs:DescribeLogGroups"
      ],
      "Resource" : [
        "arn:aws:logs:*:*:log-group:/aws/iotmanagedintegrations/*"
      ]
    },
    {
      "Sid" : "CloudWatchStreams",
      "Effect" : "Allow",
      "Action" : [
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams"
      ],
      "Resource" : [
        "arn:aws:logs:*:*:log-group:/aws/iotmanagedintegrations/*:log-stream:*"
      ]
    },
    {
      "Sid" : "CloudWatchMetrics",
      "Effect" : "Allow",
      "Action" : [
        "cloudwatch:PutMetricData"
      ],
      "Resource" : "*",
      "Condition" : {
        "StringEquals" : {
          "cloudwatch:namespace" : [
```

```
            "AWS/IoTManagedIntegrations",
            "AWS/Usage"
          ]
        }
      }
    }
  ]
}
```

Sie müssen Berechtigungen konfigurieren, damit eine Benutzer, Gruppen oder Rollen eine serviceverknüpfte Rolle erstellen, bearbeiten oder löschen können. Weitere Informationen finden Sie unter serviceverknüpfte Rollenberechtigung im IAM-Benutzerhandbuch.

Erstellen einer dienstbezogenen Rolle für verwaltete Integrationen

Sie müssen eine serviceverknüpfte Rolle nicht manuell erstellen. Wenn Sie einen Ereignistyp wie das Aufrufen der `RegisterCustomEndpoint` API-Befehle `PutRuntimeLogConfigurationCreateEventLogConfiguration`, oder in der AWS API auslösen AWS-Managementkonsole, erstellt Managed Integrations die serviceverknüpfte Rolle für Sie. AWS CLI Weitere Informationen zu`PutRuntimeLogConfiguration`,`CreateEventLogConfiguration`, oder finden Sie unter `RegisterCustomEndpoint` PutRuntimeLogConfigurationCreateEventLogConfiguration, oder. RegisterCustomEndpoint

Wenn Sie diese serviceverknüpfte Rolle löschen und sie dann erneut erstellen müssen, können Sie dasselbe Verfahren anwenden, um die Rolle in Ihrem Konto neu anzulegen. Wenn Sie einen Ereignistyp auslösen`PutRuntimeLogConfiguration`, z. B. das Aufrufen der `RegisterCustomEndpoint` API-Befehle`CreateEventLogConfiguration`, oder, erstellt Managed Integrations die dienstbezogene Rolle erneut für Sie. Alternativ können Sie den AWS Kundensupport über die kontaktieren AWS Support Center Console. Weitere Informationen zu AWS Support-Plänen finden Sie unter AWS-Supportpläne vergleichen.

Sie können die IAM-Konsole auch verwenden, um eine serviceverknüpfte Rolle mit dem Anwendungsfall IoT ManagedIntegrations — Managed Role zu erstellen. Erstellen Sie in der AWS CLI oder der AWS API eine serviceverknüpfte Rolle mit dem `iotmanagedintegrations.amazonaws.com` Dienstnamen. Weitere Informationen finden Sie unter Erstellen einer serviceverknüpften Rolle im IAM-Benutzerhandbuch. Wenn Sie diese serviceverknüpfte Rolle löschen, können Sie mit demselben Verfahren die Rolle erneut erstellen.

Bearbeiten einer dienstbezogenen Rolle für verwaltete Integrationen

Bei verwalteten Integrationen können Sie die serviceverknüpfte Rolle „ AWSServiceRoleForIoTManagedIntegrations“ nicht bearbeiten. Da möglicherweise verschiedene Entitäten auf die Rolle verweisen, kann der Rollenname nach dem Erstellen einer serviceverknüpften Rolle nicht mehr geändert werden. Sie können jedoch die Beschreibung der Rolle mit IAM bearbeiten. Weitere Informationen finden Sie unter Bearbeiten einer serviceverknüpften Rolle im IAM-Benutzerhandbuch.

Löschen einer serviceverknüpften Rolle für verwaltete Integrationen

Wenn Sie ein Feature oder einen Dienst, die bzw. der eine serviceverknüpften Rolle erfordert, nicht mehr benötigen, sollten Sie diese Rolle löschen. Auf diese Weise haben Sie keine ungenutzte juristische Stelle, die nicht aktiv überwacht oder verwaltet wird. Sie müssen jedoch die Ressourcen für Ihre serviceverknüpften Rolle zunächst bereinigen, bevor Sie sie manuell löschen können.

> **Note**
>
> Wenn verwaltete Integrationen die Rolle verwenden, wenn Sie versuchen, die Ressourcen zu löschen, schlägt das Löschen möglicherweise fehl. Wenn dies passiert, warten Sie einige Minuten und versuchen Sie es erneut.

So löschen Sie die serviceverknüpfte Rolle mit IAM

Verwenden Sie die IAM-Konsole, die oder die AWS API AWS CLI, um die mit dem AWSService RoleForIo TManaged Integrationsdienst verknüpfte Rolle zu löschen. Weitere Informationen finden Sie unter Löschen einer serviceverknüpften Rolle im IAM-Benutzerhandbuch.

Unterstützte Regionen für serviceverknüpfte Rollen für verwaltete Integrationen

Verwaltete Integrationen für die AWS IoT Geräteverwaltung unterstützen die Verwendung von dienstbezogenen Rollen in allen Regionen, in denen der Dienst verfügbar ist. Weitere Informationen finden Sie unter AWS -Regionen und Endpunkte.

Wird AWS Secrets Manager zum Datenschutz für C2C-Workflows verwendet

AWS Secrets Manager ist ein geheimer Speicherdienst, mit dem Sie Datenbankanmeldedaten, API-Schlüssel und andere geheime Informationen schützen können. Dann können Sie in Ihrem Code hartcodierte Anmeldeinformationen durch einen API-Aufruf an Secrets Manager ersetzen. Auf diese Weise wird sichergestellt, dass das Geheimnis nicht von jemandem kompromittiert werden kann, der Ihren Code untersucht, da das Geheimnis nicht vorhanden ist. Eine Übersicht finden Sie im AWS Secrets Manager -Benutzerhandbuch.

Secrets Manager verschlüsselt Geheimnisse mithilfe von AWS Key Management Service Schlüsseln. Weitere Informationen finden Sie unter Verschlüsseln und Entschlüsseln von Geheimnissen in AWS Key Management Service.

Verwaltete Integrationen für AWS IoT Device Management Integrationen mit, AWS Secrets Manager sodass Sie Ihre Daten in Secrets Manager speichern und die geheime ID in Ihren Konfigurationen verwenden können.

Wie verwaltete Integrationen Geheimnisse verwenden

Open Authorization (OAuth) ist ein offener Standard für die delegierte Zugriffsautorisierung, der es Benutzern ermöglicht, Websites oder Anwendungen Zugriff auf ihre Informationen auf anderen Websites zu gewähren, ohne ihre Passwörter weiterzugeben. Es ist eine sichere Methode für Anwendungen von Drittanbietern, im Namen des Benutzers auf Benutzerdaten zuzugreifen, was eine sicherere Alternative zur Weitergabe von Passwörtern darstellt.

Bei OAuth einer Client-ID und einem geheimen Client-Schlüssel handelt es sich um Anmeldeinformationen, die eine Client-Anwendung identifizieren und authentifizieren, wenn sie ein Zugriffstoken anfordert.

Verwaltete Integrationen für AWS IoT Device Management Benutzer OAuth zur Kommunikation mit Kunden, die die C2C-Workflows verwenden. Kunden müssen die Client-ID und das Client-Geheimnis angeben, um kommunizieren zu können. Kunden mit verwalteten Integrationen speichern eine Client-ID und einen geheimen Kundenschlüssel in ihren AWS Konten, und verwaltete Integrationen lesen die Client-ID und das geheime Kundengeheimnis in unserem Kundenkonto.

Wie erstelle ich ein Geheimnis

Um ein Geheimnis zu erstellen, folgen Sie den Schritten unter Erstellen eines AWS Secrets Manager Geheimnisses im AWS Secrets Manager Benutzerhandbuch.

Sie müssen Ihr Geheimnis mit einem vom Kunden verwalteten AWS KMS Schlüssel für verwaltete Integrationen erstellen, um den geheimen Wert lesen zu können. Weitere Informationen finden Sie im AWS Secrets Manager Benutzerhandbuch unter Berechtigungen für den AWS KMS Schlüssel.

Sie müssen auch die IAM-Richtlinien im folgenden Abschnitt verwenden.

Gewähren Sie Zugriff für verwaltete Integrationen, AWS IoT Device Management um das Geheimnis abzurufen

Damit verwaltete Integrationen den geheimen Wert aus Secrets Manager abrufen können, nehmen Sie bei der Erstellung die folgenden Berechtigungen in die Ressourcenrichtlinie für das Geheimnis auf.

JSON

```
{
  "Version":"2012-10-17",
  "Statement" : [ {
    "Effect" : "Allow",
    "Principal" : {
      "Service" : "iotmanagedintegrations.amazonaws.com"
    },
    "Action" : [ "secretsmanager:GetSecretValue" ],
    "Resource" : "*" ,
    "Condition": {
        "StringEquals": {
          "aws:SourceArn": "arn:aws:iotmanagedintegrations:AWS Region:account-
id:account-association:account-association-id"

        }
      }
  } ]
}
```

Fügen Sie der Richtlinie für Ihren vom Kunden verwalteten AWS KMS Schlüssel die folgende Erklärung hinzu.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [{
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt",
                "kms:DescribeKey"
            ],
            "Principal": {
                "Service": [
                    "iotmanagedintegrations.amazonaws.com"
                ]
            },
            "Resource": [
            "arn:aws:kms:us-east-1:123456789012:key/*"
            ]
        }

    ]
}
```

Konformitätsprüfung für verwaltete Integrationen

Informationen darüber, ob AWS-Service ein AWS-Services in den Geltungsbereich bestimmter Compliance-Programme fällt, finden Sie unter Umfang nach Compliance-Programm AWS-Services unter . Wählen Sie dort das Compliance-Programm aus, an dem Sie interessiert sind. Allgemeine Informationen finden Sie unter AWS Compliance-Programme AWS .

Sie können Prüfberichte von Drittanbietern unter herunterladen AWS Artifact. Weitere Informationen finden Sie unter Berichte herunterladen unter .

Ihre Verantwortung für die Einhaltung der Vorschriften bei der Nutzung AWS-Services hängt von der Vertraulichkeit Ihrer Daten, den Compliance-Zielen Ihres Unternehmens und den geltenden Gesetzen und Vorschriften ab. Weitere Informationen zu Ihrer Verantwortung für die Einhaltung der Vorschriften bei der Nutzung AWS-Services finden Sie in der AWS Sicherheitsdokumentation.

Verwenden Sie verwaltete Integrationen mit VPC-Endpunkten mit Schnittstellen

Sie können eine private Verbindung zwischen Ihrer Amazon VPC und AWS IoT verwalteten Integrationen herstellen, indem Sie einen Amazon VPC-Schnittstellenendpunkt erstellen. Schnittstellenendpunkte werden von einer Technologie unterstützt AWS PrivateLink, die es Ihnen ermöglicht, über private IP-Adressen privat auf Dienste zuzugreifen. AWS PrivateLink beschränkt den gesamten Netzwerkverkehr zwischen Ihrer VPC und IoT Managed Integrations auf das Amazon-Netzwerk. Sie benötigen kein Internet-Gateway, kein NAT-Gerät oder keine VPN-Verbindung.

Sie müssen es nicht verwenden AWS PrivateLink, es wird jedoch empfohlen. Weitere Informationen zu AWS PrivateLink VPC-Endpunkten finden Sie AWS PrivateLink im Handbuch unter Zugreifen auf AWS Dienste über.AWS PrivateLink

Themen

Überlegungen zu VPC-Endpunkten mit AWS IoT verwalteten Integrationen

Bevor Sie einen VPC-Schnittstellen-Endpunkt für AWS IoT verwaltete Integrationen einrichten, lesen Sie sich die Eigenschaften und Einschränkungen der Schnittstellenendpunkte im AWS PrivateLink Handbuch durch.

AWS IoT Managed Integrations unterstützt Aufrufe aller API-Aktionen von Ihrer VPC aus über VPC-Schnittstellen-Endpunkte.

Unterstützte Endpunkte

AWS IoT Managed Integrations unterstützt VPC-Endpunkte für die folgenden Serviceschnittstellen:

- API für die Steuerungsebene: `com.amazonaws.region.iotmanagedintegrations.api`

Endpunkte werden nicht unterstützt

Die folgenden Endpunkte für AWS IoT verwaltete Integrationen unterstützen keine VPC-Endpunkte:

- MQTT-Endpunkte: MQTT-Geräte werden in der Regel in Endbenutzerumgebungen und nicht innerhalb von Umgebungen eingesetzt, sodass eine Integration nicht erforderlich ist. AWS VPCs AWS PrivateLink
- OAuth Callback-Endpunkte: Viele Plattformen von Drittanbietern funktionieren nicht innerhalb der AWS Infrastruktur, wodurch die Vorteile der Flow-Unterstützung eingeschränkt werden. AWS PrivateLink OAuth

Verfügbarkeit

AWS IoT VPC-Endpunkte für verwaltete Integrationen sind in den folgenden Regionen verfügbar: AWS

- Kanada (Zentral) – ca-central-1
- Europa (Irland) – eu-west-1

Weitere Regionen werden unterstützt, sobald die Verfügbarkeit von AWS IoT Managed Integrations erweitert wird.

Dual-Stack-Unterstützung

AWS IoT Verwaltete Integrationen VPC-Endpunkte unterstützen sowohl IPv4 den Datenverkehr als auch. IPv6 Sie können VPC-Endpoints mit den folgenden IP-Adresstypen erstellen:

- IPv4: Weist IPv4 Adressen den Endpunkt-Netzwerkschnittstellen zu
- IPv6: Weist IPv6 Adressen den Endpunkt-Netzwerkschnittstellen zu (erfordert nur Subnetze IPv6)
- Dualstack: Weist Endpunkt-Netzwerkschnittstellen sowohl als auch Adressen IPv4 zu IPv6

Erstellen eines VPC-Schnittstellen-Endpunkts für AWS IoT verwaltete Integrationen

Sie können einen VPC-Endpunkt für den AWS IoT Managed Integrations Service entweder mit der Amazon VPC-Konsole oder der AWS CLI (AWS CLI) erstellen.

So erstellen Sie einen VPC-Schnittstellen-Endpunkt für AWS IoT verwaltete Integrationen (Konsole)

1. Öffnen Sie die Amazon VPC Console unter Amazon VPC Console.
2. Wählen Sie im Navigationsbereich Endpunkte aus.
3. Wählen Sie Endpunkt erstellen aus.
4. Wählen Sie bei Service category (Servicekategorie) die Option AWS services (-Services) aus.
5. Wählen Sie unter Servicename den Servicenamen aus, der Ihrer AWS Region entspricht. Zum Beispiel:
   - `com.amazonaws.ca-central-1.iotmanagedintegrations.api`
   - `com.amazonaws.eu-west-1.iotmanagedintegrations.api`
6. Wählen Sie für VPC die VPC aus, von der aus Sie auf AWS IoT verwaltete Integrationen zugreifen möchten.
7. Für Zusätzliche Einstellungen ist standardmäßig die Option DNS-Name aktivieren ausgewählt. Wir empfehlen, diese Einstellung beizubehalten. Dadurch wird sichergestellt, dass Anfragen an die öffentlichen Service-Endpunkte von AWS IoT Managed Integrations an Ihren Amazon VPC-Endpunkt weitergeleitet werden.
8. Wählen Sie unter Subnetze die Subnetze aus, in denen Endpunkt-Netzwerkschnittstellen erstellt werden sollen. Sie können ein Subnetz pro Availability Zone auswählen.
9. Wählen Sie für IP address type (IP-Adressentyp) eine der folgenden Optionen aus:
   - IPv4: Weisen Sie den Netzwerkschnittstellen der Endpunkte IPv4 Adressen zu
   - IPv6: Weist den Endpunkt-Netzwerkschnittstellen IPv6 Adressen zu (wird nur unterstützt, wenn alle ausgewählten Subnetze nur IPv6 -only sind)
   - Dualstack: Weisen Sie den Endpunkt-Netzwerkschnittstellen IPv4 sowohl als auch IPv6 Adressen zu
10. Wählen Sie für Sicherheitsgruppen die Sicherheitsgruppen aus, die den Security groups (Endpunkt-Netzwerkschnittstellen) zugeordnet werden sollen. Die Sicherheitsgruppenregeln müssen die Kommunikation zwischen der Netzwerkschnittstelle des Endpunkts und den Ressourcen in Ihrer VPC ermöglichen, die mit dem Dienst kommunizieren.
11. Wählen Sie für Policy die Option Vollzugriff aus, um alle Operationen aller Principals auf allen Ressourcen über den Schnittstellenendpunkt zuzulassen. Um den Zugriff einzuschränken, wählen Sie Benutzerdefiniert und geben Sie eine Richtlinie an.

12. (Optional) Um ein Tag hinzuzufügen, wählen Sie Add new tag (Neuen Tag hinzufügen) aus und geben Sie den Schlüssel und den Wert für den Tag ein.
13. Wählen Sie Endpunkt erstellen aus.

So erstellen Sie einen VPC-Schnittstellen-Endpunkt für IoT Managed Integrations (AWS CLI)

Verwenden Sie den create-vpc-endpointBefehl und geben Sie die VPC-ID, den VPC-Endpunkttyp (Schnittstelle), den Dienstnamen, die Subnetze, die den Endpunkt verwenden, und die Sicherheitsgruppen an, die den Endpunkt-Netzwerkschnittstellen zugeordnet werden sollen.

```
aws ec2 create-vpc-endpoint \
  --vpc-id vpc-12345678 \
  --route-table-ids rtb-12345678 \
  --service-name com.amazonaws.ca-central-1.iotmanagedintegrations.api \
  --vpc-endpoint-type Interface \
  --subnet-ids subnet-12345678 subnet-87654321 \
  --security-group-ids sg-12345678
```

Testen Sie Ihren VPC-Endpunkt

Nachdem Sie Ihren VPC-Endpunkt erstellt haben, können Sie die Verbindung testen, indem Sie API-Aufrufe an AWS IoT verwaltete Integrationen von einer EC2 Instanz in Ihrer VPC aus tätigen.

Voraussetzungen

- Eine EC2 Instance in einem privaten Subnetz innerhalb Ihrer VPC
- Entsprechende IAM-Berechtigungen für AWS IoT verwaltete Integrationsvorgänge
- Sicherheitsgruppenregeln, die HTTPS-Verkehr (Port 443) zum VPC-Endpunkt zulassen

Testen der -Verbindung

1. Connect zu Ihrer EC2 Amazon-Instance im privaten Subnetz her.
2. Überprüfen Sie die DNS-Auflösung für den privaten DNS-Namen:

```
dig api.iotmanagedintegrations.region.api.aws
```

3. Testen Sie die HTTPS-Konnektivität:

```
curl -v https://api.iotmanagedintegrations.region.api.aws
```

4. Führen Sie einen API-Aufruf für AWS IoT verwaltete Integrationen durch:

```
aws iot-managed-integrations list-destinations \
  --region region \
  --endpoint-url https://api.iotmanagedintegrations.region.api.aws
```

`region`Ersetzen Sie es durch Ihre AWS Region (z. B.`ca-central-1`).

Steuerung des Zugriffs auf Dienste über VPC-Endpunkte

Eine VPC-Endpunktrichtlinie ist eine IAM-Ressourcenrichtlinie, die Sie einem Schnittstellen-VPC-Endpunkt zuordnen, wenn Sie den Endpunkt erstellen oder ändern. Wenn Sie einem Endpunkt beim Erstellen keine Richtlinie zuordnen, wird ihm eine Standardrichtlinie mit Vollzugriff auf den Service zugeordnet. IAM-Benutzerrichtlinien oder servicespezifische Richtlinien werden von einer Endpunktrichtlinie nicht überschrieben oder ersetzt. Endpunktrichtlinien steuern unabhängig vom Endpunkt den Zugriff auf den angegebenen Service.

Endpunktrichtlinien müssen im JSON-Format erstellt werden. Weitere Informationen finden Sie unter Steuerung des Zugriffs auf Services mit VPC-Endpunkten im Amazon VPC Benutzerhandbuch.

Beispiel: VPC-Endpunktrichtlinie für AWS IoT verwaltete Integrationsaktionen

Im Folgenden finden Sie ein Beispiel für eine Endpunktrichtlinie für AWS IoT verwaltete Integrationen. Diese Richtlinie ermöglicht Benutzern, die über den VPC-Endpunkt eine Verbindung zu AWS IoT verwalteten Integrationen herstellen, Zugriff auf Ziele, verweigert jedoch den Zugriff auf Anmeldeinformationsschließfächer.

```
{
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": [
                "iotmanagedintegrations:ListDestinations",
                "iotmanagedintegrations:GetDestination",
                "iotmanagedintegrations:CreateDestination",
                "iotmanagedintegrations:UpdateDestination",
```

```
                "iotmanagedintegrations:DeleteDestination"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Deny",
            "Principal": "*",
            "Action": [
                "iotmanagedintegrations:ListCredentialLockers",
                "iotmanagedintegrations:GetCredentialLocker",
                "iotmanagedintegrations:CreateCredentialLocker",
                "iotmanagedintegrations:UpdateCredentialLocker",
                "iotmanagedintegrations:DeleteCredentialLocker"
            ],
            "Resource": "*"
        }
    ]
}
```

Beispiel: VPC-Endpunktrichtlinie, die den Zugriff auf eine bestimmte IAM-Rolle einschränkt

Die folgende VPC-Endpunktrichtlinie ermöglicht den Zugriff auf AWS IoT verwaltete Integrationen nur für IAM-Prinzipale, die die angegebene IAM-Rolle in ihrer Vertrauenskette haben. Allen anderen IAM-Prinzipalen wird der Zugriff verweigert.

```
{
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "*",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:PrincipalArn": "arn:aws:iam::123456789012:role/
IoTManagedIntegrationsVPCRole"
                }
            }
        }
    ]
}
```

Preisgestaltung

Für die Erstellung und Nutzung eines VPC-Schnittstellen-Endpunkts mit AWS IoT verwalteten Integrationen werden Ihnen Standardtarife berechnet. Weitere Informationen finden Sie unter AWS PrivateLink Preise.

Einschränkungen

- Die CreateAccountAssociationAPI ist so konzipiert, dass sie OAuth mit Cloud-Diensten von Drittanbietern funktioniert, was erfordert, dass die Anfrage das Amazon-Netzwerk verlässt. Dies ist wichtig für Kunden AWS PrivateLink , die ihren Datenverkehr innerhalb der VPC eindämmen möchten, da dieser AWS PrivateLink API-Aufruf nicht vollständig end-to-end eingedämmt werden kann.
- VPC-Endpunkte für AWS IoT verwaltete Integrationen sind in nicht verfügbar. AWS GovCloud (US) Regions

Allgemeine VPC-Endpunktbeschränkungen finden Sie unter Eigenschaften und Einschränkungen der Schnittstellenendpunkte im Amazon VPC-Benutzerhandbuch.

Connect zu verwalteten Integrationen für AWS IoT Device Management FIPS-Endgeräte her

AWS IoT bietet einen Endpunkt auf der Kontrollebene, der den Federal Information Processing Standard (FIPS) 140-2 unterstützt. FIPS-konforme Endpunkte unterscheiden sich von Standardendpunkten. AWS Um mit verwalteten Integrationen auf FIPS-konforme Weise zu interagieren, müssen Sie die unten beschriebenen Endpunkte mit Ihrem FIPS-konformen Client verwenden. AWS IoT Device Management Die AWS IoT Konsole ist nicht FIPS-konform.

In den folgenden Abschnitten wird beschrieben, wie Sie mithilfe der REST-API, eines SDK oder der auf den FIPS-kompatiblen AWS IoT Endpunkt zugreifen. AWS CLI

Endpunkte der Steuerebene

Die FIPS-konformen Endpunkte der Steuerungsebene, die die verwalteten Integrationsvorgänge unterstützen, und die zugehörigen AWS CLI Befehle sind unter FIPS-Endpunkte nach Dienst aufgeführt. Suchen Sie in FIPS Endpoints by Service nach dem AWS IoT Device Management - Managed Integrations Service und suchen Sie den Endpunkt für Ihren. AWS-Region

Um den FIPS-konformen Endpunkt zu verwenden, wenn Sie auf die verwalteten Integrationsvorgänge zugreifen, verwenden Sie das AWS SDK oder die REST-API mit dem Endpunkt, der für Sie geeignet ist. AWS-Region

Um den FIPS-kompatiblen Endpunkt zu verwenden, wenn Sie CLI-Befehle für verwaltete Integrationen ausführen, fügen Sie dem Befehl den --endpoint Parameter mit dem entsprechenden Endpunkt für Sie AWS-Region hinzu.

Überwachung verwalteter Integrationen

Die Überwachung ist ein wichtiger Bestandteil der Aufrechterhaltung der Zuverlässigkeit, Verfügbarkeit und Leistung von Managed Integrations und Ihren anderen AWS-Lösungen. AWS bietet die folgenden Überwachungstools, um verwaltete Integrationen zu überwachen, zu melden, wenn etwas nicht stimmt, und gegebenenfalls automatische Maßnahmen zu ergreifen:

- AWS CloudTrailerfasst API-Aufrufe und zugehörige Ereignisse, die von oder im Namen Ihres AWS Kontos getätigt wurden, und übermittelt die Protokolldateien an einen von Ihnen angegebenen Amazon S3 S3-Bucket. Sie können feststellen, welche Benutzer und Konten angerufen wurden AWS, von welcher Quell-IP-Adresse aus die Anrufe getätigt wurden und wann die Aufrufe erfolgten. Weitere Informationen finden Sie im AWS CloudTrail -Benutzerhandbuch.

Protokollierung von API-Aufrufen für verwaltete Integrationen mithilfe AWS CloudTrail

Managed Integrations ist in einen Service integriert AWS CloudTrail, der eine Aufzeichnung der von einem Benutzer, einer Rolle oder einem ausgeführten Aktionen bereitstellt. AWS-Service CloudTrail erfasst alle API-Aufrufe für verwaltete Integrationen als Ereignisse. Zu den erfassten Aufrufen gehören Aufrufe von der Konsole für verwaltete Integrationen und Codeaufrufen für die API-Operationen der verwalteten Integrationen. Anhand der von gesammelten Informationen können Sie die Anfrage CloudTrail, die an verwaltete Integrationen gestellt wurde, die IP-Adresse, von der aus die Anfrage gestellt wurde, wann sie gestellt wurde, und weitere Details ermitteln.

Jeder Ereignis- oder Protokolleintrag enthält Informationen zu dem Benutzer, der die Anforderung generiert hat. Die Identitätsinformationen unterstützen Sie bei der Ermittlung der folgenden Punkte:

- Ob die Anfrage mit Anmeldeinformationen des Root-Benutzers oder des Benutzers gestellt wurde.
- Die Anforderung wurde im Namen eines IAM-Identity-Center-Benutzers erstellt.
- Gibt an, ob die Anforderung mit temporären Sicherheitsanmeldeinformationen für eine Rolle oder einen Verbundbenutzer gesendet wurde.
- Ob die Anforderung aus einem anderen AWS-Service gesendet wurde.

CloudTrail ist in Ihrem aktiv AWS-Konto , wenn Sie das Konto erstellen, und Sie haben automatisch Zugriff auf den CloudTrail Eventverlauf. Der CloudTrail Ereignisverlauf bietet eine einsehbare,

durchsuchbare, herunterladbare und unveränderliche Aufzeichnung der aufgezeichneten Verwaltungsereignisse der letzten 90 Tage in einem. AWS-Region Weitere Informationen finden Sie im AWS CloudTrail Benutzerhandbuch unter Arbeiten mit dem CloudTrail Ereignisverlauf. Für die Anzeige des Eventverlaufs CloudTrail fallen keine Gebühren an.

Für eine fortlaufende Aufzeichnung der Ereignisse in AWS-Konto den letzten 90 Tagen erstellen Sie einen Trail- oder CloudTrailLake-Event-Datenspeicher.

CloudTrail Pfade

Ein Trail ermöglicht CloudTrail die Übermittlung von Protokolldateien an einen Amazon S3 S3-Bucket. Alle mit dem erstellten Pfade AWS-Managementkonsole sind regionsübergreifend. Sie können mithilfe von AWS CLI einen Einzel-Region- oder einen Multi-Region-Trail erstellen. Es wird empfohlen, einen Trail mit mehreren Regionen zu erstellen, da Sie alle Aktivitäten AWS-Regionen in Ihrem Konto erfassen. Wenn Sie einen Einzel-Region-Trail erstellen, können Sie nur die Ereignisse anzeigen, die im AWS-Region des Trails protokolliert wurden. Weitere Informationen zu Trails finden Sie unter Erstellen eines Trails für Ihr AWS-Konto und Erstellen eines Trails für eine Organisation im AWS CloudTrail -Benutzerhandbuch.

Sie können eine Kopie Ihrer laufenden Verwaltungsereignisse kostenlos an Ihren Amazon S3 S3-Bucket senden, CloudTrail indem Sie einen Trail erstellen. Es fallen jedoch Amazon S3 S3-Speichergebühren an. Weitere Informationen zur CloudTrail Preisgestaltung finden Sie unter AWS CloudTrail Preise. Informationen zu Amazon-S3-Preisen finden Sie unter Amazon S3 – Preise.

CloudTrail Datenspeicher für Ereignisse in Lake

CloudTrail Mit Lake können Sie SQL-basierte Abfragen für Ihre Ereignisse ausführen. CloudTrail Lake konvertiert bestehende Ereignisse im zeilenbasierten JSON-Format in das Apache ORC-Format. ORC ist ein spaltenförmiges Speicherformat, das für den schnellen Abruf von Daten optimiert ist. Die Ereignisse werden in Ereignisdatenspeichern zusammengefasst, bei denen es sich um unveränderliche Sammlungen von Ereignissen handelt, die auf Kriterien basieren, die Sie mit Hilfe von erweiterten Ereignisselektoren auswählen. Die Selektoren, die Sie auf einen Ereignisdatenspeicher anwenden, steuern, welche Ereignisse bestehen bleiben und für Sie zur Abfrage verfügbar sind. Weitere Informationen zu CloudTrail Lake finden Sie unter Arbeiten mit AWS CloudTrail Lake im AWS CloudTrail Benutzerhandbuch.

CloudTrail Für das Speichern und Abfragen von Ereignisdaten in Lake fallen Kosten an. Beim Erstellen eines Ereignisdatenspeichers wählen Sie die Preisoption aus, die für den Ereignisdatenspeicher genutzt werden soll. Die Preisoption bestimmt die Kosten für die Erfassung und Speicherung von Ereignissen sowie die standardmäßige und maximale Aufbewahrungsdauer

für den Ereignisdatenspeicher. Weitere Informationen zur Preisgestaltung finden Sie unter CloudTrail AWS CloudTrail Preisgestaltung.

Management-Ereignisse in CloudTrail

Verwaltungsereignisse enthalten Informationen zu Verwaltungsvorgängen, die an Ressourcen in Ihrem ausgeführt werden AWS-Konto. Sie werden auch als Vorgänge auf Steuerebene bezeichnet. In der Standardeinstellung werden Verwaltungsereignisse CloudTrail protokolliert.

Verwaltete Integrationen protokollieren die folgenden Operationen auf der Kontrollebene verwalteter Integrationen CloudTrail als Verwaltungsereignisse.

- `CreateCloudConnector`
- `UpdateCloudConnector`
- `GetCloudConnector`
- `DeleteCloudConnector`
- `ListCloudConnectors`
- `CreateConnectorDestination`
- `UpdateConnectorDestination`
- `GetConnectorDestination`
- `DeleteConnectorDestination`
- `ListConnectorDestinations`
- `CreateAccountAssociation`
- `UpdateAccountAssociation`
- `GetAccountAssociation`
- `DeleteAccountAssociation`
- `ListAccountAssociations`
- `StartAccountAssociationRefresh`
- `ListManagedThingAccountAssociations`
- `RegisterAccountAssociation`
- `DeregisterAccountAssociation`
- `SendConnectorEvent`
- `ListDeviceDiscoveries`

- `ListDiscoveredDevices`

Beispiele für Ereignisse

Ein Ereignis stellt eine einzelne Anfrage aus einer beliebigen Quelle dar und umfasst Informationen über den angeforderten API-Vorgang, Datum und Uhrzeit des Vorgangs, Anforderungsparameter usw. CloudTrail Protokolldateien sind kein geordneter Stack-Trace der öffentlichen API-Aufrufe, sodass Ereignisse nicht in einer bestimmten Reihenfolge angezeigt werden.

Das folgende Beispiel zeigt ein CloudTrail Ereignis, das einen erfolgreichen `CreateCloudConnector` API-Vorgang demonstriert.

Erfolgreiches CloudTrail Ereignis mit dem **`CreateCloudConnector`** API-Vorgang.


```
{
    "eventVersion": "1.09",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "EXAMPLE",
        "arn": "arn:aws:sts::111122223333:assumed-role/Admin/EXAMPLE",
        "accountId": "111122223333",
        "accessKeyId": "EXAMPLEKYSBQSCGRIC",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROAZOZQFKYSFZVB2J2GN",
                "arn": "arn:aws:iam::111122223333:role/Admin",
                "accountId": "111122223333",
                "userName": "Admin"
            },
            "attributes": {
                "creationDate": "2025-06-05T18:26:16Z",
                "mfaAuthenticated": "false"
            }
        }
    },
    "eventTime": "2025-06-05T18:30:40Z",
    "eventSource": "iotmanagedintegrations.amazonaws.com",
    "eventName": "CreateCloudConnector",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "PostmanRuntime/7.44.0",
```

```
    "requestParameters": {
        "EndpointType": "LAMBDA",
        "Description": "Manual testing for C2C CT Validation",
        "ClientToken": "abc7460",
        "EndpointConfig": {
            "lambda": {
                "arn": "arn:aws:lambda:us-
east-1:111122223333:function:LightweightMockConnector7460"
            }
        },
        "Name": "EdenManualTestCloudConnector"
    },
    "responseElements": {
        "X-Frame-Options": "DENY",
        "Access-Control-Expose-Headers": "Content-Length,Content-Type,X-Amzn-
Errortype,X-Amzn-Requestid",
        "Strict-Transport-Security": "max-age:47304000; includeSubDomains",
        "Cache-Control": "no-store, no-cache",
        "X-Content-Type-Options": "nosniff",
        "Content-Security-Policy": "upgrade-insecure-requests; default-src 'none';
 object-src 'none'; frame-ancestors 'none'; base-uri 'none'",
        "Pragma": "no-cache",
        "Id": "f7e633e719404c4a933596b4d0cc276e",
        "Arn": "arn:aws:iotmanagedintegrations:us-east-1:111122223333:cloud-connector/
EXAMPLE404c4a933596b4d0cc276e"
    },
    "requestID": "c0071fd1-b8e0-400a-bcc0-EXAMPLE9e4",
    "eventID": "95b318ea-2f63-4183-9c22-EXAMPLE3e",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```


Das folgende Beispiel zeigt ein CloudTrail Ereignis, das einen erfolgreichen `ListDiscoveredDevices` API-Vorgang demonstriert.

Erfolgreiches CloudTrail Ereignis mit dem **`ListDiscoveredDevices`** API-Vorgang.


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

```
        "type": "AssumedRole",
        "principalId": "EZAMPLE",
        "arn": "arn:aws:sts::444455556666:assumed-role/Admin/EXAMPLE",
        "accountId": "444455556666",
        "accessKeyId": "EXAMPLERJ26PYMH",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "EXAMPLE",
                "arn": "arn:aws:iam::444455556666:role/Admin",
                "accountId": "444455556666",
                "userName": "Admin"
            },
            "attributes": {
                "creationDate": "2025-06-10T23:37:31Z",
                "mfaAuthenticated": "false"
            }
        }
    },
    "eventTime": "2025-06-10T23:38:07Z",
    "eventSource": "iotmanagedintegrations.amazonaws.com",
    "eventName": "ListDiscoveredDevices",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "EXAMPLE-runtime/2.4.0",
    "requestParameters": {
        "Identifier": "EXAMPLE4f268483a17d8060f014"
    },
    "responseElements": null,
    "requestID": "27ae1f61-e2e6-43e4-bf17-EXAMPLEa568",
    "eventID": "34734e81-76a8-49a4-9641-EXAMPLE28ed",
    "readOnly": true,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "444455556666",
    "eventCategory": "Management"
}
```


Informationen zu CloudTrail Datensatzinhalten finden Sie im AWS CloudTrail Benutzerhandbuch unter CloudTrailDatensatzinhalte.

Dokumentenverlauf für den Entwicklerhandbuch für verwaltete Integrationen

In der folgenden Tabelle werden die Dokumentationsversionen für verwaltete Integrationen beschrieben.

Änderung	Beschreibung	Datum
Version für allgemeine Verfügbarkeit	Version für allgemeine Verfügbarkeit des Managed Integrations Developer Guide	25. Juni 2025
Erste Vorschau-Version	Erste Vorschauversion des Managed Integrations Developer Guide	03. März 2025

Die vorliegende Übersetzung wurde maschinell erstellt. Im Falle eines Konflikts oder eines Widerspruchs zwischen dieser übersetzten Fassung und der englischen Fassung (einschließlich infolge von Verzögerungen bei der Übersetzung) ist die englische Fassung maßgeblich.