aws

Guia do desenvolvedor

Integrações gerenciadas para AWS IoT Device Management

Integrações gerenciadas para AWS IoT Device Management: Guia do desenvolvedor

Table of Contents

Para que servem as integrações gerenciadas? AWS IoT Device Management

Integrações gerenciadas AWS IoT Device Management ajudam os fornecedores de soluções de IoT a unificar o controle e o gerenciamento de dispositivos de IoT de centenas de fabricantes. Você pode usar integrações gerenciadas para automatizar fluxos de trabalho de configuração de dispositivos e oferecer suporte à interoperabilidade em vários dispositivos, independentemente do fornecedor do dispositivo ou do protocolo de conectividade. Com integrações gerenciadas, você pode usar uma única interface de usuário e um conjunto de APIs para controlar, gerenciar e operar uma variedade de dispositivos.

Tópicos

Regiões do compatíveis

As integrações gerenciadas do AWS IoT Device Management são suportadas nas seguintes regiões:

- Canadá (Central)
- Europa (Irlanda)

Você é um usuário iniciante de integrações gerenciadas?

Se você é um usuário iniciante de integrações gerenciadas, recomendamos que comece lendo as seguintes seções:

- Configurar integrações gerenciadas
- Comece com integrações gerenciadas para AWS IoT Device Management

Visão geral das integrações gerenciadas

A imagem a seguir fornece uma visão geral de alto nível das integrações gerenciadas

Terminologia de integrações gerenciadas

Nas integrações gerenciadas, há muitos conceitos e termos essenciais a serem entendidos para gerenciar suas próprias implementações de dispositivos. As seções a seguir descrevem os principais conceitos e termos para fornecer uma melhor compreensão das integrações gerenciadas.

Terminologia geral de integrações gerenciadas

Um conceito importante a ser entendido para integrações gerenciadas é uma coisa gerenciada em comparação com AWS IoT Core outra coisa.

- AWS IoT Core coisa: Uma AWS IoT Core coisa é uma AWS IoT Core construção que fornece a representação digital. Espera-se que os desenvolvedores gerenciem políticas, armazenamento de dados, regras, ações, tópicos de MQTT e entrega do estado do dispositivo ao armazenamento de dados. Para obter mais informações sobre o que é uma AWS IoT Core coisa, consulte Gerenciando dispositivos com AWS IoT.
- Integrações gerenciadas Managed Thing: Com uma coisa gerenciada, fornecemos uma abstração para simplificar as interações com dispositivos e não exigimos que o desenvolvedor crie itens como regras, ações, tópicos e políticas do MQTT.

Cloud-to-cloud terminologia

Os dispositivos físicos que se integram às integrações gerenciadas podem ser originários de um provedor de nuvem terceirizado. Para integrar esses dispositivos às integrações gerenciadas e se comunicar com o provedor de nuvem terceirizado, a terminologia a seguir abrange alguns dos principais conceitos que sustentam esses fluxos de trabalho:

- Cloud-to-cloud Conector (C2C): um conector C2C estabelece uma conexão entre as integrações gerenciadas e o provedor de nuvem terceirizado.
- Provedor de nuvem terceirizado: para dispositivos fabricados e gerenciados fora das integrações gerenciadas, um provedor de nuvem terceirizado permite o controle desses dispositivos para o

usuário final e as integrações gerenciadas se comunicam com o provedor de nuvem terceirizado para vários fluxos de trabalho, como comandos de dispositivos.

Terminologia do modelo de dados

As integrações gerenciadas usam modelos de dados para organizar os dados e a end-to-end comunicação entre seus dispositivos. A terminologia a seguir abrange alguns dos principais conceitos para entender esses dois modelos de dados:

- Dispositivo: uma entidade que representa um dispositivo físico (como uma campainha de vídeo) que tem vários nós trabalhando juntos para fornecer um conjunto completo de recursos.
- Ponto final: um endpoint encapsula um recurso independente (campainha, detecção de movimento, iluminação em uma campainha de vídeo).
- Capacidade: uma entidade que representa os componentes necessários para disponibilizar um recurso em um terminal (botão ou um recurso de luz e toque na campainha de vídeo).
- Ação: uma entidade que representa uma interação com a capacidade de um dispositivo (tocar a campainha ou ver quem está na porta).
- Evento: uma entidade que representa um evento a partir de uma capacidade de um dispositivo. Um dispositivo pode enviar um evento para denunciar um (incident/alarm, an activity from a sensor etc. (e.g. there is knock/ringna porta).
- Propriedade: Uma entidade que representa um atributo específico no estado do dispositivo (a campainha está tocando, a luz da varanda está acesa, a câmera está gravando).
- Modelo de dados: a camada de dados corresponde aos elementos de dados e verbos que ajudam a suportar a funcionalidade do aplicativo. O aplicativo opera nessas estruturas de dados quando há a intenção de interagir com o dispositivo. Para obter mais informações, consulte connectedhomeip no site. GitHub
- Esquema: um esquema é uma representação do modelo de dados no formato JSON.

Configurar integrações gerenciadas

As seções a seguir orientam você na configuração inicial do uso de integrações gerenciadas para AWS IoT Device Management.

Tópicos

Inscreva-se para um Conta da AWS

Se você não tiver um Conta da AWS, conclua as etapas a seguir para criar um.

Para se inscrever em um Conta da AWS

1. Abra a https://portal.aws.amazon.com/billing/inscrição.
2. Siga as instruções online.

   Parte do procedimento de inscrição envolve receber uma chamada telefônica ou uma mensagem de texto e inserir um código de verificação pelo teclado do telefone.

   Quando você se inscreve em um Conta da AWS, um Usuário raiz da conta da AWSé criado. O usuário-raiz tem acesso a todos os Serviços da AWS e recursos na conta. Como prática recomendada de segurança, atribua o acesso administrativo a um usuário e use somente o usuário-raiz para executar tarefas que exigem acesso de usuário-raiz.

AWS envia um e-mail de confirmação após a conclusão do processo de inscrição. A qualquer momento, você pode visualizar a atividade atual da sua conta e gerenciar sua conta acessando https://aws.amazon.com/e escolhendo Minha conta.

Criar um usuário com acesso administrativo

Depois de se inscrever em um Conta da AWS, proteja seu Usuário raiz da conta da AWS Centro de Identidade do AWS IAM, habilite e crie um usuário administrativo para que você não use o usuário root nas tarefas diárias.

Proteja seu Usuário raiz da conta da AWS

1. Faça login Console de gerenciamento da AWScomo proprietário da conta escolhendo Usuário raiz e inserindo seu endereço de Conta da AWS e-mail. Na próxima página, insira a senha.

   Para obter ajuda ao fazer login usando o usuário-raiz, consulte Fazer login como usuário-raiz no Guia do usuário do Início de Sessão da AWS .
2. Habilite a autenticação multifator (MFA) para o usuário-raiz.

   Para obter instruções, consulte Habilitar um dispositivo de MFA virtual para seu usuário Conta da AWS raiz (console) no Guia do usuário do IAM.

Criar um usuário com acesso administrativo

1. Habilita o Centro de Identidade do IAM.

   Para obter instruções, consulte Habilitar o Centro de Identidade do AWS IAM no Guia do usuário do Centro de Identidade do AWS IAM .
2. No Centro de Identidade do IAM, conceda o acesso administrativo a um usuário.

   Para ver um tutorial sobre como usar o Diretório do Centro de Identidade do IAM como fonte de identidade, consulte Configurar o acesso do usuário com o padrão Diretório do Centro de Identidade do IAM no Guia Centro de Identidade do AWS IAM do usuário.

Iniciar sessão como o usuário com acesso administrativo

- Para fazer login com o seu usuário do Centro de Identidade do IAM, use o URL de login enviado ao seu endereço de e-mail quando o usuário do Centro de Identidade do IAM foi criado.

  Para obter ajuda para fazer login usando um usuário do IAM Identity Center, consulte Como fazer login no portal de AWS acesso no Guia Início de Sessão da AWS do usuário.

Atribuir acesso a usuários adicionais

1. No Centro de Identidade do IAM, crie um conjunto de permissões que siga as práticas recomendadas de aplicação de permissões com privilégio mínimo.

Para obter instruções, consulte Criar um conjunto de permissões no Guia do usuário do Centro de Identidade do AWS IAM .

2. Atribua usuários a um grupo e, em seguida, atribua o acesso de autenticação única ao grupo.

   Para obter instruções, consulte Adicionar grupos no Guia do usuário do Centro de Identidade do AWS IAM .

Comece com integrações gerenciadas para AWS IoT Device Management

As seções a seguir descrevem as etapas que você precisa seguir para começar a usar integrações gerenciadas.

Tópicos

Tipos de dispositivos

As integrações gerenciadas gerenciam vários tipos de dispositivos. Cada dispositivo está dentro de uma das três categorias a seguir:

- Dispositivos com conexão direta: esse tipo de dispositivo se conecta diretamente a um endpoint de integrações gerenciadas. Normalmente, esses dispositivos são criados e gerenciados por fabricantes de dispositivos que incluem o SDK do dispositivo final de integrações gerenciadas para a conectividade direta.
- Dispositivos conectados ao hub: esses dispositivos se conectam às integrações gerenciadas por meio de um hub que executa o SDK do Hub de integrações gerenciadas, que gerencia as funções de descoberta, integração e controle de dispositivos. Os usuários finais podem integrar esses dispositivos usando o início do pressionamento do botão ou a leitura de código de barras.

  Os dois fluxos de trabalho a seguir são compatíveis com a integração de um dispositivo conectado ao hub:

  - Um botão iniciado pelo usuário final para iniciar a descoberta do dispositivo
  - Digitalização baseada em código de barras para realizar a associação do dispositivo
- Cloud-to-cloud Dispositivos (C2C): são dispositivos projetados e gerenciados por fornecedores que mantêm sua própria infraestrutura de nuvem e aplicativos móveis de marca para controle de dispositivos. Os clientes de integrações gerenciadas podem acessar um catálogo de conectores C2C pré-construídos ou criar seus próprios conectores para desenvolver soluções de IoT que funcionem com várias nuvens de fornecedores terceirizados por meio de uma interface unificada.

Quando o usuário final liga um dispositivo C2C pela primeira vez, ele deve ser provisionado com seu respectivo provedor de nuvem terceirizado para integrações gerenciadas para obter os recursos e metadados do dispositivo. Depois de concluir esse fluxo de trabalho de provisionamento, as integrações gerenciadas podem se comunicar com o dispositivo de nuvem e o provedor de nuvem terceirizado em nome do usuário final.

> **Note**
>
> Um hub não é um tipo de dispositivo específico, conforme listado acima. Seu objetivo é desempenhar o papel de controlador de dispositivos domésticos inteligentes e facilitar a conexão entre integrações gerenciadas e provedores de nuvem terceirizados. Ele pode servir tanto como um tipo de dispositivo, conforme listado acima, quanto como um hub.

Configurar chave de criptografia

A segurança é de suma importância para dados roteados entre o usuário final, integrações gerenciadas e nuvens de terceiros. Um dos métodos que oferecemos para proteger os dados do seu dispositivo é a end-to-end criptografia, utilizando uma chave de criptografia segura para rotear seus dados.

Como cliente de integrações gerenciadas, você tem as duas opções a seguir para usar chaves de criptografia:

- Use a chave de criptografia padrão gerenciada por integrações gerenciadas.
- Forneça um AWS KMS key que você criou.

Para obter mais informações sobre o serviço AWS KMS, consulte Serviço de gerenciamento de chaves (KMS)

Chamar a PutDefaultEncryptionConfigurationAPI no Guia de referência da API de integrações gerenciadas concede acesso para atualizar qual opção de chave de criptografia você deseja usar. Por padrão, as integrações gerenciadas usam a chave de criptografia gerenciada padrão das integrações gerenciadas. Você pode atualizar a configuração da chave de criptografia a qualquer momento usando a PutDefaultEncryptionConfigurationAPI.

Além disso, chamar o comando da GetDefaultEncryptionConfigurationAPI retorna informações sobre a configuração de criptografia da AWS conta na região padrão ou especificada.

Técnicas de integração

Abaixo estão listados os tipos de integração:

Integração de dispositivos com conexão direta

Consulte as etapas Provisionado para integrar um dispositivo conectado diretamente.

Integração do hub

Consulte as etapas Integre seus hubs para integrações gerenciadas para integrar o hub.

Integração de dispositivos conectados ao hub

Consulte as etapas Integre dispositivos e opere-os no hub para integrar um dispositivo conectado ao hub.

Cloud-to-cloud integração de dispositivos

Veja as etapas Use um conector C2C (Cloud-to-Cloud) para integrar um dispositivo em nuvem de um fornecedor de nuvem terceirizado às integrações gerenciadas.

Provisionamento de dispositivos

O provisionamento de dispositivos facilita o processo de integração do dispositivo, supervisiona todo o ciclo de vida do dispositivo e estabelece um repositório centralizado de informações do dispositivo que é acessível a outros aspectos das integrações gerenciadas. As integrações gerenciadas fornecem uma interface unificada para gerenciar vários tipos de dispositivos, acomodando dispositivos de clientes primários conectados diretamente por meio de um kit de desenvolvimento de software (SDK) ou dispositivos commercial-off-the-shelf (COTS) vinculados indiretamente por meio de um dispositivo hub.

Cada dispositivo, independentemente do tipo de dispositivo, em integrações gerenciadas tem um identificador global exclusivo chamado de. `managedThingId` Esse identificador é usado na integração e no gerenciamento do dispositivo durante todo o ciclo de vida do dispositivo. Ele é totalmente gerenciado por integrações gerenciadas e exclusivo para esse dispositivo específico em todas as integrações gerenciadas. Regiões da AWS Quando um dispositivo é inicialmente adicionado às integrações gerenciadas, esse identificador é criado e anexado à coisa gerenciada nas integrações gerenciadas. Uma coisa gerenciada é uma representação digital do dispositivo físico nas integrações gerenciadas para espelhar todos os metadados do dispositivo físico. Para dispositivos de terceiros, eles podem ter seu próprio identificador exclusivo separado específico para sua nuvem de terceiros, além do `managedThingId` armazenado em integrações gerenciadas que representam o dispositivo físico.

Os dispositivos que estão sendo provisionados podem ter status diferentes, dependendo do estágio do fluxo de integração em que se encontram. A lista a seguir descreve cada status de aprovisionamento:

- ATIVADO: O dispositivo foi encontrado e o comando e o controle estão disponíveis.
- DESCOBERTO: O dispositivo foi encontrado, mas o comando e o controle ainda não estão disponíveis.
- NÃO ASSOCIADO: A coisa gerenciada foi criada, mas requer que outras ações sejam descobertas. Não é acessível a partir dos controladores de integrações AWS IoT gerenciadas (hubs) Nuvem AWS ou
- PRE_ASSOCIATED: A coisa gerenciada foi criada e está pronta para ser descoberta automática quando ligada ou conectada. Não é acessível a partir dos controladores de integrações gerenciadas (hubs) Nuvem AWS ou dos controladores de integrações AWS IoT gerenciadas.
- DELETE_IN_PROGRESS: Processo de exclusão assíncrona iniciado.

- EXCLUÍDO: O dispositivo foi excluído do Nuvem AWS.
- ISOLADO: Uma coisa gerenciada previamente descoberta ou ativada que não está mais acessível. Por exemplo, um dispositivo para uma nuvem de terceiros cujas associações de conectores foram todas excluídas.

O fluxo de integração a seguir serve para provisionar seu hub com integrações gerenciadas:

Integre seus hubs para integrações gerenciadas: configure o provisionador principal e os plug-ins específicos do protocolo que funcionam juntos para lidar com a autenticação, a comunicação e a configuração do dispositivo.

Os seguintes fluxos de integração são fornecidos para provisionar seus dispositivos conectados ao hub com integrações gerenciadas:

- Configuração simples (SS): o usuário final liga o dispositivo de IoT e escaneia seu código QR usando o aplicativo do fabricante do dispositivo. O dispositivo é então inscrito na nuvem de integrações gerenciadas e se conecta ao hub de IoT.
- Configuração sem toque (ZTS): O dispositivo é pré-associado a montante na cadeia de suprimentos. Por exemplo, em vez de os usuários finais digitalizarem o código QR do dispositivo, essa etapa é concluída anteriormente para pré-vincular o dispositivo às contas dos clientes.
- Configuração guiada pelo usuário (UGS): o usuário final liga o dispositivo e segue etapas interativas para integrá-lo às integrações gerenciadas. Isso pode incluir pressionar um botão no hub de IoT, usar um aplicativo do fabricante do dispositivo ou pressionar botões no hub e no dispositivo. Você pode usar esse método se a configuração simples falhar.

> **Note**
>
> O fluxo de trabalho de provisionamento de dispositivos em integrações gerenciadas é independente dos requisitos de integração de um dispositivo. As integrações gerenciadas fornecem uma interface de usuário simplificada para integrar e gerenciar um dispositivo, independentemente do tipo de dispositivo ou do protocolo do dispositivo.

Ciclo de vida do dispositivo e do perfil do dispositivo

O gerenciamento do ciclo de vida de seus dispositivos e perfis de dispositivos garante que sua frota de dispositivos esteja segura e funcionando com eficiência.

Tópicos

Dispositivo

Durante a integração inicial, as integrações gerenciadas criam um gêmeo digital do seu dispositivo físico chamado Managed Thing. O Managed Thing tem um `managedThingID` que fornece um identificador global exclusivo para identificar o dispositivo em integrações gerenciadas em todas as regiões. O dispositivo se emparelha com o hub local durante o provisionamento para comunicação em tempo real com integrações gerenciadas ou com uma nuvem de terceiros para dispositivos de terceiros. Um dispositivo também está associado a um proprietário, conforme identificado pelo `owner` parâmetro público APIs de uma coisa gerenciada, como`GetManagedThing`. O dispositivo está vinculado ao perfil de dispositivo correspondente com base no tipo de dispositivo.

> **Note**
>
> Um dispositivo físico pode ter vários registros se for provisionado várias vezes em clientes diferentes.

O ciclo de vida do dispositivo começa com a criação da coisa gerenciada em integrações gerenciadas usando a `CreateManagedThing` API e termina quando o cliente exclui a coisa gerenciada usando a API. `DeleteManagedThing` O ciclo de vida de um dispositivo é gerenciado pelo seguinte público: APIs

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

- `DeleteManagedThing`

Perfil do dispositivo

Um perfil de dispositivo representa um tipo específico de dispositivo, como uma lâmpada ou campainha. Ele está associado a um fabricante e contém os recursos do dispositivo. O perfil do dispositivo armazena os materiais de autenticação necessários para solicitações de configuração de conectividade do dispositivo com integrações gerenciadas. Os materiais de autenticação usados são o código de barras do dispositivo.

Durante o processo de fabricação do dispositivo, o fabricante pode registrar seus perfis de dispositivos com integrações gerenciadas. Isso permite que o fabricante obtenha os materiais necessários para os dispositivos a partir de integrações gerenciadas durante os fluxos de trabalho de integração e provisionamento. Os metadados do perfil do dispositivo são armazenados no dispositivo físico ou impressos na etiqueta do dispositivo. O ciclo de vida do perfil do dispositivo termina quando o fabricante o exclui nas integrações gerenciadas.

Modelos de dados

Um modelo de dados representa a hierarquia organizacional de como os dados são organizados em um sistema. Além disso, ele oferece suporte à end-to-end comunicação em toda a implementação do dispositivo. Para integrações gerenciadas, há dois modelos de dados usados. O modelo de dados de integrações gerenciadas e a AWS implementação do Matter Data Model. Eles têm semelhanças, mas também diferenças sutis que são descritas nos tópicos a seguir.

Para dispositivos de terceiros, os dois modelos de dados são usados para comunicação entre o usuário final, as integrações gerenciadas e o provedor de nuvem terceirizado. Para traduzir mensagens como comandos do dispositivo e eventos do dispositivo dos dois modelos de dados, a funcionalidade do Cloud-to-Cloud Conector é aproveitada.

Tópicos

Modelo de dados de integrações gerenciadas

O modelo de dados de integrações gerenciadas gerencia toda a comunicação entre o usuário final e as integrações gerenciadas.

Hierarquia de dispositivos

Os elementos de `capability` dados `endpoint` e são usados para descrever um dispositivo no modelo de dados de integrações gerenciadas.

`endpoint`

`endpoint`Representa as interfaces lógicas ou os serviços oferecidos pelo recurso.

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

`Capability`

O `capability` representa os recursos do dispositivo.

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

Para o elemento de `capability` dados, há três itens que compõem esse item: `propertyaction`, e. `event` Eles podem ser usados para interagir e monitorar o dispositivo.

- Propriedade: Estados que são mantidos pelo dispositivo, como o atributo do nível de brilho atual de uma luz regulável.
  - ```
    {
        "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

    }
    ```
- Ação: Tarefas que podem ser executadas, como trancar uma porta na fechadura da porta. As ações podem gerar respostas e resultados.
  - ```
    {
        "name": { "$ref": "/schema-versions/definition/aws.name@1.0" }, //required
        "parameters": Map<String name, JSONNode value>,
        "responseCode": HTTPResponseCode,
        "errors": {
            "code": "string",
            "message": "string"
        }
    }
    ```

- Evento: Essencialmente, um registro de transições de estado passadas. Embora `property` representem os estados atuais, os eventos são um diário do passado e incluem um contador monotonicamente crescente, um registro de data e hora e uma prioridade. Eles permitem capturar transições de estado, bem como modelagem de dados que não é facilmente alcançada com. `property`
  - ```
    {
        "name": { "$ref": "/schema-versions/definition/aws.name@1.0" },        //
    required
        "parameters": Map<String name, JSONNode value>
    }
    ```

AWS implementação do modelo de dados Matter

A AWS implementação do Matter Data Model gerencia toda a comunicação entre integrações gerenciadas e provedores de nuvem terceirizados.

Para obter mais informações, consulte Matter Data Model: Developer Resources.

Hierarquia de dispositivos

Há dois elementos de dados usados para descrever um dispositivo: `endpoint` `cluster` e.

`endpoint`

`endpoint`Representa as interfaces lógicas ou os serviços oferecidos pelo recurso.

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

`cluster`

O `cluster` representa os recursos do dispositivo.

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

```
   "events": EventMap<String eventId, JsonNode>
}
```

Para o elemento de `cluster` dados, há três itens que compõem esse item: `attribute`e `command`. `event` Eles podem ser usados para interagir e monitorar o dispositivo.

- Atributo: Estados mantidos pelo dispositivo, como o atributo do nível de brilho atual de uma luz regulável.
  - ```
    {
        "id" (hexadecimalString): (JsonNode) value
    }
    ```
- Comando: Tarefas que podem ser executadas, como trancar uma porta na fechadura da porta. Os comandos podem gerar respostas e resultados.
  - ```
    "id": {
        "fieldId": "fieldValue",
        ...
        "responseCode": HTTPResponseCode,
        "errors": {
            "code": "string",
            "message": "string"
        }
    }
    ```
- Evento: Essencialmente, um registro de transições de estado passadas. Embora `attributes` representem os estados atuais, os eventos são um diário do passado e incluem um contador monotonicamente crescente, um registro de data e hora e uma prioridade. Eles permitem capturar transições de estado, bem como modelagem de dados que não é facilmente alcançada com. `attributes`
  - ```
    "id": {
        "fieldId": "fieldValue",
        ...
    }
    ```

Esquemas do modelo de dados

As integrações gerenciadas oferecem suporte a dois tipos de esquema: capacidade e definição de tipo. Se você estiver criando um modelo de dados personalizado, use um documento de esquema
    ```

JSON para definir qualquer tipo de esquema. Cada documento de esquema tem um limite de 50.000 caracteres.

Esquema de capacidade

Um recurso é um alicerce fundamental que representa funcionalidades específicas em um endpoint. Com recursos, você pode modelar estados e comportamentos de dispositivos usando propriedades, ações e eventos. As propriedades permitem que você modele os atributos de estado do dispositivo de forma flexível com qualquer tipo de dados declarativo. Ações e eventos modelam o comportamento do dispositivo, incluindo comandos que ele pode executar e sinais que ele pode relatar.

A seguir, é exibida uma estrutura de alto nível de um esquema de recursos.

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

Ação

Uma entidade que representa uma interação com a capacidade de um dispositivo. Por exemplo, toque a campainha ou veja quem está na porta.

Event

Uma entidade que representa um evento a partir de uma capacidade do dispositivo. Um dispositivo pode enviar um evento para relatar um incidente, alarme ou atividade de um sensor, como uma batida na porta.

Propriedade

Uma entidade que representa um atributo específico no estado do dispositivo. Por exemplo, uma campainha está tocando ou a luz da varanda está acesa

Cada recurso inclui um identificador exclusivo com namespace, informações de versão e uma descrição de sua finalidade. O documento do esquema usa controle de versão semântico para manter a compatibilidade com versões anteriores e, ao mesmo tempo, ativar novos recursos.

Para obter mais informações, consulte Esquema para definições de capacidade.

Esquema de definição de tipo

Uma definição de tipo é um tipo de dados estruturado declarativo que permite a reutilização e a capacidade de composição. Ele define como as informações devem ser formatadas e restringidas. Use definições de tipo para criar formatos de dados padronizados em toda a sua solução de IoT.

Cada definição de tipo inclui:

- Um identificador exclusivo com namespace
- Cargo
- Descrição
- Propriedades que definem a formatação e as restrições dos dados

Os tipos podem ser primitivos simples, como números inteiros ou cadeias de caracteres com limites definidos, ou estruturas complexas, como enumerações ou objetos personalizados com vários campos. As definições de tipo usam a sintaxe do esquema JSON para especificar restrições, incluindo valores mínimos e máximos, comprimentos de string e padrões permitidos.

Para obter mais informações, consulte Esquema para definições de tipo.

Esquema para definições de capacidade

Um recurso é documentado usando um documento JSON declarativo que fornece um contrato claro de como o recurso deve funcionar no sistema.

Para um recurso, os elementos obrigatórios são `$idname`,`extrinsicId`, `extrinsicVersion` e pelo menos um elemento em pelo menos uma das seções a seguir:

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

Os elementos opcionais em um recurso são `$ref` `titledescription`,`version`,, `$defs` `extrinsicProperties` e. Para obter uma capacidade, `$ref` deve consultar `aws.capability` a.

As seções a seguir detalham o esquema usado para definições de capacidade.

$id (obrigatório)

O elemento $id identifica a definição do esquema. Ele deve seguir esta estrutura:

- Comece com o prefixo `/schema-versions/` URI
- Incluir o `capability` tipo de esquema
- Use uma barra (`/`) como separador de caminho de URI
- Inclua a identidade do esquema, com fragmentos separados por pontos () `.`
- Use o `@` caractere para separar o ID do esquema e a versão
- Termine com a versão semver, usando pontos (`.`) para separar os fragmentos da versão

A identidade do esquema deve começar com um namespace raiz de 3 a 12 caracteres, seguido por um nome e um subnamespace opcionais.

A versão semver inclui uma versão MAJOR (até 3 dígitos), uma versão MINOR (até 3 dígitos) e uma versão PATCH opcional (até 4 dígitos).

> **Note**
>
> Você não pode usar os namespaces `aws` reservados ou `matter`

Example Exemplo $id

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

$ ref

O `$ref` elemento faz referência a um recurso existente no sistema. Ele segue as mesmas restrições do `$id` elemento.

> **Note**
>
> Deve existir uma definição de tipo ou capacidade com o valor fornecido no `$ref` arquivo.

Example Exemplo $ref

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

nome (obrigatório)

O elemento name é uma string que representa o nome da entidade no documento do esquema. Geralmente contém abreviações e deve seguir estas regras:

- Contém somente caracteres alfanuméricos, pontos (.), barras (/), hífens (-) e espaços
- Comece com uma letra
- Máximo de 64 caracteres

O elemento name é usado na interface de usuário e na documentação do console da Amazon Web Services.

Example Nomes de exemplo

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

título

O elemento title é uma string descritiva para a entidade representada pelo documento do esquema. Ele pode conter qualquer caractere e é usado na documentação. O tamanho máximo de um título de capacidade é de 256 caracteres.

Example Exemplos de títulos

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

description

O `description` elemento fornece uma explicação detalhada da entidade representada pelo documento do esquema. Ele pode conter qualquer caractere e é usado na documentação. O tamanho máximo para uma descrição de capacidade é de 2.048 caracteres.

Example Descrição do exemplo

```
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

O elemento `version` é opcional. É uma string que representa a versão do documento do esquema. As seguintes restrições se aplicam:

- Usa o formato semver, com os seguintes fragmentos de versão separados por `.` (pontos).
  - `MAJOR`versão, máximo de 3 dígitos
  - `MINOR`versão, máximo de 3 dígitos
  - `PATCH`versão (opcional), máximo de 4 dígitos
- O comprimento pode ter entre 3 e 12 caracteres.

Example Versões de exemplo

```
1.0
```

```
1.12
```

```
1.4.1
```

Trabalhando com versões de recursos

Um recurso é uma entidade versionada imutável. Espera-se que qualquer alteração crie uma nova versão. O sistema usa versionamento semântico com formato MAJOR.MINOR.PATCH, onde:

- A versão MAJOR aumenta ao fazer alterações de API incompatíveis com versões anteriores

- A versão MINOR aumenta ao adicionar funcionalidades de maneira compatível com versões anteriores
- A versão PATCH aumenta ao fazer pequenas adições não impactantes no recurso.

Os recursos derivados dos clusters do Matter são baseados na versão 1.4 e espera-se que cada versão do Matter seja importada para o sistema. Como a versão Matter consome os níveis MAJOR e MINOR do semver, as integrações gerenciadas só podem usar as versões PATCH.

Ao adicionar versões de PATCH para o Matter, lembre-se de que o Matter usa revisões sequenciais. Todas as versões do PATCH devem estar em conformidade com a revisão documentada na especificação Matter e devem ser compatíveis com versões anteriores.

Para corrigir quaisquer problemas de incompatibilidade com versões anteriores, você deve trabalhar com a Connectivity Standards Alliance (CSA) para resolver os problemas da especificação e lançar uma nova revisão.

AWS-os recursos gerenciados foram lançados com uma versão inicial do`1.0`. Com eles, todos os três níveis da versão podem ser usados.

Versão extrínseca (obrigatória)

Essa é uma string representando uma versão gerenciada fora do AWS IoT sistema. Para recursos do Matter, `extrinsicVersion` mapeia para `revision`

Ele é representado como um valor inteiro em cadeia e o comprimento pode ser de 1 a 10 dígitos numéricos.

Example Versões de exemplo

```
7
```

```
1567
```

ExtrinsICID (obrigatório)

O `extrinsicId` elemento representa um identificador gerenciado fora do sistema de IoT da Amazon Web Services. Para os recursos do Matter`clusterId`, ele mapeia para `attributeId` `commandId`ou`eventId`,,`fieldId`, ou, dependendo do contexto.

O `extrinsicId` pode ser um inteiro decimal em sequência (1 a 10 dígitos) ou um inteiro hexadecimal em sequência (prefixo 0x ou 0X, seguido por 1 a 8 dígitos hexadecimais).

> **Note**
>
> Para AWS, o ID do fornecedor (VID) é 0x1577 e, para o Matter, é 0. O sistema garante que os esquemas personalizados não os usem reservados VIDs para recursos.

Example Exemplo de extrinsicidas

```
0018
0x001A
0x15771002
```

$ defs

A `$defs` seção é um mapa de subesquemas que podem ser referenciados no documento do esquema, conforme permitido pelo esquema JSON. Nesse mapa, a chave é usada nas definições de referência locais e o valor fornece o esquema JSON.

> **Note**
>
> O sistema apenas impõe que `$defs` seja um mapa válido e que cada subesquema seja um esquema JSON válido. Nenhuma regra adicional é aplicada.

Siga estas restrições ao trabalhar com definições:

- Use somente caracteres compatíveis com URI em nomes de definição
- Certifique-se de que cada valor seja um subesquema válido
- Inclua qualquer número de subesquemas que se encaixem nos limites de tamanho do documento do esquema

Propriedades extrínsecas

O `extrinsicProperties` elemento contém um conjunto de propriedades definidas em um sistema externo, mas mantidas dentro do modelo de dados. Para os recursos do Matter, ele mapeia

para diferentes elementos não modelados ou parcialmente modelados dentro do cluster, atributo, comando ou evento ZCL.

As propriedades extrínsecas devem seguir estas restrições:

- Os nomes das propriedades devem ser alfanuméricos, sem espaços ou caracteres especiais
- Os valores das propriedades podem ser qualquer valor do esquema JSON
- Máximo de 20 propriedades

O sistema oferece suporte a vários `extrinsicPropertiesaccess`, incluindo`apiMaturity`,`cli`,`cliFunctionName`, e outros. Essas propriedades facilitam a ACL para AWS (e vice-versa) as transformações do modelo de dados.

> **Note**
>
> As propriedades extrínsecas são suportadas para os elementos `actionevent`,`property`, e `struct` fields de um recurso, mas não para o recurso ou cluster em si.

Propriedades extrínsecas suportadas pelo sistema

O sistema rastreia os seguintes atributos de cluster, atributo, comando ou evento não modelados ou parcialmente modelados `extrinsicProperties` durante as transformações de ou para a ZCL:

`access`

Cada objeto de acesso contém o seguinte:

- op- Operação modelada como e `enum` com valores:`read`,`write`, ou `invoke`
- `privilege`- Privilégio modelado como um `enum` com valores:`view`,, `proxy_viewoperate`, ou `manage` `administer`
- `role`- Sequência ilimitada representando uma função de operador

`apiMaturity`

Uma sequência simples ilimitada representando o nível de maturidade. Isso é modelado em ZCL como um `enum` com valores:`stable`,,`provisional`, ou `internal` `deprecated`

`side`

Modelado como um enum com valores:`either`,, e `server` `client`

Propriedades booleanas

As propriedades a seguir são sinalizadores booleanos:

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

Propriedades da string

As propriedades a seguir são representadas como cadeias de caracteres ilimitadas:

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

Considerações sobre transformação

Para transformações ZCL, `extrinsicProperties` são armazenadas em um mapa sem processamento. Esquemas personalizados que usam descoberta não passam pela transformação ZCL. No entanto, se você planeja implementar transformações ZCL para esquemas personalizados no futuro, deve modelar todos os tipos de string simples ilimitados `extrinsicProperties` e definir

restrições como enums, padrões (regex) e comprimento. Essa preparação garante o manuseio adequado dessas propriedades durante a transformação.

Por outro lado, AWS as transformações de dois conectores não `extrinsicProperties` estão incluídas, pois esses detalhes não são necessários no formato do conector.

Propriedades

As propriedades representam os estados do recurso gerenciados pelo dispositivo. Cada estado é definido como um par de valores-chave, em que a chave descreve o nome do estado e o valor descreve a definição do estado.

Ao trabalhar com propriedades, siga estas restrições:

- Use somente caracteres alfanuméricos nos nomes das propriedades, sem espaços ou caracteres especiais
- Inclua qualquer número de propriedades que se encaixem nos limites de tamanho do documento do esquema

Trabalhando com propriedades

Uma propriedade dentro de um recurso é um elemento fundamental que representa um estado específico de um dispositivo alimentado por integrações gerenciadas. Ela representa a condição ou configuração atual do dispositivo. Ao padronizar a forma como essas propriedades são definidas e estruturadas, os sistemas domésticos inteligentes garantem que dispositivos de diferentes fabricantes possam se comunicar de forma eficaz, criando uma experiência perfeita e interoperável.

Para uma propriedade de capacidade, os elementos obrigatórios são `extrinsicId` `value` e. Os elementos opcionais em uma propriedade de capacidade são `description` `retrievable` `mutable`,, `reportable` `extrinsicProperties` e.

Valor

Uma estrutura ilimitada que permite que os construtores coloquem qualquer restrição compatível com o esquema JSON para definir o tipo de dados dessa propriedade.

Ao definir valores, siga estas restrições:

- Para tipos simples, use `type` e quaisquer outras restrições nativas do esquema JSON, como ou `maxLength` `maximum`

- Para tipos compostos, use `oneOf`, `allOf`, ou `anyOf`. O sistema não suporta a `not` palavra-chave
- Para se referir a qualquer tipo global, use `$ref` com uma referência detectável válida
- Para a nulidade, siga a definição do esquema do tipo OpenAPI fornecendo ao atributo anulável um sinalizador booleano (se nulo for um valor permitido) `true`

Exemplo:

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

Recuperável

Um booleano que descreve se o estado é legível ou não.

O aspecto de legibilidade do estado é adiado para a implementação do recurso pelo dispositivo. O dispositivo decide se um determinado estado é legível ou não. Esse aspecto do estado ainda não tem suporte para ser relatado no relatório de capacidade e, portanto, não é aplicado no sistema.

Exemplo: `true` ou `false`

Mutable

Um booleano que descreve se o estado é gravável ou não.

O aspecto de capacidade de gravação do estado é adiado para a implementação do recurso pelo dispositivo. O dispositivo decide se um determinado estado é gravável ou não. Esse aspecto do estado ainda não tem suporte para ser relatado no relatório de capacidade e, portanto, não é aplicado no sistema.

Exemplo: `true` ou `false`

Reportável

Um booleano que descreve se o estado é relatado pelo dispositivo quando há uma mudança no estado.

O aspecto de reportabilidade do estado é adiado para a implementação do recurso pelo dispositivo. O dispositivo decide se um determinado estado é reportável ou não. Esse aspecto do estado ainda não tem suporte para ser relatado no relatório de capacidade e, portanto, não é aplicado no sistema.

Exemplo: `true` ou `false`

Ações

As ações são operações gerenciadas pelo esquema que seguem um modelo de solicitação-resposta. Cada ação representa uma operação implementada pelo dispositivo.

Siga estas restrições ao implementar ações:

- Incluir somente ações exclusivas na matriz de ações
- Inclua qualquer número de ações que se encaixem nos limites de tamanho do documento do esquema

Trabalhar com ações

Uma ação é uma forma padronizada de interagir e controlar os recursos do dispositivo em um sistema de integração gerenciado. Ele representa um comando ou operação específica que pode ser executada em um dispositivo, com um formato estruturado para modelar quaisquer parâmetros de solicitação ou resposta necessários. Essas ações servem como ponte entre as intenções do usuário e as operações do dispositivo, permitindo um controle consistente e confiável em diferentes tipos de dispositivos inteligentes.

Para uma ação, os elementos obrigatórios são `name` `extrinsicId` e. Os elementos opcionais são `description` `extrinsicProperties`, `request` `response` e.

Descrição

A descrição tem uma restrição de tamanho máximo de 1536 caracteres.

Solicitação

A seção de solicitação é opcional e pode ser omitida se não houver parâmetros de solicitação. Se omitido, o sistema suporta o envio de uma solicitação sem qualquer carga usando apenas o nome de. `Action` Isso é usado em ações simples, como acender ou apagar uma luz.

As ações complexas precisam de parâmetros adicionais. Por exemplo, uma solicitação para transmitir imagens da câmera pode incluir parâmetros sobre o protocolo de streaming a ser usado ou se o stream deve ser enviado para um dispositivo de exibição específico.

Para uma solicitação de ação, o elemento obrigatório é`parameters`. Os elementos opcionais são `descriptionextrinsicId`, `extrinsicProperties` e.

Descrição da solicitação

A descrição segue o mesmo formato da seção 3.5, com um tamanho máximo de 2048 caracteres.

Resposta

Nas integrações gerenciadas, para qualquer solicitação de ação enviada pela SendManagedThingCommandAPI, a solicitação chega ao dispositivo e espera uma resposta assíncrona. A resposta da ação define a estrutura dessa resposta.

Para uma solicitação de ação, o elemento obrigatório é`parameters`. Os elementos opcionais são `name` `descriptionextrinsicId`,`extrinsicProperties`,, `errors` `responseCode` e.

Descrição da resposta

A descrição segue o mesmo formato e tem um tamanho máximo de 2048 caracteres. description

Nome da resposta

O nome segue o mesmo formato denome (obrigatório), com esses detalhes adicionais:

- O nome convencional de uma resposta é derivado do acréscimo `Response` ao nome da ação.
- Se quiser usar um nome diferente, você pode fornecê-lo nesse `name` elemento. Se a `name` for fornecido na resposta, esse valor terá maior precedência do que o nome convencional.

Erros

Uma matriz ilimitada de mensagens exclusivas fornecidas na resposta, se houver erros durante o processamento da solicitação.

Restrições:

- Um item de mensagem é declarado como um objeto JSON com os seguintes campos:
  - `code`: uma sequência de caracteres contendo caracteres alfanuméricos e _ (sublinhados), com um comprimento entre 1 e 64 caracteres

- `message`: um valor de string ilimitado

Example Exemplo de mensagem de erro

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

Código de resposta

Um código inteiro mostrando como a solicitação foi tratada. Recomendamos que o código do dispositivo retorne um código usando a especificação do código de status de resposta do servidor HTTP para permitir uniformidade no sistema.

Restrição: um valor inteiro que varia de 100 a 599.

Parâmetros de solicitação ou resposta

A seção de parâmetros é definida como um mapa de pares de nomes e subesquemas. Qualquer número de parâmetros pode ser definido nos parâmetros da solicitação, se eles couberem no documento do esquema.

Os nomes dos parâmetros só podem conter caracteres alfanuméricos. Espaços ou quaisquer outros caracteres não são permitidos.

campo de parâmetros

Os elementos obrigatórios em um `parameter` são `extrinsicId` `value` e. Os elementos opcionais são `description` `extrinsicProperties` e.

O elemento de descrição segue o mesmo formato quedescription, com um comprimento máximo de 1024 caracteres.

`extrinsicId`e **`extrinsicProperties`** substituições

o `extrinsicId` e `extrinsicProperties` segue o mesmo formato de ExtrinsICID (obrigatório) ePropriedades extrínsecas, com esses detalhes adicionais:

- Se um `extrinsicId` for fornecido na solicitação ou resposta, esse valor terá maior precedência do que o valor fornecido no nível da ação. O sistema deve usar o request/response nível `extrinsicId` primeiro, se estiver faltando, use o nível de ação `extrinsicId`
- Se `extrinsicProperties` forem fornecidas na solicitação ou resposta, essas propriedades terão maior precedência do que o valor va fornecido no nível da ação. O sistema deve assumir o nível de ação `extrinsicProperties` e substituir os pares de valores-chave fornecidos no nível request/response `extrinsicProperties`

Example Exemplo de substituição de extrinsicId e extrinsicProperties

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

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

No exemplo acima, os valores efetivos para a solicitação de ação seriam:

```
# 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": {
   ...
}
```

Ações integradas

Para todos os recursos, você pode realizar ações personalizadas usando as palavras-chave `ReadState` `UpdateState` e. Essas duas palavras-chave de ação atuarão nas propriedades do recurso definidas no modelo de dados.

ReadState

Envia um comando para o `managedThing` para ler os valores de suas propriedades de estado. Use `ReadState` como forma de forçar a atualização do estado do dispositivo.

UpdateState

Envia um comando para atualizar algumas das propriedades.

Forçar a sincronização do estado do dispositivo pode ser útil nos seguintes cenários:

1. O dispositivo ficou off-line por um período de tempo e não estava emitindo eventos.
2. O dispositivo acabou de ser provisionado e ainda não tem nenhum estado mantido na nuvem.

3. O estado do dispositivo está fora de sincronia com o estado real do dispositivo.

ReadState exemplos

Verifique se a luz está acesa ou apagada usando a SendManagedThingCommandAPI:

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

Leia todas as propriedades do estado do `matter.OnOff` recurso:

```
{
  "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 exemplo

Altere o `OnTime` por uma luz usando a SendManagedThingCommandAPI:

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

Eventos

Os eventos são sinais unidirecionais gerenciados pelo esquema implementados pelo dispositivo.

Implemente eventos de acordo com essas restrições:

- Inclua somente eventos exclusivos na matriz de eventos
- Inclua qualquer número de eventos que se encaixem nos limites de tamanho do documento do esquema

Eventos em sistemas de integração gerenciados

Trabalho com eventos

Um evento é uma forma padronizada de aprender proativamente sobre mudanças em um dispositivo ou em seus arredores. Ele representa um evento modelado que o dispositivo enviará para a nuvem para fornecer informações sobre algo que foi modificado no dispositivo ou detectado em seu ambiente. Como esses eventos são modelados, os clientes podem usá-los em um fluxo de controle para reagir a eventos específicos e aos detalhes fornecidos dentro deles.

Para um evento, os elementos obrigatórios são `name` `extrinsicId` e. Os elementos opcionais são `descriptionextrinsicProperties`, `request` e.

Descrição

A descrição segue o mesmo formato descrito emdescription, com um comprimento máximo de 512 caracteres.

Solicitação

A `request` seção é opcional e pode ser omitida se não houver parâmetros de solicitação. Se omitido, o sistema suporta um dispositivo que envia uma solicitação de evento sem qualquer carga usando apenas o nome do evento. Isso é usado em eventos simples, como uma falha do sensor em uma bomba ou se o alarme for silenciado em um alarme de fumaça ou monóxido de carbono.

As ações complexas precisam de parâmetros adicionais. Por exemplo, uma solicitação para transmitir imagens da câmera pode incluir parâmetros sobre o protocolo de streaming a ser usado ou se o stream deve ser enviado para um dispositivo de exibição específico.

Para uma solicitação de evento, o elemento obrigatório é`parameters`. Não há elementos opcionais.

Resposta

Atualmente, não há suporte para respostas de eventos.

Esquema para definições de tipo

As seções a seguir detalham o esquema usado para definições de tipo.

$ id

O elemento $id identifica a definição do esquema. Ele deve seguir esta estrutura:

- Comece com o prefixo `/schema-versions/` URI
- Incluir o `definition` tipo de esquema
- Use uma barra (`/`) como separador de caminho de URI
- Inclua a identidade do esquema, com fragmentos separados por pontos () `.`
- Use o `@` caractere para separar o ID do esquema e a versão
- Termine com a versão semver, usando pontos (`.`) para separar os fragmentos da versão

A identidade do esquema deve começar com um namespace raiz de 3 a 12 caracteres, seguido por um nome e um subnamespace opcionais.

A versão semver inclui uma versão MAJOR (até 3 dígitos), uma versão MINOR (até 3 dígitos) e uma versão PATCH opcional (até 4 dígitos).

> **Note**
>
> Você não pode usar os namespaces `aws` reservados ou `matter`

Example Exemplo $id

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

$ ref

O elemento $ref faz referência a uma definição de tipo existente no sistema. Ele segue as mesmas restrições do `$id` elemento.

> **Note**
>
> Deve existir uma definição de tipo ou capacidade com o valor fornecido no `$ref` arquivo.

Example Exemplo $ref

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

nome

O elemento name é uma string que representa o nome da entidade no documento do esquema. Geralmente contém abreviações e deve seguir estas regras:

- Contém somente caracteres alfanuméricos, pontos (.), barras (/), hífens (-) e espaços
- Comece com uma letra
- Máximo de 192 caracteres

O elemento name é usado na interface de usuário e na documentação do console da Amazon Web Services.

Example Nomes de exemplo

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

título

O elemento title é uma string descritiva para a entidade representada pelo documento do esquema. Ele pode conter qualquer caractere e é usado na documentação.

Example Exemplos de títulos

```
Real-time Communication (RTC) Session Controller
```

```
Energy EVSE Capability
```

description

O `description` elemento fornece uma explicação detalhada da entidade representada pelo documento do esquema. Ele pode conter qualquer caractere e é usado na documentação.

Example Descrição do exemplo

```
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.
```

ID extrínseco

O `extrinsicId` elemento representa um identificador gerenciado fora do sistema de IoT da Amazon Web Services. Para os recursos do Matter`clusterId`, ele mapeia para `attributeId` `commandIdeventId`,,`fieldId`, ou, dependendo do contexto.

O `extrinsicId` pode ser um inteiro decimal em sequência (1 a 10 dígitos) ou um inteiro hexadecimal em sequência (prefixo 0x ou 0X, seguido por 1 a 8 dígitos hexadecimais).

> **Note**
>
> Para AWS, o ID do fornecedor (VID) é 0x1577 e, para o Matter, é 0. O sistema garante que os esquemas personalizados não os usem reservados VIDs para recursos.

Example Exemplo de extrinsicidas

```
0018
0x001A
0x15771002
```

Propriedades extrínsecas

O `extrinsicProperties` elemento contém um conjunto de propriedades definidas em um sistema externo, mas mantidas dentro do modelo de dados. Para os recursos do Matter, ele mapeia

para diferentes elementos não modelados ou parcialmente modelados dentro do cluster, atributo, comando ou evento ZCL.

As propriedades extrínsecas devem seguir estas restrições:

- Os nomes das propriedades devem ser alfanuméricos, sem espaços ou caracteres especiais
- Os valores das propriedades podem ser qualquer valor do esquema JSON
- Máximo de 20 propriedades

O sistema oferece suporte a vários `extrinsicPropertiesaccess`, incluindo`apiMaturity`,`cli`,`cliFunctionName`, e outros. Essas propriedades facilitam a ACL para AWS (e vice-versa) as transformações do modelo de dados.

> **Note**
>
> As propriedades extrínsecas são suportadas para os elementos `actionevent`,`property`, e `struct` fields de um recurso, mas não para o recurso ou cluster em si.

Propriedades extrínsecas suportadas pelo sistema

O sistema rastreia os seguintes atributos de cluster, atributo, comando ou evento não modelados ou parcialmente modelados `extrinsicProperties` durante as transformações de ou para a ZCL:

`access`

Cada objeto de acesso contém o seguinte:

- op- Operação modelada como e `enum` com valores:`read`,`write`, ou `invoke`
- `privilege`- Privilégio modelado como um `enum` com valores:`view`,, `proxy_viewoperate`, ou `manage` `administer`
- `role`- Sequência ilimitada representando uma função de operador

`apiMaturity`

Uma sequência simples ilimitada representando o nível de maturidade. Isso é modelado em ZCL como um `enum` com valores:`stable`,,`provisional`, ou `internal` `deprecated`

`side`

Modelado como um enum com valores:`either`,, e `server` `client`

Propriedades booleanas

As propriedades a seguir são sinalizadores booleanos:

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

Propriedades da string

As propriedades a seguir são representadas como cadeias de caracteres ilimitadas:

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

Considerações sobre transformação

Para transformações ZCL, `extrinsicProperties` são armazenadas em um mapa sem processamento. Esquemas personalizados que usam descoberta não passam pela transformação ZCL. No entanto, se você planeja implementar transformações ZCL para esquemas personalizados no futuro, deve modelar todos os tipos de string simples ilimitados `extrinsicProperties` e definir

restrições como enums, padrões (regex) e comprimento. Essa preparação garante o manuseio adequado dessas propriedades durante a transformação.

Por outro lado, AWS as transformações de dois conectores não `extrinsicProperties` estão incluídas, pois esses detalhes não são necessários no formato do conector.

Criação e uso de definições de tipo em documentos do esquema de recursos

Todos os elementos nos esquemas se resolvem em definições de tipo. Essas definições de tipo são definições de tipo primitivo (como booleanos, cadeias de caracteres, números) ou definições de tipo com namespace (definições de tipo criadas a partir de definições de tipo primitivas por conveniência).

Ao definir um esquema personalizado, você pode usar definições primitivas e definições de tipo de namespace.

Sumário

Definições de tipo primitivo

As definições de tipo primitivo são os alicerces de todas as definições de tipo definidas em integrações gerenciadas. Todas as definições de namespace, incluindo definições de tipo

personalizadas, se transformam em uma definição de tipo primitiva por meio da `$ref` palavra-chave ou da palavra-chave. `type`

Todos os tipos primitivos são anuláveis usando a `nullable` palavra-chave, e você pode identificar todos os tipos primitivos usando a palavra-chave. `type`

booleanos

Os tipos booleanos oferecem suporte a valores padrão.

Definição da amostra:

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

Suporte ao tipo inteiro

Os tipos inteiros oferecem suporte ao seguinte:

- Valores de `default`
- Valores de `maximum`
- Valores de `minimum`
- Valores de `exclusiveMaximum`
- Valores de `exclusiveMinimum`
- Valores de `multipleOf`

Se o valor `x` estiver sendo validado, o seguinte deve ser verdadeiro:

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

> **Note**
>
> Números com uma parte fracionária zero são considerados inteiros, mas números de ponto flutuante são rejeitados.
>
> ```
> 1.0 // Schema-Compliant
> 3.1415926 // NOT Schema-Compliant
> ```

Embora você possa especificar ambos `minimum` `exclusiveMinimum` e/ou ambos `maximum``exclusiveMaximum`, não recomendamos usar os dois simultaneamente.

Definições de amostra:

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

Definição alternativa:

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

Números

Use o tipo de número para qualquer tipo numérico, incluindo números inteiros e números de ponto flutuante.

Os tipos de números oferecem suporte ao seguinte:

- Valores de `default`
- Valores de `maximum`
- Valores de `minimum`
- Valores de `exclusiveMaximum`
- Valores de `exclusiveMinimum`
- `multipleOf`valores. O múltiplo pode ser um número de ponto flutuante.

Se o valor `x` estiver sendo validado, o seguinte deve ser verdadeiro:

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

Embora você possa especificar ambos `minimum` `exclusiveMinimum` e/ou ambos `maximum``exclusiveMaximum`, não recomendamos usar os dois simultaneamente.

Definições de amostra:

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

Definição alternativa:

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

Strings

Os tipos de string oferecem suporte ao seguinte:

- Valores de `default`
- restrições de comprimento (devem ser números não negativos), incluindo valores `maxLength` `minLength`
- `pattern`valores para expressões regulares

Quando você define expressões regulares, a string é válida se a expressão corresponder a qualquer lugar dentro da string. Por exemplo, a expressão regular p corresponde a qualquer string contendo um p, como “apple”, não apenas à string “p”. Para maior clareza, recomendamos usar expressões regulares com `^...$` (por exemplo,`^p$`), a menos que você tenha um motivo específico para não fazer isso.

Definição da amostra:

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

Nulos

Os tipos nulos aceitam apenas um único valor:`null`.

Definição da amostra:

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

Matrizes

Os tipos de matriz oferecem suporte ao seguinte:

- `default`— uma lista que será usada como valor padrão.
- `items`— Definição do tipo JSON imposta a todos os elementos da matriz.

- Restrições de comprimento (deve ser um número não negativo)
  - `minItems`
  - `maxItems`
- `pattern`valores para Regex
- `uniqueItems`— um booleano indicando se os elementos na matriz precisam ser exclusivos
- `prefixItems`— uma matriz em que cada item é um esquema que corresponde a cada índice da matriz do documento. Ou seja, uma matriz em que o primeiro elemento valida o primeiro elemento da matriz de entrada, o segundo elemento valida o segundo elemento da matriz de entrada e assim por diante.

Definição da amostra:

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

Exemplos de validação de matriz:

```
//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.
```

Definição alternativa usando validação de tupla:

```
{
    "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
```

Objetos

Os tipos de objeto oferecem suporte ao seguinte:

- Restrições de propriedade
  - `properties`— Defina as propriedades (pares de valores-chave) de um objeto usando a `properties` palavra-chave. O valor de `properties` é um objeto, em que cada chave é o nome de uma propriedade e cada valor é um esquema usado para validar essa propriedade. Qualquer propriedade que não corresponda a nenhum dos nomes de propriedade na `properties` palavra-chave é ignorada por essa palavra-chave.
  - `required`— Por padrão, as propriedades definidas pela `properties` palavra-chave não são obrigatórias. No entanto, você pode fornecer uma lista das propriedades necessárias usando a `required` palavra-chave. A `required` palavra-chave usa uma matriz de zero ou mais cadeias de caracteres. Cada uma dessas sequências de caracteres deve ser exclusiva.
  - `propertyNames`— Essa palavra-chave permite o controle sobre o RegEx padrão dos nomes das propriedades. Por exemplo, talvez você queira impor que todas as propriedades de um objeto tenham nomes seguindo uma convenção específica.
  - `patternProperties`— Isso mapeia expressões regulares para esquemas. Se o nome de uma propriedade corresponder à expressão regular fornecida, o valor da propriedade deverá ser validado em relação ao esquema correspondente. Por exemplo, use `patternProperties` para especificar que um valor deve corresponder a um esquema específico, dado um tipo específico de nome de propriedade.
  - `additionalProperties`— Essa palavra-chave controla como as propriedades extras são tratadas. Propriedades extras são propriedades cujos nomes não estão listados na palavra-chave properties ou que correspondem a qualquer uma das expressões regulares em`patternProperties`. Por padrão, propriedades adicionais são permitidas. Definir esse campo como `false` significa que nenhuma propriedade adicional é permitida.

  - `unevaluatedProperties`— Essa palavra-chave é semelhante`additionalProperties`, exceto pelo fato de poder reconhecer propriedades declaradas em subesquemas. `unevaluatedProperties`funciona coletando todas as propriedades que são validadas com êxito ao processar os esquemas e usá-las como a lista de propriedades permitidas. Isso permite que você faça coisas mais complexas, como adicionar propriedades condicionalmente. Consulte o exemplo abaixo para obter mais detalhes.
- `anyOf`— O valor dessa palavra-chave DEVE ser uma matriz não vazia. Cada item da matriz DEVE ser um esquema JSON válido. Uma instância é validada com sucesso em relação a essa palavra-chave se for validada com êxito em pelo menos um esquema definido pelo valor dessa palavra-chave.
- `oneOf`— O valor dessa palavra-chave DEVE ser uma matriz não vazia. Cada item da matriz DEVE ser um esquema JSON válido. Uma instância é validada com sucesso em relação a essa palavra-chave se for validada com base em exatamente um esquema definido pelo valor dessa palavra-chave.

Exemplo de obrigatório:

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

// Schema Compliant
{
  "test": 4
}

// NOT Schema Compliant
{}
```

PropertyNames exemplo:

```
{
  "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 exemplo:

```
{
  "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 exemplo:

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

// Schema Compliant
{
  "test": "value"
```

```
}
OR
{}

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

UnevaluatedProperties exemplo:

```
{
  "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 exemplo:

```
{
```

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

// Schema Compliant
"short"
12

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

OneOf exemplo:

```
{
  "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.
```

Definições de tipo com namespace

As definições de tipo com namespace são tipos criados a partir de tipos primitivos. Esses tipos devem seguir o formato. As integrações *`namespace`*`.`*`typename`*`.` gerenciadas fornecem tipos predefinidos nos namespaces `aws` e. `matter` Você pode usar qualquer namespace para tipos personalizados, exceto os reservados `aws` e `matter` os namespaces.

Para encontrar as definições de tipo com namespace disponíveis, use a ListSchemaVersionsAPI com o `Type` filtro definido como. `definition`

Tipos do **`matter`**

Encontre tipos de dados no `matter` namespace usando a ListSchemaVersionsAPI com o `Namespace` filtro definido como `matter` e o `Type` filtro definido como. `definition`

Tipos do **`aws`**

Encontre tipos de dados no `aws` namespace usando a ListSchemaVersionsAPI com o `Namespace` filtro definido como `aws` e o `Type` filtro definido como. `definition`

Definição do tipo de bitmap

Os bitmaps têm duas propriedades obrigatórias:

- `type`deve ser um objeto
- `properties`deve ser um objeto contendo cada definição de bit. Cada bit é um objeto com propriedades `extrinsicId` `value` e. O valor de cada bit deve ser um número inteiro com um valor mínimo de 0 e um valor máximo de pelo menos 1.

Exemplo de definição de bitmap:

```
{
    "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
}
```

Definição do tipo de enumeração

Os enums exigem três propriedades:

- `type`deve ser um objeto
- `enum`deve ser uma matriz de cadeias de caracteres exclusivas, com no mínimo um item
- `extrinsicIdMap`é um objeto com propriedades que são os valores enumerados. O valor de cada uma das propriedades deve ser o identificador extrínseco que corresponde ao valor enum.

Definição de enumeração da amostra:

```
{
    "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"
```

Gerencie comandos e eventos de dispositivos de IoT

Os comandos do dispositivo fornecem a capacidade de gerenciar remotamente um dispositivo físico, garantindo controle total sobre o dispositivo, além de realizar atualizações críticas de segurança, software e hardware. Com uma grande frota de dispositivos, saber quando um dispositivo executa um comando fornece supervisão sobre toda a implementação do dispositivo. Um comando do dispositivo ou uma atualização automática acionará uma mudança de estado do dispositivo, que por sua vez criará um novo evento do dispositivo. Esse evento do dispositivo acionará uma notificação enviada automaticamente para um destino gerenciado pelo cliente.

Tópicos

Comandos de dispositivos

Uma solicitação de comando é o comando que está sendo enviado ao dispositivo. Uma solicitação de comando inclui uma carga útil que especifica a ação a ser tomada, como acender a lâmpada. Para enviar um comando de dispositivo, a `SendManagedThingCommand` API é chamada em nome do usuário final por integrações gerenciadas e a solicitação de comando é enviada ao dispositivo.

A resposta a a `SendManagedThingCommand` é a `traceId` e você pode usá-la `traceId` para rastrear a entrega do comando e qualquer fluxo de trabalho de resposta ao comando relacionado sempre que possível.

Para obter mais informações sobre a operação `SendManagedThingCommand` da API, consulte SendManagedThingCommand.

Ação **`UpdateState`**

Para atualizar o estado de um dispositivo, como a hora em que uma luz se acende, use a `UpdateState` ação ao chamar a `SendManagedThingCommand` API. Forneça a propriedade do modelo de dados e o novo valor no qual você está atualizando`parameters`. O exemplo abaixo ilustra uma solicitação de `SendManagedThingCommand` API atualizando o `OnTime` de uma lâmpada para`5`.

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

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

Ação **`ReadState`**

Para obter o estado mais recente de um dispositivo, incluindo os valores atuais de todas as propriedades do modelo de dados, use a `ReadState` ação ao chamar a `SendManagedThingCommand` API. Em`propertiesToRead`, você pode usar as seguintes opções:

- Forneça uma propriedade específica do modelo de dados para obter o valor mais recente, como `OnOff` determinar se uma luz está acesa ou apagada.
- Use o operador curinga (`*`) para ler todas as propriedades de estado do dispositivo para um recurso.

Os exemplos abaixo ilustram os dois cenários para uma solicitação de `SendManagedThingCommand` API usando a `ReadState` ação:

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

Eventos do dispositivo

Um evento de dispositivo inclui o estado atual do dispositivo. Isso pode significar que o dispositivo mudou de estado ou está relatando seu estado mesmo que o estado não tenha sido alterado. Ele inclui relatórios de propriedades e eventos definidos no modelo de dados. Um evento pode ser o término do ciclo da máquina de lavar ou o termostato atingir a temperatura desejada definida pelo usuário final.

Notificações de eventos do dispositivo

Um usuário final pode se inscrever em destinos específicos gerenciados pelo cliente que eles criam para atualizações sobre eventos específicos do dispositivo. Para criar um destino gerenciado pelo cliente, chame a `CreateDestination` API. Quando um evento do dispositivo é reportado às integrações gerenciadas pelo dispositivo, o destino gerenciado pelo cliente é notificado, caso exista.

Marcando seus recursos de integrações gerenciadas

Para ajudá-lo a gerenciar e organizar seus recursos, você pode, opcionalmente, atribuir seus próprios metadados a cada um desses recursos na forma de tags. Esta seção descreve tags e mostra a você como criá-las.

Conceitos Básicos de Tags

Você pode usar tags para categorizar seus recursos de integrações gerenciadas de maneiras diferentes (por exemplo, por finalidade, proprietário ou ambiente). Isso é útil quando você tem muitos recursos do mesmo tipo — é possível identificar rapidamente um recurso baseado nas tags que você atribuiu a ele. Cada tag consiste em uma chave e em um valor opcional, ambos definidos por você. Por exemplo, você pode definir um conjunto de tags para os seus tipos de objetos, o que ajuda a monitorar dispositivos por tipo. Recomendamos que você crie um conjunto de chave de tags que atenda às suas necessidades para cada tipo de recurso. Usar um conjunto consistente de chaves de tags facilita para você gerenciar seus recursos.

Você pode pesquisar e filtrar recursos conforme as tags que adicionar ou aplicar. Também é possível usar tags para controlar o acesso aos recursos, conforme descrito em Utilização de tags com políticas do IAM.

Para facilitar o uso, o Editor de tags no AWS Management Console fornece uma forma central e unificada de criar e gerenciar suas tags. Para obter mais informações, consulte Trabalhando com o Tag Editor em Trabalhando com o AWS Management Console.

Você também pode trabalhar com tags usando a API AWS CLI de integrações gerenciadas. Você pode associar tags a itens gerenciados, perfis de provisionamento, armários de credenciais e tarefas over-the-air (OTA) ao criá-las usando o `Tags` campo nos seguintes comandos:

- CreateManagedThing
- CreateProvisioningProfile
- CreateCredentialLocker
- CreateOtaTask
- CreateAccountAssociation

Você pode adicionar, modificar ou excluir tags de recursos existentes que oferecem suporte a marcação, usando os seguintes comandos:

- TagResource
- ListTagsForResource
- UntagResource

É possível editar chaves de tags e valores, e é possível remover as tags de um recurso a qualquer momento. É possível definir o valor de uma tag a uma string vazia, mas não pode configurar o valor de um tag como nula. Caso adicione uma tag com a mesma chave de outra existente no recurso, o novo valor substituirá o antigo. Se você excluir um recurso, todas as tags associadas ao recurso também serão excluídas.

Restrições e limitações de tags

As restrições básicas a seguir se aplicam a tags:

- Número máximo de tags por recurso — 50
- Comprimento máximo da chave — 127 caracteres Unicode em UTF-8
- Valor máximo da chave — 255 caracteres Unicode em UTF-8
- As chaves e valores das tags diferenciam maiúsculas de minúsculas.
- Não use o prefixo `aws:` no nome nem no valor das suas tags. Está reservado para AWS uso. Você não pode editar nem excluir nomes ou valores de tag com esse prefixo. As tags com esse prefixo não contam para as tags por limite de recurso.
- Se o seu esquema de tags é usado em vários serviços e recursos , lembre-se de que outros serviços talvez tenham restrições em caracteres permitidos. Os caracteres permitidos incluem letras, espaços e números representáveis em UTF-8, além dos seguintes caracteres especiais: + - = . _ : / @.

Utilização de tags com políticas do IAM

Você pode aplicar permissões em nível de recurso com base em tags nas políticas do IAM que você usa para ações de API de integrações gerenciadas. Isso oferece a você mais controle sobre quais recursos um usuário pode criar, modificar ou usar. Você pode usar o elemento `Condition` (também chamado bloco `Condition`) juntamente com os seguintes valores e chaves de contexto de condição em uma política do IAM para controlar o acesso do usuário (permissões) baseado em tags de um recurso:

- Use `aws:ResourceTag/`*`tag-key`*`: `*`tag-value`*, para permitir ou negar ações do usuário em recursos com tags específicas.
- Use `aws:RequestTag/`*`tag-key`*`: `*`tag-value`* para exigir que uma tag específica seja (ou não seja) usada ao fazer uma solicitação de API para criar ou modificar um recurso que permite tags.
- Use `aws:TagKeys: [`*`tag-key`*`, ...]` para exigir que um conjunto específico de chaves de tag seja (ou não seja) usado ao fazer uma solicitação de API para criar ou modificar um recurso que permite tags.

> **Note**
>
> As chaves e valores de contexto da condição em uma política do IAM se aplicam somente às ações de integrações gerenciadas em que um identificador de um recurso capaz de ser marcado é um parâmetro obrigatório. Por exemplo, o uso de não GetCustomEndpointé permitido ou negado com base nas chaves e valores do contexto de condição porque nenhum recurso etiquetável (itens gerenciados, perfis de provisionamento, armários de credenciais, over-the-air tarefas) é referenciado nessa solicitação. Para obter mais informações sobre recursos de integrações gerenciadas que são marcáveis e chaves de condição que eles suportam, leia o recurso Ações, recursos e chaves de condição para integrações AWS IoT gerenciadas do. AWS IoT Device Management

Para obter mais informações sobre o uso de tags, consulte Controlar o acesso usando tags no Guia do usuário do AWS Identity and Access Management . A seção Referência de política JSON do IAM desse guia detalhou a sintaxe, as descrições e os exemplos dos elementos, variáveis e lógica de avaliação das políticas JSON no IAM.

O exemplo de política a seguir aplica duas restrições baseadas em tags para a `CreateManagedThing` ação. Um usuário do IAM restrito por essa política:

- Não é possível criar uma coisa gerenciada com a tag “env=prod” (no exemplo, veja a linha). `"aws:RequestTag/env" : "prod"`
- Não é possível modificar ou acessar uma coisa gerenciada que tenha uma tag “env=prod” existente (no exemplo, veja a linha). `"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": "*"
    }
  ]
```

```
}
```

Você também pode especificar vários valores de tag para uma determinada chave de tag, colocando-as em uma lista como esta:

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

> **Note**
>
> Se você permitir ou negar aos usuários o acesso a recursos com base em tags, considere negar explicitamente aos usuários a capacidade de adicionar essas tags ou removê-las dos mesmos recursos. Caso contrário, é possível que um usuário contorne suas restrições e obtenha acesso a um recurso modificando as tags.

Notificações de integrações gerenciadas

As notificações de integrações gerenciadas fornecem atualizações e informações importantes dos dispositivos. As notificações incluem eventos do conector, comandos do dispositivo, eventos do ciclo de vida, atualizações OTA (Over-the-Air) e relatórios de erros. Esses insights fornecem informações práticas para criar fluxos de trabalho automatizados, realizar ações imediatas ou armazenar dados de eventos para solução de problemas.

Atualmente, somente os streams de dados do Amazon Kinesis são suportados como destino para notificações de integrações gerenciadas. Primeiro, você precisará configurar um stream de dados do Amazon Kinesis e permitir que integrações gerenciadas acessem o stream de dados antes de configurar as notificações.

Configurar o Amazon Kinesis para notificações

Etapas de configuração do Amazon Kinesis

- Etapa 1: Criar um stream de dados do Amazon Kinesis
- Etapa 2: criar uma política de permissões
- Etapa 3: Navegue até o painel do IAM e selecione Funções
- Etapa 4: usar uma política de confiança personalizada
- Etapa 5: aplicar sua política de permissões
- Etapa 6: insira um nome de função

Para configurar o Amazon Kinesis para notificações de integrações gerenciadas, siga estas etapas:

Etapa 1: Criar um stream de dados do Amazon Kinesis

Um Amazon Kinesis Data Stream pode ingerir uma grande quantidade de dados em tempo real, armazená-los de forma durável e disponibilizá-los para consumo pelos aplicativos.

Para criar um stream de dados do Amazon Kinesis

- Para criar um stream de dados do Kinesis, siga as etapas descritas em Criar e gerenciar fluxos de dados do Kinesis.

Etapa 2: criar uma política de permissões

Crie uma política de permissões que permita que integrações gerenciadas acessem seu stream de dados do Kinesis.

Para criar uma política de permissões

- Para criar uma política de permissões, copie a política abaixo e siga as etapas descritas em Criar políticas usando o editor JSON

  JSON

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

Etapa 3: Navegue até o painel do IAM e selecione Funções

Abra o painel do IAM e clique em Funções.

Para navegar até o painel do IAM

- Abra o painel do IAM e clique em Funções.

  Para obter mais informações, consulte Criação de função do IAM no Guia AWS Identity and Access Managementdo usuário.

Etapa 4: usar uma política de confiança personalizada

Você pode usar uma política de confiança personalizada para conceder às integrações gerenciadas acesso ao stream de dados do Kinesis.

Para usar uma política de confiança personalizada

- Crie uma nova função e escolha Política de confiança personalizada. Clique em Avançar.

  A política a seguir permite que as integrações gerenciadas assumam a função, e a `Condition` declaração ajuda a evitar problemas confusos de deputados.

  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:*"
                }
            }
        }
    ]
}
```

Etapa 5: aplicar sua política de permissões

Adicione a política de permissões que você criou na etapa 2 à função.

Para adicionar uma política de permissões

- Na página Adicionar permissões, pesquise e adicione a política de permissões que você criou na etapa 2. Clique em Avançar.

Etapa 6: insira um nome de função

- Insira um nome de função e clique em Criar função.

Configurar notificações de integrações gerenciadas

Etapas de configuração da notificação

- Etapa 1: conceder permissões ao usuário para chamar a CreateDestination API
- Etapa 2: chame a CreateDestination API
- Etapa 3: chame a CreateNotificationConfiguration API

Para configurar notificações de integrações gerenciadas, siga estas etapas:

Etapa 1: conceder permissões ao usuário para chamar a CreateDestination API

- Conceda permissões ao usuário para chamar a **`CreateDestination`** API

  A política a seguir define os requisitos para o usuário chamar a CreateDestinationAPI.

  Consulte Conceder permissões a um usuário para passar uma função para um AWS serviço no Guia do AWS Identity and Access Managementusuário para obter permissões de senha para integrações gerenciadas.

  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":"*"
          }
       ]
}
```

Etapa 2: chame a CreateDestination API

- Chame a **`CreateDestination`** API

  Depois de criar seu stream de dados do Amazon Kinesis e sua função de acesso ao stream, chame a CreateDestinationAPI para criar seu destino de notificação para onde as notificações serão encaminhadas. Para o `DeliveryDestinationArn` parâmetro, use o `arn` do seu novo stream de dados do Amazon Kinesis.

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

> **Note**
>
> `ClientToken`é um símbolo de idempotência. Se você tentar novamente uma solicitação que foi concluída com êxito inicialmente usando o mesmo token e os mesmos parâmetros do cliente, a tentativa de repetição será bem-sucedida sem realizar nenhuma ação adicional.

Etapa 3: chame a CreateNotificationConfiguration API

- Chame a **`CreateNotificationConfiguration`** API

  Por fim, use a CreateNotificationConfigurationAPI para criar a configuração de notificação que roteia os tipos de eventos escolhidos para o seu destino representado pelo stream de dados do Kinesis. No `DestinationName` parâmetro, use o mesmo nome de destino de quando você chamou a `CreateDestination` API inicialmente.

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

Tipos de eventos monitorados com integrações gerenciadas

A seguir estão os tipos de eventos monitorados com notificações de integrações gerenciadas:

- `DEVICE_COMMAND`
  - O status do comando SendManagedThingCommandda API. Os valores válidos são `succeeded` ou `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`
  - A solicitação de comando da Web Real-Time Communication (WebRTC).

    O padrão WebRTC permite a comunicação entre dois pares. Esses pares podem transmitir vídeo, áudio e dados arbitrários em tempo real. As integrações gerenciadas oferecem suporte ao WebRTC para permitir esses tipos de streaming entre o aplicativo móvel do cliente e o dispositivo do usuário final. Para obter mais informações sobre o padrão WebRTC, consulte 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`
  - O status de descoberta do dispositivo.

```
{
      "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
  - Uma notificação da ocorrência de um evento no dispositivo.

```
{
      "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
  - O status do ciclo de vida do dispositivo.

```
{
      "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
  - Uma notificação OTA do dispositivo.
- DEVICE_STATE
  - Uma notificação quando o estado de um dispositivo foi atualizado.

```
{
      "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 conectores (C2C)

Um cloud-to-cloud conector permite criar e facilitar a comunicação bidirecional entre dispositivos de terceiros e. AWS

Tópicos

O que é um cloud-to-cloud conector (C2C)?

Um cloud-to-cloud conector é um pacote de software pré-construído que o vincula com segurança Nuvem AWS ao endpoint de um provedor de nuvem terceirizado. Usando o conector C2C, os provedores de soluções podem aproveitar as integrações gerenciadas do AWS IoT Device Management para controlar dispositivos conectados a nuvens de terceiros.

As integrações gerenciadas incluem um catálogo de conectores em que os AWS clientes podem visualizar e selecionar os conectores com os quais desejam se integrar. Para obter mais informações, consulte O que é o catálogo de conectores C2C?.

As integrações gerenciadas exigem que cada conector seja implementado como uma AWS Lambda função.

O que é o catálogo de conectores C2C?

O catálogo de conectores de integrações gerenciadas para o AWS IoT Device Management é uma coleção de conectores C2C que facilitam a comunicação bidirecional entre integrações gerenciadas do AWS IoT Device Management e um provedor de nuvem terceirizado. Você pode ver os conectores no Console de gerenciamento da AWS ou no AWS CLI.

Para usar o console para visualizar o catálogo de conectores de integrações gerenciadas

1. Abra o console de integrações gerenciadas
2. No painel de navegação esquerdo, escolha Integrações gerenciadas
3. No painel de navegação esquerdo do console de integrações gerenciadas, escolha Catálogo.

AWS Lambda funciona como conectores C2C

Cada função Lambda do conector C2C traduz e transporta comandos e eventos entre integrações gerenciadas e as ações correspondentes em plataformas de terceiros. Para obter mais informações sobre o Lambda, consulte O que é. AWS Lambda

Por exemplo, suponha que um usuário final possua uma lâmpada inteligente fabricada por um OEM terceirizado. Com um conector C2C, um usuário final pode emitir um comando para ligar ou desligar essa luz por meio de uma plataforma de integrações gerenciadas. Esse comando será então encaminhado para a função Lambda hospedada no conector, que traduzirá a solicitação em uma chamada de API para a plataforma de terceiros para ligar ou desligar o dispositivo.

A função Lambda é necessária quando você chama a `CreateCloudConnector` API. O código que é implantado na função Lambda deve implementar todas as interfaces e funcionalidades conforme mencionado em. Crie um conector C2C (nuvem a nuvem)

Fluxo de trabalho de integrações gerenciadas

Os desenvolvedores devem registrar conectores C2C com integrações gerenciadas para. AWS IoT Device Management Esse processo de registro cria um recurso de conector lógico que os clientes podem acessar para usar o conector.

> **Note**
>
> Um conector C2C é um conjunto de metadados criados em integrações gerenciadas para que o AWS IoT Device Management descreva o conector.

O diagrama a seguir mostra a função de um conector C2C ao enviar um comando do aplicativo móvel para um dispositivo conectado à nuvem. O conector C2C atua como uma camada de tradução entre integrações gerenciadas para o AWS IoT Device Management e uma plataforma de nuvem terceirizada.

Diretrizes para usar um conector C2C () cloud-to-cloud

Qualquer conector C2C que você criar é seu conteúdo, e qualquer conector C2C criado por outro cliente que você acessa é conteúdo de terceiros. AWS não cria nem gerencia nenhum conector C2C como parte das integrações gerenciadas.

Você pode compartilhar seus conectores C2C com outros clientes de integrações gerenciadas. Se o fizer, você autoriza, AWS como provedor de serviços, a listar esses conectores C2C e as informações de contato relacionadas no AWS console e entende que outros AWS clientes podem entrar em contato com você. Você é o único responsável por conceder aos clientes acesso aos seus conectores C2C e por quaisquer termos que regem o acesso de outro AWS cliente aos seus conectores C2C.

Crie um conector C2C (nuvem a nuvem)

As seções a seguir abordam as etapas para criar um conector C2C (Cloud-to-Cloud) para integrações gerenciadas para o AWS IoT Device Management.

Tópicos

Pré-requisitos

Antes de criar um conector C2C (Cloud-to-Cloud), você precisa do seguinte:

- E Conta da AWS para hospedar seu conector C2C e registrá-lo por meio de integrações gerenciadas. Para obter mais informações, consulte Criar um Conta da AWS.

- Ao criar seu conector, você precisa de certas permissões do IAM. Para usar o
- Certifique-se de que os provedores de nuvem terceirizados aos quais o conector se destina suportem a autorização OAuth 2.0. Para obter mais informações, consulte OAuth Requisitos 2.0 para vinculação de contas.

  Além disso, para testar o conector, o desenvolvedor do conector deve ter o seguinte:
  - Um ID de cliente da nuvem de terceiros para associar ao seu conector C2C
  - Um segredo de cliente da nuvem de terceiros para associar ao seu conector C2C
  - Um URL de autorização OAuth 2.0
  - Um URL de token OAuth 2.0
- Qualquer chave de API exigida pela sua API de terceiros
- Quaisquer chaves de API exigidas pelo seu registro de API de terceiros ou pela lista de permissões para o URL de OAuth retorno de chamada hospedado por. AWS Alguns terceiros permitem explicitamente um URL de OAuth redirecionamento, enquanto outros têm um fluxo de trabalho em que os usuários podem fazer login e registrar o URL. OAuth Consulte o terceiro específico para entender o que é necessário para permitir a lista de permissões do endpoint de redirecionamento de integrações OAuth gerenciadas

Permissões obrigatórias

Ao criar seu conector, você precisa de certas permissões do IAM. Além das `iotmanagedintegrations:` permissões para as ações, você precisa das seguintes permissões:

- CreateAccountAssociation, CreateConnectorDestination, GetAccountAssociation, e StartAccountAssociationRefresh, exigem `secretsmanager:GetSecretValue`
- CreateCloudConnector requer `lambda:Invoke`

Para obter mais informações sobre `iotmanagedintegrations:` permissões e ações, consulte Ações definidas por integrações AWS gerenciadas

Requisitos do conector C2C

O conector C2C que você desenvolve facilita a comunicação bidirecional entre as integrações gerenciadas do AWS IoT Device Management e a nuvem de um fornecedor terceirizado. O conector deve implementar interfaces para integrações gerenciadas para que o AWS IoT Device Management execute ações em nome dos usuários finais. Essas interfaces fornecem a funcionalidade de

descobrir dispositivos de usuários finais, iniciar comandos de dispositivos que são enviados de integrações gerenciadas para o AWS IoT Device Management e identificar usuários com base em um token de acesso. Para suportar as operações do dispositivo, o conector deve gerenciar a tradução das mensagens de solicitação e resposta entre as integrações gerenciadas do AWS IoT Device Management e a plataforma terceirizada relacionada.

A seguir estão os requisitos para o conector C2C:

- O servidor de autorização de terceiros deve estar em conformidade com os padrões OAuth 2.0, bem como com as configurações listadas em. OAuth requisitos de configuração
- Será necessário um conector C2C para interpretar os identificadores das AWS implementações do Matter Data Model e deve emitir as respostas e eventos que estejam em conformidade com as AWS implementações do Matter Data Model. Para obter mais informações, consulte AWS implementação do modelo de dados Matter.
- Um conector C2C deve ser capaz de chamar as integrações gerenciadas para o AWS IoT Device Management com autenticação. APIs `SigV4` Para eventos assíncronos enviados com a SendConnectorEvent API, as mesmas Conta da AWS credenciais usadas para registrar o conector devem ser usadas para assinar a solicitação relacionada. SendConnectorEvent
- O conector deve implementar as `AWS.DeactivateUser` operações `AWS.ActivateUser` `AWS.DiscoverDevicesAWS.SendCommand`,, e.
- Quando seu conector C2C recebe eventos de terceiros relacionados às respostas de comandos do dispositivo ou à descoberta do dispositivo, ele deve encaminhá-los para integrações gerenciadas com a API. `SendConnectorEvent` Para obter mais informações sobre esses eventos e a `SendConnectorEvent` API, consulte SendConnectorEvent.

> **Note**
>
> A `SendConnectorEvent` API faz parte do SDK de integrações gerenciadas e é usada em vez da criação manual e da assinatura de solicitações.

OAuth Requisitos 2.0 para vinculação de contas

Cada conector C2C depende de um servidor de autorização OAuth 2.0 para autenticar os usuários finais. Por meio desse servidor, os usuários finais vinculam suas contas de terceiros à plataforma do dispositivo do cliente. A vinculação de contas é a primeira etapa exigida por um usuário final

para usar dispositivos compatíveis com seu conector C2C. Para obter mais informações sobre as diferentes funções na vinculação de contas e OAuth 2.0, consulteFunções de vinculação de contas.

Embora seu conector C2C não precise implementar uma lógica comercial específica para suportar o fluxo de autorização, o servidor de autorização OAuth2 .0 associado ao seu conector C2C deve atender a. OAuth requisitos de configuração

> **Note**
>
> Integrações gerenciadas AWS IoT Device Management apenas para suporte OAuth 2.0 com um fluxo de código de autorização. Consulte RFC 6749 para obter mais informações.

A vinculação de contas é um processo que permite que integrações gerenciadas e o conector acessem os dispositivos de um usuário final usando um token de acesso. Esse token fornece integrações gerenciadas para o AWS IoT Device Management com a permissão do usuário final, de forma que o conector possa interagir com os dados do usuário final por meio de chamadas de API. Para obter mais informações, consulte Fluxo de trabalho de vinculação de contas.

Recomendamos que você não registre esses tokens confidenciais em nenhum registro. No entanto, se eles estiverem armazenados em registros, recomendamos que você use as políticas de proteção de dados do CloudWatch Logs para mascarar os tokens nos registros. Para obter mais informações, consulte Ajude a proteger dados de logs confidenciais com mascaramento.

As integrações gerenciadas AWS IoT Device Management não recebem um token de acesso diretamente; elas o fazem por meio do Tipo de Concessão de Código de Autorização. Primeiro, as integrações gerenciadas para o AWS IoT Device Management devem obter um código de autorização. Em seguida, ele troca o código por um token de acesso e um token de atualização. O token de atualização é usado para solicitar um novo token de acesso quando o token de acesso antigo expirar. Se o token de acesso e o token de atualização expirarem, você deverá executar o fluxo de vinculação de contas novamente. Você pode fazer isso com a operação `StartAccountAssociationRefresh` da API.

> **Important**
>
> O token de acesso emitido deve ser definido por usuário, mas não por OAuth cliente. O token não deve fornecer acesso a todos os dispositivos de todos os usuários do cliente.
> O servidor de autorização deve fazer o seguinte:

- Emita tokens de acesso que contenham um ID extraível do usuário final (proprietário do recurso), como um token JWT.
- Retorne o ID do usuário final para cada token de acesso emitido.

OAuth requisitos de configuração

A tabela a seguir ilustra os parâmetros necessários do seu servidor de OAuth autorização para integrações gerenciadas para que o AWS IoT Device Management realize a vinculação de contas:

OAuth Parâmetros do servidor

Campo	Obrigatório	Comentário
`clientId`	Sim	Um identificador público para seu aplicativo. Ele é usado para iniciar fluxos de autentica ção e pode ser compartilhado publicamente.
`clientSecret`	Sim	Uma chave secreta usada para autenticar o aplicativo com o servidor de autorizaç ão, especialmente ao trocar um código de autorização por um token de acesso. Ele deve ser mantido em sigilo e não compartilhado publicamente.
`authorizationType`	Sim	O tipo de autorização suportado por essa configura ção de autorização. Atualment e, "OAuth 2.0" é o único valor suportado.
`authUrl`	Sim	O URL de autorização do provedor de nuvem terceiriz ado.

`tokenUrl`	Sim	O URL do token para o provedor de nuvem terceirizado.
`tokenEndpointAuthenticationScheme`	Sim	Esquema de autenticação de “HTTP_BASIC” ou “REQUEST_BODY_CREDENTIALS”. HTTP_BASIC sinaliza que as credenciais do cliente estão incluídas no cabeçalho de autorização, enquanto a escada sinaliza que elas estão incluídas no corpo da solicitação.

O OAuth servidor que você usa deve ser configurado para que os valores da string do token de acesso sejam codificados em Base64 com o conjunto de caracteres UTF-8.

Funções de vinculação de contas

Para criar um conector C2C, você precisará de um servidor de autorização OAuth 2.0 e vinculação de contas. Para obter mais informações, consulte Fluxo de trabalho de vinculação de contas.

OAuth 2.0 define as quatro funções a seguir ao implementar a vinculação de contas:

1. Servidor de autorização
2. Proprietário do recurso (usuário final)
3. Servidor de recursos
4. Cliente

O seguinte define cada uma dessas OAuth funções:

Servidor de autorização

O servidor de autorização é o servidor que identifica e autentica a identidade de um usuário final em uma nuvem de terceiros. Os tokens de acesso fornecidos por esse servidor podem vincular

a conta da plataforma do cliente do usuário AWS final à conta da plataforma de terceiros. Esse processo é conhecido como vinculação de contas.

O servidor de autorização oferece suporte à vinculação de contas fornecendo o seguinte:

- Exibe uma página de login para o usuário final fazer login no seu sistema. Isso geralmente é chamado de endpoint de autorização.
- Autentica o usuário final em seu sistema.
- Gera um código de autorização que identifica o usuário final.
- Transmite o código de autorização para integrações gerenciadas do AWS IoT Device Management.
- Aceita o código de autorização das integrações gerenciadas do AWS IoT Device Management e retorna um token de acesso que as integrações gerenciadas do AWS IoT Device Management podem usar para acessar os dados do usuário final em seu sistema. Isso geralmente é concluído por meio de um URI separado, chamado URI de token ou endpoint.

> **Important**
>
> O servidor de autorização deve suportar o fluxo do Código de Autorização OAuth 2.0 para ser usado com integrações gerenciadas para o AWS IoT Device Management Connector. As integrações gerenciadas para o AWS IoT Device Management também oferecem suporte ao fluxo de código de autorização com o Proof Key for Code Exchange (PKCE). O servidor de autorização deve:
>
> - Emita tokens de acesso que contenham ID extraível do usuário final ou do proprietário do recurso, por exemplo, tokens JWT
> - Ser capaz de retornar o ID do usuário final para cada token de acesso emitido
>
> Caso contrário, seu conector não será capaz de suportar a `AWS.ActivateUser` operação necessária. Isso evitará o uso de conectores com integrações gerenciadas.

Se o desenvolvedor ou proprietário do conector não mantiver seu próprio servidor de autorização, o servidor de autorização usado deverá fornecer autorização para recursos gerenciados pela plataforma terceirizada do desenvolvedor do conector. Isso significa que todos os tokens recebidos pelas integrações gerenciadas do servidor de autorização devem fornecer limites de segurança significativos nos dispositivos (o recurso). Por exemplo, um token de usuário final não permite comandos no dispositivo de outro usuário final; as permissões fornecidas pelo token são mapeadas para recursos dentro da plataforma. Considere o exemplo da Lights

Incorporated. Quando um usuário final inicia o fluxo de vinculação da conta com seu conector, ele é redirecionado para a página de login da Lights Incorporated, que fica na frente do servidor de autorização. Depois de fazer login e conceder permissões ao cliente, eles fornecem um token que dá ao conector acesso aos recursos em sua conta da Lights Incorporated.

Proprietário do recurso (usuário final)

Como proprietário do recurso, você permite integrações gerenciadas para o acesso do cliente do AWS IoT Device Management aos recursos associados à sua conta por meio da vinculação de contas. Por exemplo, considere a lâmpada inteligente que um usuário final incorporou ao aplicativo móvel da Lights Incorporated. O proprietário do recurso se refere à conta do usuário final que comprou e integrou o dispositivo. Em nosso exemplo, o proprietário do recurso é modelado como uma conta da Lights Incorporated OAuth2 4.0. Como proprietária do recurso, essa conta fornece permissões para emitir comandos e gerenciar o dispositivo.

Servidor de recursos

Este é o servidor que hospeda recursos protegidos que requerem autorização para acesso (dados do dispositivo). O AWS cliente precisa acessar recursos protegidos em nome de um usuário final, e faz isso por meio de integrações gerenciadas para conectores do AWS IoT Device Management após a vinculação de contas. Considerando a lâmpada inteligente anterior como exemplo, o servidor de recursos é um serviço baseado em nuvem de propriedade da Lights Incorporated que gerencia a lâmpada após sua integração. Por meio do servidor de recursos, o proprietário do recurso pode emitir comandos para a lâmpada inteligente, como ligá-la e desligá-la. O recurso protegido só fornece permissões para a conta do usuário final e outras para as accounts/entities quais ele possa ter fornecido permissão.

Cliente

Nesse contexto, o cliente é seu conector C2C. Um cliente é definido como um aplicativo ao qual é concedido acesso aos recursos em um servidor de recursos em nome do usuário final. O processo de vinculação de contas representa o conector, o cliente, solicitando acesso aos recursos de um usuário final na nuvem de terceiros.

Embora o conector seja o OAuth cliente, as integrações gerenciadas do AWS IoT Device Management realizam operações em nome do conector. Por exemplo, integrações gerenciadas para o AWS IoT Device Management fazem solicitações ao servidor de autorização para obter um token de acesso. O conector ainda é considerado o cliente porque é o único componente que já acessa o recurso protegido (dados do dispositivo) no servidor de recursos.

Considere a lâmpada inteligente que foi integrada por um usuário final. Depois que a vinculação da conta for concluída entre a plataforma do cliente e o servidor de autorização da Lights Incorporated, o próprio conector se comunicará com o servidor de recursos para recuperar informações sobre a lâmpada inteligente do usuário final. O conector pode então receber comandos do usuário final. Isso inclui ligar ou desligar a luz em seu nome por meio do servidor de recursos da Lights Incorporated. Assim, designamos o conector como cliente.

Fluxo de trabalho de vinculação de contas

Para que as integrações gerenciadas de um cliente para a plataforma AWS IoT Device Management interajam com os dispositivos de um usuário final em sua plataforma de terceiros por meio de seu conector C2C, ele obtém o token de acesso por meio do seguinte fluxo de trabalho:

1. Quando um usuário inicia a integração de dispositivos de terceiros por meio do aplicativo do cliente, as integrações gerenciadas para o AWS IoT Device Management retornam o URI de autorização, bem como o. AssociationId
2. O front-end do aplicativo armazena AssociationId e redireciona o usuário final para a página de login da plataforma de terceiros.
   - O usuário final faz login. O usuário final concede ao cliente acesso aos dados do dispositivo.
3. A plataforma terceirizada cria um código de autorização. O usuário final é redirecionado para integrações gerenciadas para o URI de retorno de chamada da plataforma AWS IoT Device Management, incluindo o código anexado à solicitação de redirecionamento.
4. As integrações gerenciadas trocam esse código com o URI do token da plataforma de terceiros.
5. O URI do token valida o código de autorização e retorna um token de acesso e um token de atualização OAuth2 .0, associados ao usuário final.
6. As integrações gerenciadas chamam o conector C2C com `AWS.ActivateUser` operação para concluir o fluxo de vinculação de contas e obter. UserId
7. As integrações gerenciadas retornam OAuth RedirectUrl (da configuração da Política de Conectores) da página de autenticação bem-sucedida para o aplicativo do cliente.

> **Note**
>
> Em caso de falhas, as integrações gerenciadas do AWS IoT Device Management acrescentam parâmetros de consulta error e error_description ao URL, fornecendo detalhes do erro ao aplicativo do cliente.

8. O aplicativo do cliente redireciona o usuário final para o. OAuth RedirectUrl Nesse ponto, o front-end AssociationId do aplicativo conhece a associação desde a primeira etapa.

   Todas as solicitações subsequentes feitas a partir de integrações gerenciadas do AWS IoT Device Management por meio do conector C2C para a plataforma de nuvem de terceiros, como comandos para descobrir dispositivos e enviar comandos, OAuth2 incluirão o token de acesso .0.

O diagrama a seguir mostra a relação entre os principais componentes da vinculação de contas:

Implemente operações de interface do conector C2C

As integrações gerenciadas AWS IoT Device Management definem quatro operações que você AWS Lambda deve realizar para se qualificar como conector. Seu conector C2C deve implementar cada uma das seguintes operações:

1. `AWS.ActivateUser`- Integrações gerenciadas para o AWS IoT Device Management serviço chamam essa API para recuperar um identificador de usuário globalmente exclusivo associado ao token OAuth2 .0 fornecido. Opcionalmente, essa operação pode ser usada para executar quaisquer requisitos adicionais para o processo de vinculação de contas.
2. `AWS.DiscoverDevices`- integrações gerenciadas para o serviço AWS IoT Device Management chamam essa API ao seu conector para descobrir os dispositivos do usuário
3. `AWS.SendCommand`- integrações gerenciadas para o serviço AWS IoT Device Management chamam essa API ao seu conector para enviar comandos para os dispositivos do usuário
4. `AWS.DeactivateUser`- integrações gerenciadas para o serviço AWS IoT Device Management chamam essa API para seu conector para desativar o token de acesso do usuário para desvincular seu servidor de autorização.

As integrações gerenciadas AWS IoT Device Management sempre invocam a função Lambda com uma carga útil de string JSON por meio da ação. AWS Lambda `invokeFunction` As operações de solicitação devem incluir um `operationName` campo em cada carga útil da solicitação. Para saber mais, consulte Invoke in AWS Lambda API Reference.

Cada tempo limite de invocação é definido para dois segundos e, se a invocação falhar, ela será repetida cinco vezes.

O Lambda que você implementa para seu conector analisará a carga útil `operationName` da solicitação e implementará a funcionalidade correspondente para mapear para a nuvem de terceiros:

```
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**
>
> O desenvolvedor do conector deve implementar as `deactivateUser.deactivateUser` operações `activateUserManager.activateUser(request)` `deviceDiscoveryManager.listDevices(request)sendCommandManager.sendCommand(re` e listadas no exemplo anterior.

O exemplo a seguir detalha uma solicitação genérica de conector de integrações gerenciadas, na qual campos comuns para cada interface necessária estão presentes. No exemplo, você pode ver que há um cabeçalho de solicitação e uma carga útil de solicitação. Os cabeçalhos de solicitação são comuns em todas as interfaces de operação.

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

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

Cabeçalhos de solicitação padrão

Os campos de cabeçalho padrão são os seguintes.

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

Qualquer API hospedada por um conector deve processar os seguintes parâmetros de cabeçalho:

Cabeçalhos e campos padrão

Campo	Obrigatório/opcional	Descrição
`header:auth`	Sim	Informações de autorização fornecidas pelo construtor do conector C2C durante o registro do conector.
`header:auth:token`	Sim	Token de autorização do usuário gerado pelo provedor de nuvem terceirizado e vinculado `connector AssociationID` a.

`header:auth:type`	Sim	O tipo de autorização necessária.

> **Note**
>
> Todas as solicitações ao seu conector terão o token de acesso do usuário final anexado. Você pode supor que a vinculação de contas entre o usuário final e o cliente de integrações gerenciadas já tenha ocorrido.

Carga da solicitação

Além dos cabeçalhos comuns, cada solicitação terá uma carga útil. Embora essa carga tenha campos exclusivos para cada tipo de operação, cada carga tem um conjunto de campos padrão que sempre estarão presentes.

Campos de carga útil da solicitação:

- `operationName`: a operação de uma determinada solicitação, igual a um dos seguintes valores:`AWS.ActivateUser`,`AWS.SendCommand`,`AWS.DiscoverDevices`,`AWS.DeactivateUser`.
- `operationVersion`: Cada operação é versionada para permitir sua evolução ao longo do tempo e fornecer uma definição de interface estável para conectores de terceiros. As integrações gerenciadas passam um campo de versão na carga útil de todas as solicitações.
- `connectorId`: o ID do conector para o qual a solicitação foi enviada.

Cabeçalhos de resposta padrão

Cada operação responderá com uma `ACK` das integrações gerenciadas do AWS IoT Device Management, confirmando que seu conector C2C recebeu a solicitação e começou a processá-la. A seguir está um exemplo genérico dessa resposta:

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

```
  }
}
```

Cada resposta de operação deve ter o seguinte cabeçalho comum:

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

A tabela a seguir lista o cabeçalho de resposta padrão:

Cabeçalho e campo de resposta padrão

Campo	Obrigatório/opcional	Comentário
`header:responseCode`	Sim	ENUM de valores que indicam o status de execução da solicitação.

Nas várias interfaces de conectores e esquemas de API descritos neste documento, há um `Message` campo `responseMessage` ou. Esse é um campo opcional usado para que o conector C2C Lambda responda com qualquer contexto relacionado à solicitação e sua execução. De preferência, qualquer erro que resulte em um código de status diferente `200` deve incluir um valor de mensagem descrevendo o erro.

Responda às solicitações de operação do conector C2C com a API SendConnectorEvent

Integrações gerenciadas AWS IoT Device Management esperam que seu conector se comporte de forma assíncrona em todas as operações. `AWS.SendCommand` `AWS.DiscoverDevices` Isso significa que a resposta inicial a essas operações simplesmente “reconhece” que seu conector C2C recebeu a solicitação.

Usando a `SendConnectorEvent` API, espera-se que seu conector envie os tipos de eventos da lista abaixo para for `AWS.DiscoverDevices` and `AWS.SendCommand` operations, bem como eventos proativos do dispositivo (como uma luz sendo ligada e desligada manualmente). Para

ler uma explicação detalhada desses tipos de eventos e seus casos de usoImplemente a AWS. DiscoverDevices operação, consulteImplemente a AWS. SendCommand operação, Envie eventos do dispositivo com a SendConnectorEvent API e.

Por exemplo, se seu conector C2C receber uma `DiscoverDevices` solicitação, as integrações gerenciadas do AWS IoT Device Management esperam que ele responda de forma síncrona com o formato de resposta definido acima. Em seguida, você deve invocar a `SendConnectorEvent` API com a estrutura de solicitação definida emImplemente a AWS. DiscoverDevices operação, para um evento DEVICE_DISCOVERY. A chamada `SendConnectorEvent` na API pode ocorrer em qualquer lugar em que você tenha acesso às suas credenciais Lambda do conector C2C. Conta da AWS O fluxo de descoberta de dispositivos não é bem-sucedido até que as integrações gerenciadas do AWS IoT Device Management recebam esse evento.

> **Note**
>
> Como alternativa, a chamada da `SendConnectorEvent` API pode ocorrer antes da resposta de invocação Lambda do conector C2C, se necessário. No entanto, esse fluxo contradiz o modelo assíncrono para desenvolvimento de software.

- SendConnectorEvent- Seu conector chama essas integrações gerenciadas para a API AWS IoT Device Management para enviar eventos de dispositivos para integrações gerenciadas para o AWS IoT Device Management. Somente 3 tipos de eventos aceitos pelas integrações gerenciadas:
  - “DEVICE_DISCOVERY" — Essa operação de evento deve ser usada para enviar uma lista de dispositivos descobertos na nuvem de terceiros para um token de acesso específico.
  - “DEVICE_COMMAND_RESPONSE" — Essa operação de evento deve ser usada para enviar um evento de dispositivo específico como resultado da execução do comando.
  - “DEVICE_EVENT" — Esta operação de evento deve ser usada para qualquer evento originado no dispositivo que não seja o resultado direto de um comando baseado no usuário. Isso pode servir como um tipo de evento geral para relatar proativamente as alterações ou notificações do estado do dispositivo.

Implemente a AWS. ActivateUser operação

A `AWS.ActivateUser` operação é necessária para integrações gerenciadas do AWS IoT Device Management para recuperar um identificador de usuário do token .0 de um usuário OAuth2 final. As integrações gerenciadas do AWS IoT Device Management passarão o OAuth token no cabeçalho da

solicitação e esperam que seu conector inclua o identificador de usuário globalmente exclusivo na carga útil da resposta. Essa operação ocorre após um fluxo bem-sucedido de vinculação de contas.

A lista a seguir descreve os requisitos do seu conector para facilitar um fluxo de `AWS.Activate` usuários bem-sucedido.

- Seu conector C2C Lambda pode processar uma mensagem de solicitação de `AWS.ActivateUser` operação de integrações gerenciadas para o AWS IoT Device Management.
- Seu conector C2C Lambda pode determinar um identificador de usuário exclusivo a partir de um token .0 fornecido. OAuth2 Normalmente, ele pode ser extraído do próprio token, se for um token JWT, ou solicitado do servidor de autorização pelo token.

Workflow do **`AWS.ActivateUser`**

1. Integrações gerenciadas para AWS IoT Device Management invocar seu conector C2C Lambda com a seguinte carga útil:

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

2. O conector C2C determina o ID do usuário, a partir do token ou consultando seu servidor de recursos terceirizado, para incluir na resposta. `AWS.ActivateUser`
3. O conector C2C responde à invocação `AWS.ActivateUser` da operação Lambda, incluindo a carga útil padrão e o identificador de usuário correspondente no campo. `userId`

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

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

Implemente a AWS. DiscoverDevices operação

A descoberta de dispositivos alinha a lista de dispositivos físicos de propriedade do usuário final com as representações digitais desses dispositivos do usuário final mantidas em integrações gerenciadas para o AWS IoT Device Management. Ela é executada por um AWS cliente em dispositivos de propriedade do usuário final somente após a conclusão da vinculação da conta entre o usuário e as integrações gerenciadas para o AWS IoT Device Management. A descoberta de dispositivos é um processo assíncrono em que integrações gerenciadas para o AWS IoT Device Management chamam um conector para iniciar a solicitação de descoberta do dispositivo. Um conector C2C retorna uma lista de dispositivos de usuário final descobertos de forma assíncrona com um identificador de referência (chamado de`deviceDiscoveryId`) gerado por integrações gerenciadas.

O diagrama a seguir ilustra o fluxo de trabalho de descoberta de dispositivos entre o usuário final e as integrações gerenciadas do AWS IoT Device Management:

AWS. DiscoverDevices fluxo de trabalho

1. O cliente inicia o processo de descoberta do dispositivo em nome do usuário final.
2. As integrações gerenciadas AWS IoT Device Management geram um identificador de referência chamado `deviceDiscoveryId` para a solicitação de descoberta do dispositivo gerada pelo AWS Cliente.
3. Integrações gerenciadas para AWS IoT Device Management enviar uma solicitação de descoberta de dispositivo ao conector C2C usando a interface de `AWS.DiscoverDevices` operação, incluindo uma válida OAuth `accessToken` do usuário final e da. `deviceDiscoveryId`
4. Suas lojas de `deviceDiscoveryId` conectores serão incluídas no `DEVICE_DISCOVERY` evento. Esse evento também conterá uma lista dos dispositivos do usuário final descobertos e deverá ser enviado para integrações gerenciadas do AWS IoT Device Management com `SendConnectorEvent` a API como `DEVICE_DISCOVERY` um evento.

5. Seu conector C2C deve chamar o servidor de recursos para buscar todos os dispositivos pertencentes ao usuário final.
6. Seu conector C2C Lambda responde à invocação do Lambda `invokeFunction` () com a resposta ACK de volta às integrações gerenciadas do AWS IoT Device Management, atuando como a resposta inicial para a operação. `AWS.DiscoverDevices` As integrações gerenciadas notificam o cliente com um ACK sobre o processo iniciado de descoberta de dispositivos.
7. Seu servidor de recursos envia a você uma lista de dispositivos pertencentes e operados pelo usuário final.
8. Seu conector converte cada dispositivo do usuário final nas integrações gerenciadas para o formato de dispositivo necessário do AWS IoT Device Management`ConnectorDeviceId`, `ConnectorDeviceName` incluindo um relatório de capacidade para cada dispositivo.
9. O conector C2C também fornece informações sobre o `UserId` proprietário do dispositivo descoberto. Ele pode ser recuperado do seu servidor de recursos como parte da lista de dispositivos ou em uma chamada separada, dependendo da implementação do servidor de recursos.
10. Em seguida, seu conector C2C chamará as integrações gerenciadas para a API `SendConnectorEvent` AWS IoT Device Management via SigV4 Conta da AWS usando credenciais e com o parâmetro de operação definido como “DEVICE_DISCOVERY”. Cada dispositivo na lista de dispositivos enviados para integrações gerenciadas do AWS IoT Device Management será representado por parâmetros específicos do dispositivo, como, `connectorDeviceId` e a. `connectorDeviceName` `capabilityReport`

    - Com base na resposta do seu servidor de recursos, você precisa notificar adequadamente as integrações gerenciadas do AWS IoT Device Management.

      Por exemplo, se seu servidor de recursos estiver tendo uma resposta paginada à lista de dispositivos descobertos de um usuário final, para cada pesquisa você poderá enviar um evento de `DEVICE_DISCOVERY` operação individual, com um `statusCode` parâmetro de. `3xx` Se a descoberta do dispositivo ainda estiver em andamento, repita as etapas 5, 6 e 7.
11. Integrações gerenciadas para AWS IoT Device Management enviar uma notificação ao cliente sobre a descoberta dos dispositivos do usuário final.
12. Se seu conector C2C enviar um evento de `DEVICE_DISCOVERY` operação com o `statusCode` parâmetro atualizado com um valor de 200, as integrações gerenciadas notificarão o cliente sobre a conclusão do fluxo de trabalho de descoberta do dispositivo.

> **Important**
>
> As etapas 7 a 11 podem ocorrer antes da etapa 6, se desejado. Por exemplo, se sua plataforma de terceiros tiver uma API para listar os dispositivos de um usuário final, o evento DEVICE_DISCOVERY poderá ser enviado `SendConnectorEvent` antes que o conector C2C Lambda responda com o ACK típico.

Requisitos do conector C2C para descoberta de dispositivos

A lista a seguir descreve os requisitos do seu conector C2C para facilitar a descoberta bem-sucedida do dispositivo.

- O conector C2C Lambda a pode processar uma mensagem de solicitação de descoberta de dispositivo a partir de integrações gerenciadas para o AWS IoT Device Management e lidar com a operação. `AWS.DiscoverDevices`
- Seu conector C2C pode chamar as integrações gerenciadas para o AWS IoT Device APIs Management via SigV4 usando as credenciais do usuário para registrar o conector. Conta da AWS

Processo de descoberta de dispositivos

As etapas a seguir descrevem o processo de descoberta de dispositivos com seu conector C2C e integrações gerenciadas para o AWS IoT Device Management.

Processo de descoberta de dispositivos

1. As integrações gerenciadas acionam a descoberta do dispositivo:
   - Envie uma solicitação POST para `DiscoverDevices` com a seguinte carga JSON:

```
/DiscoverDevices
{
   "header": {
        "auth": {
            "token": "ashriu32yr97feqy7afsaf",
            "type": "OAuth2.0"
        }
   },
   "payload": {
```

```
        "operationName": "AWS.DiscoverDevices",
        "operationVersion": "1.0",
        "connectorId": "Your-Connector-Id",
        "deviceDiscoveryId": "12345678"
    }
}
```

2. O conector reconhece a descoberta:

   - O conector envia uma confirmação com a seguinte resposta JSON:

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

3. O conector envia o evento de descoberta do dispositivo:

   - Envie uma solicitação POST para `/connector-event/{your_connector_id}` com a seguinte carga JSON:

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

Construa um CapabilityReport para o evento DISCOVER_DEVICES

Conforme visto na estrutura de eventos definida acima, cada dispositivo relatado em um evento DISCOVER_DEVICES, servindo como resposta a uma `AWS.DiscoverDevices` operação, exigirá CapbilityReport a descrição dos recursos do dispositivo correspondente. Um `CapabilityReport` indica integrações gerenciadas para os recursos do dispositivo AWS IoT Device Management em um formato compatível com Matter. Os seguintes campos devem ser fornecidos no ` CapabilityReport`:

- `nodeId`, String: identificador do nó do dispositivo contendo o seguinte `endpoints`
- `version`, String: versão deste nó do dispositivo, definida pelo desenvolvedor do conector
- `endpoints`, Lista<Cluster>: Lista de AWS implementações do Matter Data Model suportadas por esse endpoint de dispositivo.
  - `id`, String: identificador de endpoint definido pelo desenvolvedor do conector
  - `deviceTypes`, Lista<String>: lista dos tipos de dispositivos que esse endpoint captura, ou seja, “Câmera”.
  - `clusters`, Lista<Cluster>: Lista de AWS implementações do Matter Data Model que esse endpoint suporta.
    - `id`, String: identificador de cluster conforme definido pelo padrão Matter.
    - `revision`, Integer: número da revisão do cluster conforme definido pelo padrão Matter.

- `attributes`, Mapa<String, Object>: Mapa dos identificadores de atributos e seus valores correspondentes do estado atual do dispositivo, com identificadores e valores válidos definidos pelo padrão da matéria.
  - `id`, String: ID do atributo conforme definido pelas AWS implementações do Matter Data Model.
  - `value`, Objeto: o valor atual do atributo definido pelo ID do atributo. O tipo de 'valor' pode mudar dependendo do atributo. O `value` campo é opcional para cada atributo e só deve ser incluído se o conector lambda puder determinar o estado atual durante a descoberta.
- `commands`, List<String>: Lista de comandos IDs compatíveis com esse cluster, conforme definido pelo padrão Matter.
- `events`, Lista<String>: Lista de eventos IDs compatíveis com esse cluster, conforme definido pelo padrão Matter.

Para obter a lista atual de recursos suportados e suas AWS implementações correspondentes do Matter Data Model, consulte a versão mais recente da documentação do Data Model.

Implemente a AWS. SendCommand operação

A `AWS.SendCommand` operação permite integrações gerenciadas para que o AWS IoT Device Management envie comandos iniciados pelo usuário final por meio AWS do cliente para seu servidor de recursos. Seu servidor de recursos pode oferecer suporte a vários tipos de dispositivos, onde cada tipo tem seu próprio modelo de resposta. A execução de comandos é um processo assíncrono em que integrações gerenciadas para o AWS IoT Device Management enviam uma solicitação de execução de comando com um `traceID`, que seu conector incluirá em uma resposta de comando enviada de volta às integrações gerenciadas por meio da API ``. SendConnectorEvent As integrações gerenciadas do AWS IoT Device Management esperam que o servidor de recursos retorne uma resposta confirmando que o comando foi recebido, mas não necessariamente indicando que o comando foi recebido executado.

O diagrama a seguir ilustra o fluxo de execução do comando com um exemplo em que o usuário final tenta acender as luzes de sua casa:

Fluxo de trabalho de execução de comandos do

1. Um usuário final envia um comando para acender uma luz usando o aplicativo do AWS cliente.

2. O cliente retransmite as informações do comando para integrações gerenciadas do AWS IoT Device Management com as informações do dispositivo do usuário final.
3. As integrações gerenciadas geram o “TraceID” que seu conector usará ao enviar respostas de comando de volta ao serviço.
4. integrações gerenciadas para o AWS IoT Device Management enviam a solicitação de comando para o seu conector, usando `AWS.SendCommand` a interface de operação.

    - A carga definida por essa interface consiste no identificador do dispositivo, nos comandos do dispositivo formulados como Matterendpoints/clusters/commands, no token de acesso do usuário final e em outros parâmetros necessários.

5. Seu conector armazena o `traceId` a ser incluído na resposta do comando.

    - Seu conector traduz a solicitação de comando de integrações gerenciadas para o formato apropriado do seu servidor de recursos.

6. Seu conector `UserId` obtém o token de acesso do usuário final fornecido e o associa ao comando.

    a. Eles `UserId` podem ser recuperados do seu servidor de recursos usando uma chamada separada ou extraídos do token de acesso no caso de JWT e tokens similares.

    b. A implementação depende dos detalhes do servidor de recursos e do token de acesso.

7. Seu conector chama o servidor de recursos para “Ligar” a luz do usuário final.
8. O servidor de recursos interage com o dispositivo.

    a. O conector retransmite às integrações gerenciadas do AWS IoT Device Management que o servidor de recursos entregou o comando, respondendo com um ACK como resposta inicial síncrona do comando.

    b. Em seguida, as integrações gerenciadas as retransmitem para o aplicativo do cliente.

9. Depois que o dispositivo acende a luz, esse evento do dispositivo é capturado pelo seu servidor de recursos.
10. Seu servidor de recursos envia o evento do dispositivo para o conector.
11. Seu conector transforma o evento do dispositivo gerado pelo servidor de recursos em integrações gerenciadas do tipo de operação do evento DEVICE_COMMAND_RESPONSE.
12. Seu conector chama a `SendConnectorEvent` API com operação como “DEVICE_COMMAND_RESPONSE”.

- Ele anexa o `traceId` fornecido pelas integrações gerenciadas para o AWS IoT Device Management na solicitação inicial.

13. As integrações gerenciadas notificam o cliente sobre a mudança no estado do dispositivo do usuário final.
14. O cliente notifica o usuário final de que a luz do dispositivo está acesa.

> **Note**
>
> A configuração do servidor de recursos determina a lógica para lidar com mensagens de resposta e solicitação de comando do dispositivo com falha. Isso inclui tentativas de repetição de mensagens usando o mesmo ID de referência para o comando.

Requisitos do conector C2C para execução de comandos do dispositivo

A lista a seguir descreve os requisitos do conector C2C para facilitar a execução bem-sucedida do comando do dispositivo.

- O conector C2C Lambda pode processar mensagens de solicitação de `AWS.SendCommand` operação de integrações gerenciadas para o AWS IoT Device Management.
- Seu conector C2C deve acompanhar os comandos enviados ao seu servidor de recursos e mapeá-los com o `traceID` apropriado.
- Você pode chamar integrações gerenciadas para as APIs do serviço AWS IoT Device Management via SigV4 AWS usando as credenciais usadas para registrar o conector Conta da AWS C2C.

1. As integrações gerenciadas enviam o comando para o conector (consulte a etapa 4 no diagrama anterior).

   - ```
     /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. Comando ACK do conector C2C (consulte a etapa 7 no diagrama anterior, onde o conector envia ACK para as integrações gerenciadas do AWS IoT Device Management Service).

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

3. O conector envia o evento Device Command Response (consulte a etapa 11 no diagrama anterior).

   - ```
     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**
>
> As alterações no estado do dispositivo como resultado da execução de um comando não serão refletidas nas integrações gerenciadas do AWS IoT Device Management até que o evento DEVICE_COMMAND_RESPONSE correspondente seja recebido por meio da API. SendConnectorEvent Isso significa que até que as integrações gerenciadas recebam o evento da etapa 3 anterior, independentemente de sua resposta de invocação do conector indicar sucesso ou não, o estado do dispositivo não será atualizado.

Interpretar “endpoints” de assuntos incluídos na AWS. SendCommand pedido

As integrações gerenciadas usarão os recursos do dispositivo relatados durante a descoberta do dispositivo para determinar quais comandos um dispositivo pode aceitar. Cada capacidade do dispositivo é modelada por meio de AWS implementações do Matter Data Model; portanto, todos

os comandos recebidos serão derivados do campo `comandos` em um determinado cluster. É responsabilidade do seu conector analisar o campo `endpoints`, determinar o comando Matter correspondente e traduzi-lo de forma que o comando correto chegue ao dispositivo. Normalmente, isso significa traduzir o modelo de dados do Matter nas solicitações de API relacionadas.

Depois que o comando for executado, seu conector determina quais `atributos` definidos pelas AWS implementações do Matter Data Model foram alterados como resultado. Essas alterações são então reportadas às integrações gerenciadas do AWS IoT Device Management por meio de eventos da API DEVICE_COMMAND_RESPONSE enviados com a API. `SendConnectorEvent`

Considere o campo `endpoints` incluído no seguinte exemplo de carga útil: `AWS.SendCommand`

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

A partir desse objeto, o conector pode determinar o seguinte:

1. Defina as informações do endpoint e do cluster:
   a. Defina o endpoint `id` como “1".

      > **Note**
      >
      > Se um dispositivo definir vários endpoints, como um único cluster (comoOn/Off) can control multiple capabilities (i.e. turn a light on/off as well as turning a strobe on/off), esse ID será usado para rotear o comando para o recurso correto.

   b. Defina o cluster `id` como “0x0202" (cluster de controle de ventilador).
2. Defina as informações do comando:

a. Defina o identificador do comando como “0xff01" (comando Atualizar estado definido por). AWS

b. Atualize os identificadores de atributos incluídos com os valores fornecidos na solicitação.

3. Atualize o atributo:

a. Defina o identificador do atributo como “0x0000" (FanMode atributo do cluster de controle do ventilador).

b. Defina o valor do atributo como “3" (Alta velocidade do ventilador).

As integrações gerenciadas definiram dois tipos de comando “personalizados” que não são estritamente definidos pelas AWS implementações do Matter Data Model: os comandos ReadState e. UpdateState Para obter e definir atributos de cluster definidos pelo Matter, as integrações gerenciadas enviarão ao seu conector uma `AWS.SendCommand` solicitação com o comando IDs referente a UpdateState (id: 0xff01) ou ReadState (id: 0xff02), com os parâmetros correspondentes dos atributos que devem ser atualizados ou lidos. Esses comandos podem ser invocados para QUALQUER tipo de dispositivo para atributos definidos como mutáveis (atualizáveis) ou recuperáveis (legíveis) a partir da AWS implementação correspondente do Matter Data Model.

Envie eventos do dispositivo com a SendConnectorEvent API

Visão geral dos eventos iniciados pelo dispositivo

Embora a `SendConnectorEvent` API seja usada para responder `AWS.SendCommand` e `AWS.DiscoverDevices` operar de forma assíncrona, ela também é usada para notificar integrações gerenciadas sobre quaisquer eventos iniciados pelo dispositivo. Os eventos iniciados pelo dispositivo podem ser definidos como qualquer evento gerado por um dispositivo sem um comando iniciado pelo usuário. Esses eventos do dispositivo podem incluir, mas não estão limitados a, mudanças de estado do dispositivo, detecção de movimento, níveis de bateria e muito mais. Você pode enviar esses eventos de volta às integrações gerenciadas usando a `SendConnectorEvent` API com a operação DEVICE_EVENT.

A seção a seguir usa uma câmera inteligente instalada em uma casa como exemplo para explicar melhor o fluxo de trabalho desses eventos:

Fluxo de trabalho de eventos do

1. Sua câmera detecta movimento para o qual gera um evento que é enviado ao seu servidor de recursos.
2. Seu servidor de recursos processa o evento e o envia para seu conector C2C.
3. Seu conector traduz esse evento para as integrações gerenciadas da interface AWS IoT Device Management. `DEVICE_EVENT`
4. Seu conector C2C envia esse evento de dispositivo para integrações gerenciadas usando a `SendConnectorEvent` API com a operação definida como “DEVICE_EVENT”.
5. As integrações gerenciadas identificam o cliente relevante e retransmitem esse evento para o cliente.
6. O cliente recebe esse evento e o exibe ao usuário por meio do identificador do usuário.

Para obter mais informações sobre a operação da `SendConnectorEvent` API, consulte o `SendConnectorEvent` Guia de referência da API de integrações gerenciadas do AWS IoT Device Management.

Requisitos de eventos iniciados pelo dispositivo

A seguir estão alguns requisitos para eventos iniciados pelo dispositivo.

- Seu recurso de conector C2C deve ser capaz de receber eventos de dispositivo assíncronos do seu servidor de recursos
- Seu recurso de conector C2C deve ser capaz de chamar integrações gerenciadas para as APIs do serviço AWS IoT Device Management via SigV4 AWS usando as credenciais usadas para registrar o conector C2C. Conta da AWS

O exemplo a seguir demonstra um conector enviando um evento originado pelo dispositivo por meio da API: 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"
                 }
             ]
         }]
     }]
}
```

No exemplo a seguir, vemos o seguinte:

- Isso vem do endpoint do dispositivo com ID igual a 1.
- A capacidade do dispositivo à qual esse evento se refere tem um ID de cluster de 0x0202, pertencente ao cluster de assuntos do Fan Control.
- O atributo que foi alterado tem o ID de 0x000, pertencente ao Fan Mode Enum dentro do cluster. Ele foi atualizado para o valor 3, referente ao valor de Alto.
- Como `connectorId` é um parâmetro retornado pelo serviço de nuvem na criação, os conectores devem consultar usando GetCloudConnector e filtrar por`lambdaARN`. O próprio lambda `ARN` é consultado usando `Lambda.get_function_url_config` a API. Isso permite que `CloudConnectorId` o seja acessado dinamicamente no lambda e não configurado estaticamente como antes.

Implemente a AWS. DeactivateUser operação

Visão geral da desativação do usuário

A desativação dos tokens de acesso do usuário fornecidos é necessária quando um cliente exclui sua conta de AWS cliente ou quando um usuário final gostaria de desvincular sua conta no sistema do sistema do AWS cliente. Em qualquer caso de uso, as integrações gerenciadas precisam facilitar esse fluxo de trabalho usando o conector C2C.

A imagem abaixo ilustra a desvinculação de uma conta de usuário final do sistema

Fluxo de trabalho de desativação do usuário

1. O usuário inicia o processo de desvinculação entre a conta do AWS cliente e o servidor de autorização de terceiros associado ao conector C2C.
2. O cliente inicia a exclusão da associação do usuário por meio de integrações gerenciadas para o AWS IoT Device Management.
3. As integrações gerenciadas iniciam o processo de desativação por meio de solicitação ao seu conector usando a `AWS.DeactivateUser` interface de operação.
    - O token de acesso do /user está incluído no cabeçalho da solicitação.
4. Seu conector C2C aceita a solicitação e invoca seu servidor de autorização para revogar o token e qualquer acesso que ele forneça.
    - Por exemplo, eventos de uma conta de usuário desvinculada não devem mais ser enviados para integrações gerenciadas após a execução. `AWS.DeactivateUser`
5. Seu servidor de autorização revoga o acesso e envia uma resposta de volta ao seu conector C2C.
6. Seu conector C2C envia integrações gerenciadas para o AWS IoT Device Management um ACK informando que o token de acesso do usuário foi revogado.
7. As integrações gerenciadas excluem todos os recursos de propriedade do usuário final que foram associados ao seu servidor de recursos.
8. As integrações gerenciadas enviam um ACK ao cliente, informando que todas as associações relacionadas ao seu sistema foram excluídas.
9. O cliente notifica o usuário final de que sua conta foi desvinculada da sua plataforma.

AWS. DeactivateUser requisitos

- A função Lambda do conector C2C recebe uma mensagem de solicitação de integrações gerenciadas para lidar com a operação. `AWS.DeactivateUser`
- O conector C2C deve revogar o token OAuth2.0 fornecido e o token de atualização correspondente do usuário em seu servidor de autorização.

Veja a seguir um exemplo de `AWS.DeactivateUser` solicitação que seu conector receberá:

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

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

Invoque seu conector C2C

AWS Lambda permite que políticas baseadas em recursos autorizem quem pode invocar um Lambda. Como as integrações gerenciadas para o AWS IoT Device Management são AWS service (Serviço da AWS) uma, você deve permitir que as integrações gerenciadas invoquem seu conector C2C Lambda por meio da política de recursos.

Anexe uma política de recursos com pelo menos as seguintes permissões mínimas ao seu conector C2C Lambda. Isso fornece integrações gerenciadas com privilégios de invocação da função Lambda. Essa política inclui uma `Condition` chave para ajudar você a limitar sua usabilidade somente `connectorId` aos usuários pretendidos.

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"
        }
      }
    }
  ]
}
```

Adicione permissões à sua função do IAM

Todas as integrações gerenciadas APIs exigem autenticação AWS SigV4 para serem invocadas. O SigV4 é um protocolo de assinatura para autenticar solicitações de AWS API usando suas credenciais. Conta da AWS A função do IAM que você usa para invocar as integrações gerenciadas APIs deve ter as seguintes permissões para poder invocar com êxito o: APIs

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

Para obter informações adicionais sobre como adicionar essas permissões, entre em contato com Suporte.

Recursos adicionais do

Para registrar seu conector C2C, você precisará do seguinte:

- O ARN do Lambda que designa o conector que você gostaria de registrar.

Teste manualmente seu conector C2C

Para testar manualmente seu conector C2C end-to-end, você deve simular o cliente e o usuário final.

Você precisará dos seguintes recursos:

- Um AWS Lambda ARN designando o conector que você gostaria de testar.
- Uma conta de usuário OAuth 2.0 de teste da sua plataforma de nuvem.
- Um conector registrado com integrações gerenciadas para o AWS IoT Device Management. Para obter mais informações, consulte Use um conector C2C (Cloud-to-Cloud).

Use um conector C2C (Cloud-to-Cloud)

Um conector C2C gerencia a tradução de mensagens de solicitação e resposta e permite a comunicação entre integrações gerenciadas e uma nuvem de fornecedores terceirizados. Ele facilita o controle unificado em diferentes tipos de dispositivos, plataformas e protocolos, permitindo que dispositivos de terceiros sejam integrados e gerenciados.

O procedimento a seguir lista as etapas para usar o conector C2C.

Etapas para usar o conector C2C:

1. CreateCloudConnector

   Configure um conector para permitir a comunicação bidirecional entre suas integrações gerenciadas e nuvens de fornecedores terceirizados.

   Ao configurar o conector, forneça os seguintes detalhes:

   - Nome: escolha um nome descritivo para o conector.
   - Descrição: Forneça um breve resumo da finalidade e das capacidades do conector.
   - AWS Lambda ARN: especifique o Amazon Resource Name (ARN) da AWS Lambda função que alimentará o conector.

   Crie e implante uma AWS Lambda função que se comunica com um fornecedor terceirizado APIs para criar um conector. Em seguida, chame a CreateCloudConnectorAPI nas integrações gerenciadas e forneça o ARN da AWS Lambda função para registro. Certifique-se de que a AWS Lambda função seja implantada na mesma AWS conta em que você criou o conector

nas integrações gerenciadas. Você receberá um ID de conector exclusivo para identificar a integração.

Exemplo de solicitação e resposta de CreateCloudConnector API:

```
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"
}
```

Fluxo de criação:

> **Note**
>
> Use o GetCloudConnector, UpdateCloudConnectorDeleteCloudConnector, e ListCloudConnectors APIs conforme necessário para esse procedimento.

2. CreateConnectorDestination

Configure os destinos para fornecer as configurações e as credenciais de autenticação que os conectores precisam para estabelecer conexões seguras com nuvens de fornecedores terceirizados. Use Destinations para registrar suas credenciais de autenticação de terceiros com integrações gerenciadas, como detalhes de autorização OAuth 2.0, incluindo o URL de autorização, o esquema de autenticação e a localização das credenciais. AWS Secrets Manager

Pré-requisitos

Antes de criar um ConnectorDestination, você deve:

- Chame a CreateCloudConnectorAPI para criar um conector. O ID que a função retorna é usado na chamada CreateConnectorDestinationda API API.
- Recupere o `tokenUrl` para a plataforma 3P do conector. (Você pode trocar um AuthCode por um AccessToken).
- Recupere o AuthURL para a plataforma 3P do conector. (Os usuários finais podem se autenticar usando seu nome de usuário e senha).
- Use o `clientId` e `clientSecret` (da plataforma 3P) no gerenciador secreto da sua conta.

Exemplo de solicitação e resposta de CreateConnectorDestination API:

```
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"
}
```

Fluxo de criação de destinos na nuvem:

> **Note**
>
> Use o GetCloudConnector, UpdateCloudConnectorDeleteCloudConnector, e ListCloudConnectors APIs conforme necessário para esse procedimento.

3. CreateAccountAssociation

As associações representam os relacionamentos entre as contas de nuvem de terceiros dos usuários finais e um destino de conector. Depois de criar uma associação e vincular os usuários finais às integrações gerenciadas, seus dispositivos podem ser acessados por meio de uma ID de associação exclusiva. Essa integração permite três funções principais: descobrir dispositivos, enviar comandos e receber eventos.

Pré-requisitos

Antes de criar um, AccountAssociationvocê deve concluir o seguinte:

- Chame a CreateConnectorDestinationAPI para criar um destino. O ID que a função retorna é usado na chamada da CreateAccountAssociationAPI.
- Invoque a API CreateAccountAssociation.

Exemplo de solicitação e resposta de CreateAccountAssociation API:

```
Request:

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

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

Response:

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

> **Note**
>
> Use o GetCloudConnector, UpdateCloudConnectorDeleteCloudConnector, e ListCloudConnectors APIs conforme necessário para esse procedimento.

An AccountAssociationtem um estado que é consultado de GetAccountAssociatione. ListAccountAssociations APIs Isso APIs mostra o estado da Associação. A StartAccountAssociationRefreshAPI permite a atualização de um AccountAssociationestado quando seu token de atualização expira.

4. Descoberta de dispositivos

   Cada item gerenciado está vinculado a detalhes específicos do dispositivo, como seu número de série e um modelo de dados. O modelo de dados descreve a funcionalidade do dispositivo, indicando se é uma lâmpada, interruptor, termostato ou outro tipo de dispositivo. Para descobrir um dispositivo 3P e criar um ManagedThing para o dispositivo 3P, você deve seguir as etapas abaixo em sequência.

   a. Chame a StartDeviceDiscoveryAPI para iniciar o processo de descoberta do dispositivo.

      Exemplo de solicitação e resposta de StartDeviceDiscovery API:

      ```
      Request:

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

```
Response:

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

b. Invoque a GetDeviceDiscoveryAPI para verificar o status do processo de descoberta.

c. Invoque a ListDiscoveredDevicesAPI para listar os dispositivos descobertos.

Exemplo de solicitação e resposta de ListDiscoveredDevices API:

```
Request:

//Empty body

Response:

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

d. Invoque a CreateManagedThingAPI para selecionar os dispositivos da lista de descoberta a serem importados para integrações gerenciadas.

Exemplo de solicitação e resposta de CreateManagedThing API:

```
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. Invoque a GetManagedThingAPI para ver isso recém-criado`managedThing`. O status será`UNASSOCIATED`.

f. Invoque a RegisterAccountAssociationAPI para associá-la `managedThing` a um específico`accountAssociation`. Ao final de uma RegisterAccountAssociationAPI bem-sucedida, o `ASSOCIATED` estado `managedThing` muda.

Exemplo de solicitação e resposta de RegisterAccountAssociation API:

```
Request:

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

Response:

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

5. Envie um comando para o dispositivo 3P

Para controlar um dispositivo recém-integrado, use a SendManagedThingCommandAPI, com o ID de associação criado anteriormente e uma ação de controle com base na capacidade suportada pelo dispositivo. O conector usa credenciais armazenadas do processo de vinculação de contas para se autenticar na nuvem de terceiros e invocar a chamada de API relevante para a operação.

Exemplo de solicitação e resposta de SendManagedThingCommand API:

```
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"
}
```

Envie o comando para o fluxo do dispositivo 3P:

6. O conector envia eventos para integrações gerenciadas

   A SendConnectorEventAPI captura quatro tipos de eventos, do conector às integrações gerenciadas, representados pelos seguintes valores de enumeração para o parâmetro Operation Type:

   - DEVICE_COMMAND_RESPONSE: a resposta assíncrona que o conector envia em resposta a um comando.
   - DEVICE_DISCOVERY: em resposta a um processo de descoberta de dispositivos, o conector envia a lista de dispositivos descobertos para integrações gerenciadas e usa a API. SendConnectorEvent
   - DEVICE_EVENT: envia os eventos recebidos do dispositivo.
   - DEVICE_COMMAND_REQUEST: solicitações de comando iniciadas a partir do dispositivo. Por exemplo, fluxos de trabalho do WebRTC.

   O conector também pode encaminhar eventos do dispositivo usando a SendConnectorEventAPI, com um `userId` parâmetro opcional.

   - Para eventos de dispositivos com`userId`:

     Exemplo de solicitação e resposta de SendConnectorEvent API:

```
Request:

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

```
            }
        }]
    }
}

Response:

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

- Para eventos de dispositivos sem`userId`:

  Exemplo de solicitação e resposta de SendConnectorEvent API:

```
Request:

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

Response:

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

Para remover o vínculo entre uma associação específica `managedThing` e uma conta, use o mecanismo de cancelamento de registro:

Exemplo de solicitação e resposta de DeregisterAccountAssociation API:

```
Request:

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

Response:

HTTP/1.1 200 // Empty body
```

Enviar fluxo de eventos:

7. Atualize o status do conector para “Listado” para torná-lo visível para outros clientes de integrações gerenciadas

   Por padrão, os conectores são privados e visíveis somente para a AWS conta que os criou. Você pode optar por tornar um conector visível para outros clientes de integrações gerenciadas.

   Para compartilhar seu conector com outros usuários, use a opção Tornar visível Console de gerenciamento da AWS na página de detalhes do conector para enviar seu ID de conector AWS para análise. Depois de aprovado, o conector fica disponível para todos os usuários de integrações gerenciadas no mesmo Região da AWS. Além disso, você pode restringir o acesso a AWS uma conta específica IDs modificando a política de acesso na AWS Lambda função associada ao conector. Para garantir que seu conector possa ser usado por outros clientes, gerencie as permissões de acesso do IAM em sua função Lambda de AWS outras contas para seu conector visível.

   Analise os AWS service (Serviço da AWS) termos e as políticas da sua organização que regem o compartilhamento de conectores e as permissões de acesso antes de tornar os conectores visíveis para outros clientes de integrações gerenciadas.

Integrações gerenciadas Hub SDK

Use os tópicos desta seção para aprender como integrar e controlar dispositivos do hub de IoT usando o SDK do Hub de integrações gerenciadas. Para obter mais informações sobre as integrações gerenciadas End Device SDK, consulte o. Integrações gerenciadas SDK para dispositivos finais

Arquitetura do Hub SDK

Integração de dispositivos

Analise como os componentes do Hub SDK oferecem suporte à integração de dispositivos antes de começar a trabalhar com integrações gerenciadas. Esta seção aborda os componentes arquitetônicos essenciais necessários para a integração de dispositivos, incluindo como o provisionador principal e os plug-ins específicos do protocolo trabalham juntos para lidar com a autenticação, a comunicação e a configuração do usuário do dispositivo.

Componentes do Hub SDK para integração de dispositivos

Componentes do SDK

- Provisionador principal
- Plug-ins de provisionamento específicos do protocolo
- Middleware específico de protocolo

Provisionador principal

O provisionador principal é o componente central que orquestra a integração de dispositivos na implantação do hub de IoT. Ele coordena toda a comunicação entre as integrações gerenciadas e seus plug-ins de provisionador específicos do protocolo, garantindo a integração segura e confiável do dispositivo. Quando você integra um dispositivo, o provisionador principal gerencia o fluxo de autenticação, gerencia as mensagens MQTT e processa as solicitações do dispositivo por meio dessas funções:

Conexão MQTT

Cria conexões com o corretor MQTT para publicação e assinatura de tópicos na nuvem.

Fila de mensagens e manipulador

Processa as solicitações de adição e remoção de dispositivos recebidas em sequência.

Interface de plug-in de protocolo

Funciona com plug-ins de provisionamento específicos de protocolo para integração de dispositivos, gerenciando os modos de autenticação e junção de rádio.

Cliente Hub SDK APIs

Receba e encaminhe relatórios de capacidade do dispositivo de plug-ins CDMB específicos do protocolo para integrações gerenciadas.

Plug-ins de provisionamento específicos do protocolo

Os plug-ins de provisionamento específicos do protocolo são bibliotecas que gerenciam a integração de dispositivos para diferentes protocolos de comunicação. Cada plug-in traduz comandos do provisionador principal em ações específicas de protocolo para seus dispositivos de IoT. Esses plug-ins executam:

- Inicialização de middleware específica do protocolo
- Configuração do modo de junção de rádio com base nas solicitações principais do provisionador
- Remoção de dispositivos por meio de chamadas de API de middleware

Middleware específico de protocolo

O middleware específico do protocolo atua como uma camada de tradução entre os protocolos do seu dispositivo e as integrações gerenciadas. Esse componente processa a comunicação em ambas as direções: recebendo comandos dos plug-ins do provisionador e enviando-os para pilhas de protocolos, além de coletar respostas dos dispositivos e roteá-las de volta pelo sistema.

Fluxos de integração de dispositivos

Revise a sequência de operações que ocorrem quando você integra dispositivos usando o SDK do Hub. Esta seção mostra como os componentes interagem durante o processo de integração e descreve os métodos de integração suportados.

Fluxos de integração

- Configuração simples (SS)
- Configuração sem toque (ZTS)
- Configuração guiada pelo usuário (UGS)

Configuração simples (SS)

O usuário final liga o dispositivo de IoT e escaneia seu código QR usando o aplicativo do fabricante do dispositivo. O dispositivo é então inscrito na nuvem de integrações gerenciadas e se conecta ao hub de IoT.

Configuração sem toque (ZTS)

A configuração Zero-touch (ZTS) simplifica a integração do dispositivo ao pré-associar o dispositivo a montante na cadeia de suprimentos. Por exemplo, em vez de os usuários finais digitalizarem o código QR do dispositivo, essa etapa é concluída anteriormente para pré-vincular os dispositivos às contas dos clientes. Por exemplo, essa etapa pode ser concluída no centro de distribuição.

Quando o usuário final recebe e liga o dispositivo, ele se inscreve automaticamente na nuvem de integrações gerenciadas e se conecta ao hub de IoT sem exigir nenhuma ação adicional de configuração.

Configuração guiada pelo usuário (UGS)

O usuário final liga o dispositivo e segue as etapas interativas para integrá-lo às integrações gerenciadas. Isso pode incluir pressionar um botão no hub de IoT, usar um aplicativo do fabricante do dispositivo ou pressionar botões no hub e no dispositivo. Você pode usar esse método se a configuração simples falhar.

Controle de dispositivos

As integrações gerenciadas gerenciam o registro de dispositivos, a execução de comandos e o controle. Você pode criar experiências para o usuário final sem conhecer os protocolos específicos do dispositivo usando o gerenciamento de dispositivos independente do fornecedor e do protocolo.

Com o controle do dispositivo, você pode visualizar e modificar os estados do dispositivo, como o brilho da lâmpada ou a posição da porta. O recurso emite eventos para mudanças de estado, que você pode usar para análises, regras e monitoramento.

Atributos principais

Modificar ou ler o estado do dispositivo

Visualize e altere os atributos do dispositivo com base nos tipos de dispositivos. Você pode acessar:

- Estado do dispositivo: valores atuais dos atributos do dispositivo
- Estado de conectividade: status de acessibilidade do dispositivo
- Status de saúde: valores do sistema, como nível da bateria e intensidade do sinal (RSSI)

Notificação de mudança de estado

Receba eventos quando os atributos do dispositivo ou os estados de conectividade mudarem, como ajustes de brilho da lâmpada ou alterações no status da fechadura da porta.

Modo off-line

Os dispositivos se comunicam com outros dispositivos no mesmo hub de IoT, mesmo sem conexão com a Internet. Os estados do dispositivo são sincronizados com a nuvem quando a conectividade é retomada.

Sincronização de estados

Acompanhe as alterações de estado de várias fontes, aplicativos do fabricante do dispositivo e ajustes manuais do dispositivo.

Analise os componentes e processos do Hub SDK necessários para controlar dispositivos por meio de integrações gerenciadas. Este tópico descreve como o Edge Agent, o Common Data Model Bridge (CDMB) e os plug-ins específicos do protocolo trabalham juntos para lidar com comandos de dispositivos, gerenciar estados de dispositivos e processar respostas em diferentes protocolos.

Fluxos de controle de dispositivos

O diagrama a seguir demonstra o fluxo de controle do end-to-end dispositivo descrevendo como um usuário final liga um plugue inteligente Zigbee.

Componentes do Hub SDK para controle de dispositivos

A arquitetura do Hub SDK usa os seguintes componentes para processar e rotear comandos de controle de dispositivos em sua implementação de IoT. Cada componente desempenha um papel específico na tradução dos comandos da nuvem em ações do dispositivo, no gerenciamento dos estados do dispositivo e no tratamento de respostas. As seções a seguir detalham como esses componentes funcionam juntos em sua implantação:

O Hub SDK consiste nos seguintes componentes e facilita a integração e o controle de dispositivos em hubs de IoT.

Componentes primários:

Agente Edge

Atua como um gateway entre o hub de IoT e as integrações gerenciadas.

Ponte comum de modelo de dados (CDMB)

Traduz entre o modelo de AWS dados e os modelos de dados do protocolo local, como Z-Wave e Zigbee. Ele inclui um CDMB principal e plug-ins CDMB específicos do protocolo.

Provisionador

Lida com a descoberta e a integração de dispositivos. Ele inclui um provisionador principal e plug-ins de provisionador específicos do protocolo para tarefas de integração específicas do protocolo.

Componentes secundários

Integração do hub

Provisiona o hub com certificados e chaves de cliente para comunicação segura na nuvem.

Proxy MQTT

Fornece conexões MQTT com a nuvem de integrações gerenciadas.

Logger

Grava registros localmente ou na nuvem de integrações gerenciadas.

Instale e valide as integrações gerenciadas Hub SDK

Escolha entre os seguintes métodos de implantação para instalar o SDK do Hub de integrações gerenciadas em seus dispositivos —AWS IoT Greengrass para implantação automatizada ou instalação manual de scripts. Esta seção descreve as etapas de configuração e validação de ambas as abordagens.

Métodos de implantação

- Instale o Hub SDK com AWS IoT Greengrass
- Implemente o Hub SDK com um script
- Implemente o Hub SDK com systemd

Instale o Hub SDK com AWS IoT Greengrass

Implante os componentes do SDK do Hub de integrações gerenciadas para seus dispositivos usando AWS IoT Greengrass (versão Java).

> **Note**
>
> Você já deve ter configurado e ter uma compreensão do AWS IoT Greengrass. Para obter mais informações, consulte O que está AWS IoT Greengrass na documentação do guia do AWS IoT Greengrass desenvolvedor.

O AWS IoT Greengrass usuário deve ter permissão para modificar os seguintes diretórios:

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

Tópicos

- Implante componentes localmente
- Implantação na nuvem
- Verifique o provisionamento do hub
- Verifique a operação do CDMB

- Verifique a operação do LPW-Provisioner

Implante componentes localmente

Use a CreateDeployment AWS IoT Greengrass API em seu dispositivo para implantar os componentes do Hub SDK. Os números de versão não são estáticos e podem variar de acordo com a versão que você usa no momento. Use o seguinte formato para**`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"
```

Implantação na nuvem

Siga as instruções no guia do AWS IoT Greengrass desenvolvedor para realizar as seguintes etapas:

1. Faça upload de artefatos para o Amazon S3.
2. Atualize as receitas para incluir a localização do artefato Amazon S3.
3. Crie uma implantação em nuvem no dispositivo para os novos componentes.

Verifique o provisionamento do hub

Confirme o sucesso do provisionamento verificando seu arquivo de configuração. Abra o `/data/aws/iotmi/config/iotmi_config.json` arquivo e verifique se o estado está definido como`PROVISIONED`.

Verifique a operação do CDMB

Verifique se há mensagens de inicialização do CDMB e da inicialização bem-sucedida no arquivo de registros. A *logs file* localização pode variar dependendo de onde AWS IoT Greengrass está instalado.

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

Exemplo

```
[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
```

Verifique a operação do LPW-Provisioner

Verifique no arquivo de registros as mensagens de inicialização do LPW-Provisioner e a inicialização bem-sucedida. A *logs file* localização pode variar dependendo de onde AWS IoT Greengrass está instalado.

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

Exemplo

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

Implemente o Hub SDK com um script

Implante os componentes do SDK do Hub de integrações gerenciadas manualmente usando scripts de instalação e, em seguida, valide a implantação. Esta seção descreve as etapas de execução do script e o processo de verificação.

Tópicos

- Prepare seu ambiente
- Execute o script do Hub SDK

- Verifique o provisionamento do hub
- Verifique a operação do agente
- Verifique a operação do LPW-Provisioner

Prepare seu ambiente

Conclua estas etapas antes de executar o script de instalação do SDK:

1. Crie uma pasta com o nome `middleware` dentro da `artifacts` pasta.
2. Copie seus arquivos de middleware do hub para a `middleware` pasta.
3. Execute os comandos de inicialização antes de iniciar o SDK.

> **Important**
>
> Repita os comandos de inicialização após cada reinicialização do hub.

```
#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
```

Execute o script do Hub SDK

Navegue até o diretório de artefatos e execute o `start_iotmi_sdk.sh` script. Esse script inicia os componentes do SDK do hub na sequência correta. Analise os seguintes exemplos de registros para verificar se a inicialização foi bem-sucedida:

> **Note**
>
> Os registros de todos os componentes em execução podem ser encontrados dentro da `artifacts/logs` pasta.

```
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----
```

Verifique o provisionamento do hub

Verifique se o `iot_provisioning_state` campo em `/data/aws/iotmi/config/iotmi_config.json` está definido como`PROVISIONED`.

Verifique a operação do agente

Verifique se há mensagens de inicialização do agente e a inicialização bem-sucedida no arquivo de registros.

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

Exemplo

```
[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**
>
> Verifique se o `iotmi.db` banco de dados existe em seu `artifacts` diretório.

Verifique a operação do LPW-Provisioner

Verifique se há mensagens de inicialização e `LPW-Provisioner` inicialização bem-sucedida no arquivo de registros.

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

O seguinte código mostra um exemplo.

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

Implemente o Hub SDK com systemd

> **Important**
>
> Siga o `readme.md` no `hubSystemdSetup` diretório do arquivo release.tgz para obter as atualizações mais recentes.

Esta seção descreve os scripts e processos para implantar e configurar serviços em um dispositivo hub baseado em Linux.

Visão geral

O processo de implantação consiste em dois scripts principais:

- `copy_to_hub.sh`: é executado na máquina host para copiar os arquivos necessários para o hub
- `setup_hub.sh`: é executado no hub para configurar o ambiente e implantar serviços

Além disso, `systemd/deploy_iotshd_services_on_hub.sh` gerencia a sequência de inicialização do processo e o gerenciamento de permissões do processo e é acionado automaticamente pelo`setup_hub.sh`.

Pré-requisitos

Os pré-requisitos listados são necessários para uma implantação bem-sucedida.

- o serviço systemd está disponível no hub
- Acesso SSH ao dispositivo hub
- Privilégios de Sudo no dispositivo hub
- `scp`utilitário instalado na máquina host
- `sed`utilitário instalado na máquina host
- utilitário de descompactação instalado na máquina host

Estrutura do arquivo

A estrutura de arquivos foi projetada para facilitar a organização e o gerenciamento de seus diversos componentes, permitindo o acesso e a navegação eficientes do conteúdo.

```
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
```

No arquivo tgz da versão do SDK, a estrutura geral do arquivo é:

```
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
```

Configuração inicial do

Extraia o pacote SDK

```
tar -xzf managed-integrations-Hub-SDK-v<Version>-linux-aarch64-<timestamp>.tgz
```

Navegue até o diretório extraído e prepare o pacote:

```
# Create package.zip containing required artifacts
```

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

Adicionar arquivos de configuração do dispositivo

Siga as duas etapas listadas para criar os arquivos de configuração do dispositivo e copiá-los para o hub.

1. Adicione arquivos de configuração do dispositivo para criar os arquivos de configuração do dispositivo necessários. O SDK usa esse arquivo para sua função.
2. Copie os arquivos de configuração para copiar os arquivos de configuração criados para o hub.

Copiar arquivos para o hub

Execute o script de implantação em sua máquina host:

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

Example Exemplo

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

Isso copia:

- O arquivo do pacote (renomeado para package.zip no hub)
- Arquivos de configuração
- Certificados
- Arquivos de serviço Systemd

Configurar hub

Depois que os arquivos forem copiados, faça o SSH no hub e execute o script de configuração:

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

Configurações de usuários e grupos

Por padrão, usamos o hub do usuário e o hub do grupo para os componentes do SDK. Há várias maneiras de configurá-las:

- Use um usuário/grupo personalizado:

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

- Crie-os manualmente antes de executar o script de configuração:

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

- Adicione os comandos em`setup_hub.sh`.

Gerenciar serviços da

Para reiniciar todos os serviços, execute o seguinte script no hub:

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

O script de configuração criará os diretórios necessários, definirá as permissões apropriadas e implantará os serviços automaticamente. Se você não estiver usando SSH/SCP, deverá modificar seu método de implantação `copy_to_hub.sh` específico. Certifique-se de que todos os arquivos e configurações do certificado estejam configurados corretamente antes da implantação.

Integre seus hubs para integrações gerenciadas

Configure seus dispositivos de hub para se comunicarem com integrações gerenciadas configurando a estrutura de diretórios, os certificados e os arquivos de configuração do dispositivo necessários. Esta seção descreve como os componentes do subsistema de integração do hub funcionam juntos, onde armazenar certificados e arquivos de configuração, como criar e modificar o arquivo de configuração do dispositivo e as etapas para concluir o processo de provisionamento do hub.

Subsistema de integração do hub

O subsistema de integração do hub usa esses componentes principais para gerenciar o provisionamento e a configuração do dispositivo:

Componente de integração do hub

Gerencia o processo de integração do hub coordenando o estado do hub, a abordagem de provisionamento e os materiais de autenticação.

Arquivo de configuração do dispositivo

Armazena dados essenciais de configuração do hub no dispositivo, incluindo:

- Estado de provisionamento do dispositivo (provisionado ou não provisionado)
- Certificado e localizações-chave
- Informações de autenticação Outros processos do SDK, como o proxy MQTT, fazem referência a esse arquivo para determinar o estado do hub e as configurações de conexão.

Interface do manipulador de certificados

Fornece uma interface utilitária para leitura e gravação de certificados e chaves de dispositivos. Você pode implementar essa interface para trabalhar com:

- Armazenamento do sistema de arquivos
- Módulos de segurança de hardware (HSM)
- Módulos de plataforma confiáveis (TPM)
- Soluções personalizadas de armazenamento seguro

Componente proxy MQTT

Gerencia device-to-cloud a comunicação usando:

- Certificados e chaves de cliente provisionados
- Informações sobre o estado do dispositivo a partir do arquivo de configuração
- Conexões MQTT para integrações gerenciadas

O diagrama a seguir descreve a arquitetura do subsistema de integração do hub e seus componentes. Se você não estiver usando AWS IoT Greengrass, você pode ignorar esse componente do diagrama.

Configuração de integração do hub

Conclua essas etapas de configuração para cada dispositivo de hub antes de iniciar o processo de integração do provisionamento da frota. Esta seção descreve como criar itens gerenciados, configurar estruturas de diretórios e configurar os certificados necessários.

Etapas de configuração

Etapa 1: registrar um endpoint personalizado

Crie um terminal de comunicação dedicado que seus dispositivos usem para trocar dados com integrações gerenciadas. Esse endpoint estabelece um ponto de conexão seguro para todas as device-to-cloud mensagens, incluindo comandos do dispositivo, atualizações de status e notificações.

Para registrar um endpoint

- Use a RegisterCustomEndpointAPI para criar um endpoint para comunicação de device-to-managed integrações.

  RegisterCustomEndpointExemplo de solicitação

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

  Resposta:

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

  > **Note**
  >
  > Armazene o endereço do endpoint. Você precisará dele para a futura comunicação com dispositivos.

Para retornar as informações do endpoint, use a `GetCustomEndpoint` API.

Para obter mais informações, consulte a RegisterCustomEndpointAPI e a GetCustomEndpointAPI no Guia de referência da API de integrações gerenciadas.

Etapa 2: criar um perfil de aprovisionamento

Um perfil de provisionamento contém as credenciais de segurança e as configurações de que seus dispositivos precisam para se conectar às integrações gerenciadas.

Para criar um perfil de aprovisionamento de frota

- Chame a CreateProvisioningProfileAPI para gerar o seguinte:
  - Um modelo de provisionamento que define as configurações de conexão do dispositivo
  - Um certificado de solicitação e uma chave privada para autenticação de dispositivos

> **Important**
>
> Armazene o certificado de solicitação, a chave privada e o ID do modelo com segurança. Você precisará dessas credenciais para integrar dispositivos a integrações gerenciadas. Se você perder essas credenciais, deverá criar um novo perfil de aprovisionamento.

`CreateProvisioningProfile`exemplo de solicitação

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

Resposta:

```
{
"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"
}
```

Etapa 3: criar uma coisa gerenciada (provisionamento de frota)

Use a `CreateManagedThing` API para criar algo gerenciado para seu dispositivo hub. Cada hub exige sua própria coisa gerenciada com materiais de autenticação exclusivos. Para obter mais informações, consulte a CreateManagedThingAPI na Referência da API de integrações gerenciadas.

Ao criar uma coisa gerenciada, especifique estes parâmetros:

- `Role`: defina esse valor `CONTROLLER` para hubs que não oferecem suporte a comando e controle, caso contrário, defina `DEVICE` como.
- `AuthenticationMaterialType`: defina esse valor como`WIFI_SETUP_QR_BAR_CODE`.
- `AuthenticationMaterial`: inclua os seguintes campos. Você pode usar um `UPC` ou`EAN`, mas não os dois.
  - SN: O número de série exclusivo deste dispositivo
  - UPC: O código de produto universal para este dispositivo
  - EAN: O número internacional do artigo deste dispositivo

> **Important**
>
> Cada dispositivo deve ter um número de série (SN) exclusivo em seu material de autenticação.

`CreateManagedThing`Exemplo de solicitação:

```
{
 "Role": "CONTROLLER",
 "AuthenticationMaterialType": "WIFI_SETUP_QR_BAR_CODE",
```

```
 "AuthenticationMaterial": "SN:123456789524;UPC:829576019524"
}
```

Para obter mais informações, consulte CreateManagedThinga Referência da API de integrações gerenciadas.

(Opcional) Get Managed Thing

O que `ProvisioningStatus` você gerencia deve ser `PRE_ASSOCIATED` antes que você possa continuar. Para obter mais informações sobre ProvisioningStatus, consulte Provisionamento de dispositivos. Use a `GetManagedThing` API para verificar se sua coisa gerenciada existe e está pronta para provisionamento. Para obter mais informações, consulte GetManagedThinga Referência da API de integrações gerenciadas.

Etapa 4: criar a estrutura de diretórios

Crie diretórios para seus arquivos de configuração e certificados. Por padrão, o processo de integração do hub usa o. `/data/aws/iotmi/config/iotmi_config.json`

Você pode especificar caminhos personalizados para certificados e chaves privadas no arquivo de configuração. Este guia usa o caminho padrão`/data/aws/iotmi/certs`.

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

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

Etapa 5: Adicionar materiais de autenticação ao dispositivo hub

Copie certificados e chaves para seu dispositivo hub e, em seguida, crie um arquivo de configuração específico do dispositivo. Esses arquivos estabelecem uma comunicação segura entre seu hub e as integrações gerenciadas durante o processo de provisionamento.

Para copiar o certificado de reclamação e a chave

- Copie esses arquivos de autenticação da sua resposta de `CreateProvisioningProfile` API para o seu dispositivo hub:

- `claim_cert.pem`: O certificado de solicitação (comum a todos os dispositivos)
- `claim_pk.key`: a chave privada para o certificado de solicitação

Coloque os dois arquivos no `/data/aws/iotmi/certs` diretório.

> **Important**
>
> Ao armazenar certificados e chaves privadas no formato PEM, garanta a formatação adequada manipulando os caracteres de nova linha corretamente. Para arquivos codificados em PEM, os caracteres de nova linha (`\n`) devem ser substituídos por separadores de linha reais, pois o simples armazenamento de novas linhas com escape não será recuperado corretamente posteriormente.

> **Note**
>
> Se você usa armazenamento seguro, armazene essas credenciais em seu local de armazenamento seguro em vez de no sistema de arquivos. Para obter mais informações, consulte Crie um manipulador de certificados personalizado para armazenamento seguro.

Etapa 6: criar o arquivo de configuração do dispositivo

Crie um arquivo de configuração que contenha identificadores exclusivos de dispositivos, locais de certificados e configurações de provisionamento. O SDK usa esse arquivo durante a integração do hub para autenticar seu dispositivo, gerenciar o status do provisionamento e armazenar as configurações de conexão.

> **Note**
>
> Cada dispositivo hub exige seu próprio arquivo de configuração com valores exclusivos específicos do dispositivo.

Use o procedimento a seguir para criar ou modificar seu arquivo de configuração e copiá-lo para o hub.

- Crie ou modifique o arquivo de configuração (provisionamento da frota).

  Configure esses campos obrigatórios no arquivo de configuração do dispositivo:

  - Caminhos de certificado
    1. `iot_claim_cert_path`: Localização do seu certificado de reclamação (`claim_cert.pem`)
    2. `iot_claim_pk_path`: Localização da sua chave privada (`claim_pk.key`)
    3. Use `SECURE_STORAGE` para ambos os campos ao implementar o Secure Storage Cert Handler
  - Configurações de conexão
    1. `fp_template_name`: O `ProvisioningProfile` nome anterior.
    2. `endpoint_url`: o URL do endpoint de integrações gerenciadas da resposta da `RegisterCustomEndpoint` API (o mesmo para todos os dispositivos em uma região).
  - Identificadores de dispositivo
    1. SN: número de série do dispositivo que corresponde à sua chamada de `CreateManagedThing` API (exclusivo por dispositivo)
    2. UPCCódigo de produto universal da sua chamada de `CreateManagedThing` API (o mesmo para todos os dispositivos deste produto)

```
{
    "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"
    }
}
```

Conteúdo do arquivo de configuração

Revise o conteúdo do `iotmi_config.json` arquivo.

Conteúdo

Chave	Valores	Adicionado pelo cliente?	Observações
`iot_provisioning_method`	`FLEET_PROVISIONING`	Sim	Especifique o método de aprovisionamento que você deseja usar.
`iot_claim_cert_path`	O caminho do arquivo que você especifica ou`SECURE_STORAGE` . Por exemplo, `/data/aws/iotmi/certs/claim_cert.pem` .	Sim	Especifique o caminho do arquivo que você deseja usar ou`SECURE_STORAGE` .
`iot_claim_pk_path`	O caminho do arquivo que você especifica ou`SECURE_STORAGE` . Por exemplo, `/data/aws/iotmi/certs/claim_pk.pem` .	Sim	Especifique o caminho do arquivo que você deseja usar ou`SECURE_STORAGE` .
`fp_template_name`	O nome do modelo de aprovisionamento da frota deve ser igual ao nome do `ProvisioningProfile` que foi usado anteriormente.	Sim	Igual ao nome do `ProvisioningProfile` que foi usado anteriormente
`endpoint_url`	O URL do endpoint para integrações gerenciadas.	Sim	Seus dispositivos usam esse URL para se conectar à nuvem de integrações gerenciadas. Para obter essas informaçõ

Chave	Valores	Adicionado pelo cliente?	Observações
			es, use a RegisterCustomEndpointAPI.
SN	O número de série do dispositivo. Por exemplo, .AIDACKCEVSQ6C2EXAMPLE	Sim	Você deve fornecer essas informações exclusivas para cada dispositivo.
UPC	Código de produto universal do dispositivo. Por exemplo, .841667145075	Sim	Você deve fornecer essas informações para o dispositivo.
managed_thing_id	O ID da coisa gerenciada.	Não	Essas informações são adicionadas posteriormente pelo processo de integração após o provisionamento do hub.
iot_provisioning_state	O estado de aprovisionamento.	Sim	O estado de provisionamento deve ser definido como. NOT_PROVISIONED
iot_permanent_cert_path	O caminho do certificado de IoT. Por exemplo, ./data/aws/iotmi/iot_cert.pem	Não	Essas informações são adicionadas posteriormente pelo processo de integração após o provisionamento do hub.
iot_permanent_pk_path	O caminho do arquivo da chave privada da IoT. Por exemplo, ./data/aws/iotmi/iot_pk.pem	Não	Essas informações são adicionadas posteriormente pelo processo de integração após o provisionamento do hub.

Chave	Valores	Adicionado pelo cliente?	Observações
`client_id`	O ID do cliente que será usado para conexões MQTT.	Não	Essas informações são adicionadas posteriormente pelo processo de integração após o provisionamento do hub, para que outros componentes sejam consumidas.
`mqtt_keep_alive_interval`	O alcance é de 30 a 1200 e as unidades estão em segundos. O valor padrão é 300.	Sim	Use isso para definir um intervalo de manutenção de atividade para conexões MQTT.
`event_manager_upper_bound`	O valor padrão é 500.	Não	Essas informações são adicionadas posteriormente pelo processo de integração após o provisionamento do hub, para que outros componentes sejam consumidas.

Etapa 7: Copie o arquivo de configuração para o seu hub

Copie seu arquivo de configuração `/data/aws/iotmi/config` ou seu caminho de diretório personalizado. Você fornecerá esse caminho para o `HubOnboarding` binário durante o processo de integração.

Para provisionamento de frotas

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

```
                claim_pk.key
```

Integre dispositivos e opere-os no hub

Configure seus dispositivos para serem integrados ao seu hub de integrações gerenciadas criando uma coisa gerenciada e conectando-a ao seu hub. Os dispositivos podem ser integrados a um hub por meio de configuração simples ou configuração guiada pelo usuário.

Tópicos

- Configuração simples para integrar e operar dispositivos
- Configuração guiada pelo usuário para integrar e operar dispositivos

Configuração simples para integrar e operar dispositivos

Configure seus dispositivos para serem integrados ao seu hub de integrações gerenciadas criando uma coisa gerenciada e conectando-a ao seu hub. Esta seção descreve as etapas para concluir o processo de integração do dispositivo usando uma configuração simples.

Pré-requisitos

Conclua estas etapas antes de tentar integrar um dispositivo:

- Integre um dispositivo de hub ao hub de integrações gerenciadas.
- Instale a versão mais recente do a AWS CLI partir da Referência de AWS CLI Comandos de Integrações Gerenciadas
- Inscreva-se para receber notificações de eventos do DEVICE_LIFE_CYCLE.

Etapas de configuração

- Etapa 1: criar um armário de credenciais
- Etapa 2: adicione o armário de credenciais ao seu hub
- Etapa 3: Crie uma coisa gerenciada com credenciais.
- Etapa 4: conecte o dispositivo e verifique seu status.
- Etapa 5: Obtenha os recursos do dispositivo
- Etapa 6: enviar um comando para a coisa gerenciada

Etapa 1: criar um armário de credenciais

Crie um armário de credenciais para seu dispositivo.

Para criar um armário de credenciais

- Use o comando `create-credential-locker`. A execução desse comando acionará a criação de todos os recursos de fabricação, incluindo o key pair de configuração Wi-Fi e o certificado do dispositivo.

  create-credential-locker exemplo

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

  Resposta:

  ```
  {
    "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"
  }
  ```

  Para obter mais informações, consulte o create-credential-lockercomando na Referência de AWS CLI comandos de integrações gerenciadas.

Etapa 2: adicione o armário de credenciais ao seu hub

Adicione o armário de credenciais ao seu hub.

Para adicionar um armário de credenciais ao seu hub

- Use o comando a seguir para adicionar um armário de credenciais ao seu hub.

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

Etapa 3: Crie uma coisa gerenciada com credenciais.

Crie uma coisa gerenciada com credenciais para seu dispositivo. Cada dispositivo requer sua própria coisa gerenciada.

Para criar uma coisa gerenciada

- Use o `create-managed-thing` comando para criar uma coisa gerenciada para o seu dispositivo.

  create-managed-thing exemplo

```
#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**
  >
  > Existem comandos separados para dispositivos Z-wave e Zigbee.

  Resposta:

```
{
  "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"
}
```

  Para obter mais informações, consulte o create-managed-thingcomando na Referência de AWS CLI comandos de integrações gerenciadas.

Etapa 4: conecte o dispositivo e verifique seu status.

Conecte o dispositivo e verifique seu status.

- Use o `get-managed-thing` comando para verificar o status do seu dispositivo. O que ProvisioningStatus você gerencia deve estar ATIVADO. Para obter mais informações sobre ProvisioningStatus, consulte Provisionamento de dispositivos.

  get-managed-thing exemplo

```
#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"
```

  Resposta:

```
{
  "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"
}
```

Para obter mais informações, consulte o get-managed-thingcomando na Referência de AWS CLI comandos de integrações gerenciadas.

Etapa 5: Obtenha os recursos do dispositivo

Use o `get-managed-thing-capabilities` comando para obter seu ID de endpoint e ver uma lista de ações possíveis para seu dispositivo.

Para obter os recursos de um dispositivo

- Use o `get-managed-thing-capabilities` comando e anote o ID do endpoint.

  get-managed-thing-capabilties exemplo

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

  Resposta:

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

Para obter mais informações, consulte o get-managed-thing-capabilitiescomando na Referência de AWS CLI comandos de integrações gerenciadas.

Etapa 6: enviar um comando para a coisa gerenciada

Use o `send-managed-thing-command` comando para enviar um comando de ação de alternância para sua coisa gerenciada.

Para enviar um comando para sua coisa gerenciada

- Use o `send-managed-thing-command` comando para enviar um comando para sua coisa gerenciada.

  send-managed-thing-command exemplo

```
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**
>
> Este exemplo usa jq cli para, mas você também pode passar a string inteira `endpointId`

Resposta:

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

Para obter mais informações, consulte o send-managed-thing-commandcomando na Referência de AWS CLI comandos de integrações gerenciadas.

Etapa 7: remover a coisa gerenciada do seu hub

Limpe seu hub removendo a coisa gerenciada.

Para excluir uma coisa gerenciada

- Use o `delete-managed-thing` comando para remover uma coisa gerenciada do hub do seu dispositivo.

  delete-managed-thing exemplo

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

  Para obter mais informações, consulte o delete-managed-thingcomando na Referência de AWS CLI comandos de integrações gerenciadas.

> **Note**
>
> Se o dispositivo estiver preso em um `DELETE_IN_PROGRESS` estado, anexe a `--force` bandeira ao. `delete-managed-thing command`

> **Note**
>
> Para dispositivos Z-wave, você precisa colocar o dispositivo no modo de emparelhamento após executar o comando.

Configuração guiada pelo usuário para integrar e operar dispositivos

Configure seus dispositivos para serem integrados ao seu hub de integrações gerenciadas criando uma coisa gerenciada e conectando-a ao seu hub. Esta seção descreve as etapas para concluir o processo de integração do dispositivo usando a configuração guiada pelo usuário.

Pré-requisitos

Conclua estas etapas antes de tentar integrar um dispositivo:

- Integre um dispositivo de hub ao hub de integrações gerenciadas.
- Instale a versão mais recente do a AWS CLI partir da Referência de AWS CLI Comandos de Integrações Gerenciadas
- Inscreva-se para receber notificações de eventos DEVICE_DISCOVERY-STATUS.

Etapas de configuração guiadas pelo usuário

- Pré-requisito: ativar o modo de emparelhamento em seu dispositivo Z Wave
- Etapa 1: iniciar a descoberta do dispositivo
- Etapa 2: consultar o ID do trabalho de descoberta
- Etapa 3: criar uma coisa gerenciada para seu dispositivo
- Etapa 4: consultar a coisa gerenciada
- Etapa 5: Obtenha recursos de gerenciamento de coisas
- Etapa 6: enviar um comando para a coisa gerenciada

- Etapa 7: verificar o estado da coisa gerenciada
- Etapa 8: remover o item gerenciado do seu hub

Pré-requisito: ativar o modo de emparelhamento em seu dispositivo Z Wave

Ative o modo de emparelhamento no dispositivo Z-wave. O modo de emparelhamento pode variar para cada dispositivo Z-Wave, portanto, consulte as instruções do dispositivo para configurar corretamente o modo de emparelhamento. Geralmente é um botão que o usuário deve pressionar.

Etapa 1: iniciar a descoberta do dispositivo

Inicie a descoberta de dispositivos em seu hub para obter um ID de trabalho de descoberta que é usado para integrar seu dispositivo.

Para iniciar a descoberta de dispositivos

- Use o start-device-discoverycomando para obter o ID do trabalho de descoberta.

  start-device-discovery exemplo

```
#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 \
```

  Resposta:

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

> **Note**
>
> Existem comandos separados para dispositivos Z-wave e Zigbee.

Para obter mais informações, consulte a start-device-discoveryAPI na Referência de AWS CLI comandos de integrações gerenciadas.

Etapa 2: consultar o ID do trabalho de descoberta

Use o `list-discovered-devices` comando para obter o material de autenticação do seu dispositivo.

Para consultar seu ID de trabalho de descoberta

- Use o ID do trabalho de descoberta com o list-discovered-devices comando para obter o material de autenticação do seu dispositivo.

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

Resposta:

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

Etapa 3: criar uma coisa gerenciada para seu dispositivo

Use o `create-managed-thing` comando para criar uma coisa gerenciada para o seu dispositivo. Cada dispositivo requer sua própria coisa gerenciada.

Para criar uma coisa gerenciada

- Use o `create-managed-thing` comando para criar uma coisa gerenciada para o seu dispositivo.

  create-managed-thing exemplo

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

  Resposta:

  ```
  {
      "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"
    }
  ```

  Para obter mais informações, consulte o create-managed-thingcomando na Referência de AWS CLI comandos de integrações gerenciadas.

Etapa 4: consultar a coisa gerenciada

Você pode verificar se uma coisa gerenciada está ativada usando o `get-managed-thing` comando.

Para consultar uma coisa gerenciada

- Use o `get-managed-thing` comando para verificar se o status de provisionamento do item gerenciado está definido como. `ACTIVATED` Para obter mais informações sobre o status do provisionamento, consulte Provisionamento de dispositivos.

  get-managed-thing exemplo

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

Resposta:

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

Para obter mais informações, consulte o get-managed-thingcomando na Referência de AWS CLI comandos de integrações gerenciadas.

Etapa 5: Obtenha recursos de gerenciamento de coisas

Você pode ver uma lista das ações disponíveis de uma coisa gerenciada usando `get-managed-thing-capabiltiies` o.

Para obter os recursos de um dispositivo

- Use o `get-managed-thing-capabilities` comando para obter o ID do endpoint. Observe também a lista de ações possíveis.

  get-managed-thing-capabilities exemplo

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

  Resposta:

```
{
```

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

Para obter mais informações, consulte o get-managed-thing-capabilitiescomando na Referência de AWS CLI comandos de integrações gerenciadas.

Etapa 6: enviar um comando para a coisa gerenciada

Você pode usar o send-managed-thing-command comando para enviar um comando de ação de alternância para sua coisa gerenciada.

Envie um comando para a coisa gerenciada usando uma ação de alternância.

- Use o `send-managed-thing-command` comando para enviar um comando de ação de alternância.

  send-managed-thing-command exemplo

```
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**
>
> Este exemplo usa jq cli para, mas você também pode passar a string inteira `endpointId`

  Resposta:

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

```
  }
```

Para obter mais informações, consulte o send-managed-thing-commandcomando na Referência de AWS CLI comandos de integrações gerenciadas.

Etapa 7: verificar o estado da coisa gerenciada

Verifique o estado da coisa gerenciada para validar se a ação de alternância foi bem-sucedida.

Para verificar o estado do dispositivo de uma coisa gerenciada

- Use o `get-managed-thing-state` comando para validar se a ação de alternância foi bem-sucedida.

  get-managed-thing-state exemplo

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

  Resposta:

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

```
            ]
        }
    ]
}
```

Para obter mais informações, consulte o get-managed-thing-statecomando na Referência de AWS CLI comandos de integrações gerenciadas.

Etapa 8: remover o item gerenciado do seu hub

Limpe seu hub removendo a coisa gerenciada.

Para excluir uma coisa gerenciada

- Use o delete-managed-thingcomando para remover uma coisa gerenciada.

  delete-managed-thing exemplo

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

  Para obter mais informações, consulte o delete-managed-thingcomando na Referência de AWS CLI comandos de integrações gerenciadas.

  > **Note**
  >
  > Se o dispositivo estiver preso em um `DELETE_IN_PROGRESS` estado, anexe o `--force` sinalizador ao `delete-managed-thing` comando.

  > **Note**
  >
  > Para dispositivos Z-wave, você precisa colocar o dispositivo no modo de emparelhamento após executar o comando.

Crie um manipulador de certificados personalizado para armazenamento seguro

O gerenciamento de certificados de dispositivos é crucial ao integrar o hub de integrações gerenciadas. Embora os certificados sejam armazenados no sistema de arquivos por padrão, você pode criar um manipulador de certificados personalizado para maior segurança e gerenciamento flexível de credenciais.

O SDK de dispositivo final de integrações gerenciadas fornece um manipulador de certificados para proteger a interface de armazenamento que você pode implementar como uma biblioteca de objetos compartilhados (.so). Crie sua implementação de armazenamento seguro para ler e gravar certificados e, em seguida, vincule o arquivo da biblioteca ao HubOnboarding processo em tempo de execução.

Definição e componentes da API

Analise o `secure_storage_cert_handler_interface.hpp` arquivo a seguir para entender os componentes e os requisitos da API para sua implementação

Tópicos

Definição de API

Conteúdo de **`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
```

Componentes principais

- CERT_TYPE_T - diferentes tipos de certificados no hub.
  - RECLAMAÇÃO - o certificado de reclamação, originalmente no hub, será trocado por um certificado permanente.

  - DHA - não usado por enquanto.
  - PERMANENTE - certificado permanente para conexão com o endpoint de integrações gerenciadas.
- read_cert_and_private_key - (FUNÇÃO A SER IMPLEMENTADA) Lê o certificado e o valor da chave na entrada de referência. Essa função deve ser capaz de ler tanto o certificado CLAIM quanto o PERMANENT e é diferenciada pelo tipo de certificado mencionado acima.
- write_permanent_cert_and_private_key - (FUNÇÃO A SER IMPLEMENTADA) grava o certificado permanente e o valor da chave no local desejado.

Exemplo de construção

Separe seus cabeçalhos de implementação internos da interface pública (`secure_storage_cert_handler_interface.hpp`) para manter uma estrutura de projeto limpa. Com essa separação, você pode gerenciar componentes públicos e privados enquanto cria seu manipulador de certificados.

> **Note**
>
> Declare `secure_storage_cert_handler_interface.hpp` como público.

Tópicos

Estrutura do projeto

Herde a interface

Crie uma classe concreta que herde a interface. Oculte esse arquivo de cabeçalho e outros arquivos em um diretório separado para que os cabeçalhos privados e públicos possam ser diferenciados facilmente durante a criação.

```
#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
```

Implementação

Implemente a classe de armazenamento definida acima,`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;
 }
```

Implemente a função de fábrica definida na interface,`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
    )
```

Uso

Após a compilação, você terá um arquivo de biblioteca de objetos `libSecureStorageCertHandler.so` compartilhados e seus links simbólicos associados. Copie o arquivo da biblioteca e os links simbólicos para a localização da biblioteca esperada pelo HubOnboarding binário.

Tópicos

- Considerações importantes
- Use armazenamento seguro

Considerações importantes

- Verifique se sua conta de usuário tem permissões de leitura e gravação para o HubOnboarding binário e a `libSecureStorageCertHandler.so` biblioteca.
- Mantenha `secure_storage_cert_handler_interface.hpp` como seu único arquivo de cabeçalho público. Todos os outros arquivos de cabeçalho devem permanecer em sua implementação privada.
- Verifique o nome da sua biblioteca de objetos compartilhados. Enquanto você cria`libSecureStorageCertHandler.so`, HubOnboarding pode exigir uma versão específica no nome do arquivo, como. `libSecureStorageCertHandler.so.1.0` Use o `ldd` comando para verificar as dependências da biblioteca e criar links simbólicos conforme necessário.
- Se sua implementação da biblioteca compartilhada tiver dependências externas, armazene-as em um diretório que HubOnboarding possa ser acessado, como `/usr/lib or the iotmi_common` diretório.

Use armazenamento seguro

Atualize seu `iotmi_config.json` arquivo configurando `iot_claim_cert_path` e `iot_claim_pk_path` para**`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"
  }
}
```

Plugin de protocolo personalizado

Você pode usar um plug-in de protocolo personalizado para integrar seus protocolos de IoT proprietários às integrações AWS IoT Device Management gerenciadas do ecossistema. Por meio de interfaces SDK bem definidas, você pode integrar dispositivos, definir recursos e lidar com fluxos de controle em tempo real, mantendo total compatibilidade com integrações gerenciadas e componentes do SDK do hub.

A lista a seguir discute os principais recursos do plug-in de protocolo personalizado.

Personalização do modelo de dados

Defina seus próprios esquemas AWS de modelo de dados e carregue-os em integrações gerenciadas durante o fluxo de provisionamento. Você pode usar esses esquemas posteriormente em seus fluxos de trabalho.

Implementação flexível de plugins

- Crie seu próprio componente de plug-in usando Cliente Hub SDK o.
- Implemente plug-ins separados para funcionalidades diferentes, como provisionamento e controle, ou crie um cliente unificado para ambas.

- Mantenha um limite claro entre os ativos de integrações gerenciadas e seus próprios ativos, como pilhas de middleware, para uma implementação lógica de código desacoplada e fácil de desenvolver.

Compatibilidade com versões anteriores

Para clientes existentes, integre facilmente seu novo protocolo personalizado e, ao mesmo tempo, mantenha os tipos de rádio existentes funcionando como estão.

O diagrama a seguir ilustra a arquitetura do plug-in de protocolo personalizado.

Cliente Hub SDK

A biblioteca cliente do Hub SDK serve como uma interface entre as integrações gerenciadas do Hub SDK e sua própria pilha de protocolos em execução no mesmo hub. Ele expõe um conjunto de públicos APIs para facilitar a interação da sua pilha de protocolos com os componentes do SDK do Device Hub. Os casos de uso incluem controle de plug-in personalizado, provisionador de plug-in personalizado e controlador local.

Tópicos

Obtenha suas integrações gerenciadas Hub SDK

O Hub SDK Client vem com o SDK de integrações gerenciadas. Entre em contato conosco pelo console de integrações gerenciadas para acessar o SDK do hub.

Sobre o kit de ferramentas do Hub SDK

Após o download, você verá uma `IotMI-DeviceSDK-Toolkit` pasta que contém todos os arquivos de cabeçalho públicos e `.so` arquivos que você pode consumir em seu aplicativo. A equipe

de integrações gerenciadas também fornece um exemplo `main.cpp` para fins de demonstração, junto com o binário do aplicativo de demonstração, no `bin/` qual você pode executar diretamente. Opcionalmente, você pode usar isso como ponto de partida para seu aplicativo.

Crie seu aplicativo personalizado com o cliente Hub SDK

Use as etapas a seguir para criar seu aplicativo personalizado.

1. Inclua os arquivos de cabeçalho (.h) e os arquivos de objetos compartilhados (.so) em seu aplicativo.

   Você deve incluir os arquivos de cabeçalho públicos (.h) e os arquivos de objetos compartilhados (.so) em seu aplicativo. Para os arquivos.so, você pode colocá-los em uma pasta lib. O layout final será semelhante a este:

   ```
   ### 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. Crie um cliente Hub SDK em seu aplicativo principal.

   a. Em seu aplicativo principal, você deve primeiro inicializar o cliente Hub SDK antes que ele possa ser usado para processar solicitações. Você pode simplesmente construir o cliente com um`clientId`.

   b. Depois de ter o cliente, você pode conectá-lo ao Device SDK de integrações gerenciadas.

      Veja a seguir um exemplo de como criar o cliente Hub SDK e como se conectar.

      ```
      #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* deve ser a mesma especificada start-device-discovery na configuração guiada pelo usuário ou create-managed-thing no fluxo de provisionamento de configuração simples.

3. Publique e assine seguindo as etapas a seguir.

   a. Depois que a conexão for estabelecida, agora você poderá assinar as tarefas recebidas do SDK do Hub de integrações gerenciadas. As tarefas recebidas podem ser tarefas de controle ou tarefas de provisionamento. Você também precisa definir sua própria função de retorno de chamada nas tarefas recebidas e seu próprio contexto personalizado para seus próprios propósitos de rastreamento.

```
    // 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. Depois que a conexão for estabelecida, agora você poderá publicar solicitações do seu aplicativo no SDK do Hub de integrações gerenciadas. Você pode definir seu próprio tipo de mensagem de tarefa, com cargas diferentes para diferentes fins comerciais. As solicitações podem incluir solicitações de controle e solicitações de provisão, de forma semelhante ao fluxo de assinatura. Por fim, você pode atribuir um endereço para obter `rspPayload` a resposta do SDK do Hub de integrações gerenciadas de forma sincronizada.

```
// 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. Crie o seu próprio `CMakeLists.txt` e crie seu aplicativo a partir daí. A saída final pode ser um binário executável, como `MyFirstApplication`

Executando seu aplicativo personalizado

Antes de executar seu aplicativo personalizado, conclua as etapas a seguir para configurar seu hub e iniciar o SDK do Hub de integrações gerenciadas:

- Siga as instruções de integração em. Integre seus hubs para integrações gerenciadas
- Conclua o processo de instalação documentado emInstale e valide as integrações gerenciadas Hub SDK.

Depois que os pré-requisitos forem atendidos, você poderá executar seu aplicativo personalizado. Por exemplo:

```
./MyFirstApplication
```

Important

Você deve atualizar manualmente o script de início listado Implemente o Hub SDK com um script com um script para iniciar seu próprio aplicativo. O pedido é importante, não altere o pedido.
Atualize o seguinte. Alteração

```
./IotMI-DeviceSDK-Toolkit/bin/DeviceSDKClientDemo >> $LOGS_DIR/
logDeviceSDKClientDemo_logs.txt &
```

com

```
./MyFirstApplication >> $LOGS_DIR/MyFirstApplication_logs.txtt &
```

Referência da API do cliente Hub SDK

O cliente Hub SDK (SDKClient classe Device) fornece uma interface para seu aplicativo personalizado interagir com as integrações gerenciadas Device SDK. Com esse cliente, você pode fazer o seguinte:

- Assine as tarefas relacionadas ao provisionamento e ao controle dos componentes de integrações gerenciadas.
- Publique solicitações relacionadas ao provisionamento e ao controle nos componentes de integrações gerenciadas.

Para obter informações sobre integrações gerenciadas para AWS IoT Device Management APIs, consulte O que é AWS Lambda.

Tópicos

Inicialização do cliente

Para começar a usar o`DeviceSDKClient`, inicialize-o com um ID de cliente.

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

Isso cria uma nova `DeviceSDKClient` instância com o especificado`clientId`. `clientId`Deve corresponder ao que você registrou com integrações gerenciadas.

Parâmetros

`clientId`(string) - O ID do cliente dessa instância.

```
connect()
```

Conecta a `DeviceSDKClient` instância às integrações gerenciadas.

Devoluções

- `IOTMI_STATUS_OK`- A conexão foi bem-sucedida.
- `IOTMI_STATUS_CUSTOM_PLUGIN_CONNECTION_ERROR`- Ocorreu um erro ao se conectar às integrações gerenciadas.

Provisionar assinatura de tarefas

Use esses métodos para assinar tarefas relacionadas ao provisionamento a partir dos componentes de integrações gerenciadas.

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

Assina tarefas relacionadas ao provisionamento, como integração e desprovisionamento de dispositivos, a partir dos componentes de integrações gerenciadas.

Parâmetros

- `callback`(Device SDKClient _SubscriberCallback) - Uma função de retorno de chamada que é executada quando uma tarefa é recebida.
- `context`(char*) - Um contexto personalizado passado para a função de retorno de chamada.

Devoluções

- `IOTMI_STATUS_OK`- A assinatura foi bem-sucedida.
- `IOTMI_STATUS_CUSTOM_PLUGIN_CLIENT_NOT_CONNECTED`- A SDKClient instância do dispositivo não está conectada às integrações gerenciadas.
- `IOTMI_STATUS_CUSTOM_PLUGIN_SUBSCRIBE_ERROR`- Ocorreu um erro ao assinar as tarefas.

Publicação de tarefas de provisionamento

Use esses métodos para publicar solicitações relacionadas ao provisionamento nos componentes de integrações gerenciadas.

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

Publica uma solicitação relacionada ao provisionamento nos componentes de integrações gerenciadas. Por exemplo, um evento de dispositivo integrado ou status de desprovisionamento

Parâmetros

`request`(DataModel: :iotmi_client_request_t) - Um ponteiro para uma estrutura de solicitação contendo os detalhes.

Devoluções

- `IOTMI_STATUS_OK`- A solicitação foi publicada com sucesso.
- `IOTMI_STATUS_CUSTOM_PLUGIN_CLIENT_NOT_CONNECTED`- A `DeviceSDKClient` instância não está conectada às integrações gerenciadas.
- `IOTMI_STATUS_INVALID_PARAMETER`- Um ou mais parâmetros na solicitação são inválidos.
- `IOTMI_STATUS_INVALID_JSON_OBJECT`- A carga útil da solicitação não é um objeto JSON válido.
- `IOTMI_STATUS_NO_MEMORY`- Ocorreu um erro de alocação de memória.

Controle a assinatura de tarefas

Use esses métodos para assinar tarefas relacionadas ao controle dos componentes de integrações gerenciadas.

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

Assina tarefas relacionadas ao controle (por exemplo, solicitações de controle de dispositivos) dos componentes de integrações gerenciadas.

Parâmetros

- `callback`(Device SDKClient _SubscriberCallback) - Uma função de retorno de chamada que é executada quando uma tarefa é recebida.
- `context`(char) - Um contexto personalizado passado para a função de retorno de chamada.

Devoluções

- `IOTMI_STATUS_OK`- A assinatura foi bem-sucedida.

- `IOTMI_STATUS_CUSTOM_PLUGIN_CLIENT_NOT_CONNECTED`- A `DeviceSDKClient` instância não está conectada às integrações gerenciadas.
- `IOTMI_STATUS_CUSTOM_PLUGIN_SUBSCRIBE_ERROR`- Ocorreu um erro ao assinar as tarefas.

Publicação de tarefas de controle

Use esses métodos para publicar solicitações relacionadas ao controle nos componentes de integrações gerenciadas.

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

Publica uma solicitação relacionada ao controle nos componentes de integrações gerenciadas. Por exemplo, eventos não solicitados, solicitações de comando ou consultas de estado do dispositivo.

Parâmetros

`request`(DataModel: :iotmi_client_request_t) - Um ponteiro para uma estrutura de solicitação contendo os detalhes.

Devoluções

- `IOTMI_STATUS_OK`- A solicitação foi publicada com sucesso.
- `IOTMI_STATUS_CUSTOM_PLUGIN_CLIENT_NOT_CONNECTED`- A SDKClient instância do dispositivo não está conectada às integrações gerenciadas.
- `IOTMI_STATUS_INVALID_PARAMETER`- Um ou mais parâmetros na solicitação são inválidos.
- `IOTMI_STATUS_INVALID_JSON_OBJECT`- A carga útil da solicitação não é um objeto JSON válido.
- `IOTMI_STATUS_NO_MEMORY`- Ocorreu um erro de alocação de memória.

Recursos de registro

Use esses métodos para implementar os recursos de registro fornecidos pelas integrações gerenciadas.

Inicialização do logger

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

Você deve inicializar o registrador antes de usar qualquer funcionalidade de registro.

Parâmetros

`logger_name`- O nome do registrador que você especifica. O valor padrão é: `MyApplication`

Registrando macros

`LOGGER_LOGD(...)`

Use essa macro em seu aplicativo para registro no nível DEBUG.

`LOGGER_LOGI(...)`

Use essa macro em seu aplicativo para registro em nível INFO.

`LOGGER_LOGW(...)`

Use essa macro em seu aplicativo para registro em nível WARN.

`LOGGER_LOGE(...)`

Use essa macro em seu aplicativo para registro em nível de ERRO.

> **Note**
>
> Para obter mais informações sobre os recursos de registro, consulte a documentação de registro do Hub. Os plug-ins de protocolo personalizados oferecem suporte total a todos os recursos de registro que as integrações gerenciadas oferecem.

Outros APIs

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

Retorna o ID do cliente associado à SDKClient instância do dispositivo.

Devoluções

O ID do cliente.

Tipos de dados

Esta seção define os tipos de dados usados para o plug-in de protocolo personalizado.

iotmi_client_request_t

Representa uma solicitação a ser publicada nos componentes de integrações gerenciadas.

MessageType

O tipo da mensagem (CommonTypes: :C2 MIMessage Type). A lista a seguir mostra os valores válidos.

- `C2MI_DEVICE_ONBOARDED`: indica uma mensagem de integração do dispositivo com carga útil relacionada.
- `C2MI_DE_PROVISIONING_PRE_ASSOCIATED_COMPLETE`: indica uma notificação de conclusão da tarefa de desprovisionamento para um dispositivo pré-associado.
- `C2MI_DE_PROVISIONING_ACTIVATED_COMPLETE`: indica uma notificação de conclusão da tarefa de desprovisionamento para um dispositivo ativado.
- `C2MI_DE_PROVISIONING_COMPLETE_RESPONSE`: indica uma resposta completa da tarefa de desprovisionamento.
- `C2MI_CONTROL_EVENT`: indica um evento de controle com possível alteração do estado do dispositivo.
- `C2MI_CONTROL_SEND_COMMAND`: indica um comando de controle de um controlador local.
- `C2MI_CONTROL_SEND_DEVICE_STATE_QUERY`: indica uma consulta de estado do dispositivo de controle de um controlador local.

Carga útil do REQ

A carga útil da solicitação, normalmente uma string formatada em JSON.

Carga útil do RSP

A carga útil de resposta, preenchida pelos componentes de integrações gerenciadas.

iotmi_client_event_t

Representa um evento recebido dos componentes de integrações gerenciadas.

ID do evento

O identificador exclusivo do evento.

length

A duração dos dados do evento.

data

Um ponteiro para os dados do evento, incluindo o. `messageType` A lista a seguir mostra os valores possíveis.

- `C2MI_PROVISION_UGS_TASK`: indica uma tarefa de provisão para o fluxo UGS.
- `C2MI_PROVISION_SS_TASK`: indica uma tarefa de provisionamento para o SimpleSetup fluxo.
- `C2MI_DE_PROVISION_PRE_ASSOCIATED_TASK`: indica uma tarefa de desprovisionamento para um dispositivo pré-associado.
- `C2MI_DE_PROVISION_ACTIVATED_TASK`: indica uma tarefa de desprovisionamento para um dispositivo ativado.
- `C2MI_DEVICE_ONBOARDED_RESPONSE`: indica uma resposta de integração do dispositivo.
- `C2MI_CONTROL_TASK`: indica uma tarefa de controle.
- `C2MI_CONTROL_EVENT_NOTIFICATION`: indica uma notificação de evento de controle para um controlador local.

ctx

Um contexto personalizado associado ao evento.

Controle do hub

O controle do Hub é uma extensão do SDK do dispositivo final de integrações gerenciadas que permite a interface com o `MQTTProxy` componente no SDK do Hub. Com o controle de hub, você pode implementar código usando o SDK do dispositivo final e controlar seu hub por meio da nuvem de integrações gerenciadas como um dispositivo separado. O SDK de controle do hub será fornecido como um pacote separado dentro do SDK do Hub, rotulado como. `iot-managed-integrations-hub-control-x.x.x`

Tópicos

Pré-requisitos

Para configurar o controle do hub, você precisa do seguinte:

- Um hub integrado ao Hub SDK, versão 0.4.0 ou superior.
- Baixe a versão mais recente do SDK do dispositivo final no Console de gerenciamento da AWS.
- Um componente proxy MQTT em execução no hub, versão 0.5.0 ou superior.

Componentes do SDK do dispositivo final

Use os seguintes componentes do SDK do dispositivo final:

- Gerador de código para o modelo de dados
- Manipulador de modelos de dados

Como o Hub SDK já tem um processo de integração e uma conexão com a nuvem, você não precisa dos seguintes componentes:

- Provisionado
- Interface PKCS
- Manipulador de trabalhos
- Agente MQTT

Integre com o SDK do dispositivo final

1. Siga as instruções no Gerador de código para modelo de dados para gerar o código C de baixo nível.
2. Siga as instruções em Integração do SDK do dispositivo final para:

a. Configurar o ambiente de construção

Crie o código no Amazon Linux 2023/x86_64 como seu host de desenvolvimento. Instale as dependências de compilação necessárias:

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

b. Desenvolva funções de retorno de chamada de hardware

Antes de implementar as funções de retorno de chamada do hardware, entenda como a API funciona. Este exemplo usa o On/Off cluster e o OnOff atributo para controlar a função de um dispositivo. Para obter detalhes da API, consulteFunção C de baixo nível 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. Configure endpoints e conecte funções de retorno de chamada de hardware

Depois de implementar as funções, crie endpoints e registre seus retornos de chamada. Conclua estas tarefas:

i. Crie um agente de dispositivo

ii. Preencha os pontos de função de retorno de chamada para cada estrutura de cluster que você deseja suportar

iii. Configure endpoints e registre clusters compatíveis

```
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);
}
```

Exemplo: controle de hub de construção

O controle do hub é fornecido como parte do pacote Hub SDK. O subpacote de controle do hub é rotulado `iot-managed-integrations-hub-control-x.x.x` e contém bibliotecas diferentes do SDK do dispositivo não modificado.

1. Mova os arquivos gerados pelo código para a `example` pasta:

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

2. Para criar o controle do hub, execute os seguintes comandos:

```
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. Execute os exemplos com o `MQTTProxy` componente no hub, com os `MQTTProxy` componentes `HubOnboarding` e em execução.

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

Consulte Modelo de dados de integrações gerenciadas o modelo de dados. Siga a Etapa 5 Comece a usar o SDK do dispositivo final para configurar endpoints e gerenciar as comunicações entre o usuário final e. iot-managed-integrations

Exemplos compatíveis

Os exemplos a seguir foram criados e testados:

- iotmi_device_dm_purificador_demo
- diagnóstico básico do dispositivo iotmi
- iotmi_device_dm_camera_demo

Plataformas compatíveis

A tabela a seguir mostra as plataformas suportadas para o controle do hub.

Arquitetura	Sistema operacional	Versão GCC	Versão Binutils
X86_64	Linux	10.5.0	2,37
aarch64	Linux	10.5.0	2,37

Ativar CloudWatch registros

O Hub SDK fornece uma funcionalidade abrangente de registro. Por padrão, o Hub SDK grava registros no sistema de arquivos local. No entanto, você pode aproveitar a API de nuvem para configurar o streaming de CloudWatch registros para o Logs, que oferece:

- Monitore o desempenho do dispositivo: capture registros detalhados de tempo de execução para gerenciamento proativo de dispositivos. Habilite a análise e o monitoramento avançados de registros em toda a sua frota de dispositivos
- Solucione problemas: gere entradas de registro granulares para uma análise rápida de diagnóstico. Registre eventos no nível do sistema e do aplicativo para uma investigação aprofundada.
- Registro flexível e centralizado: gerenciamento remoto de registros sem acesso direto ao dispositivo. Agregue registros de vários dispositivos em um único repositório pesquisável.

Pré-requisitos

- Integre o dispositivo gerenciado à nuvem. Para mais detalhes, consulte Configuração de integração do hub.
- Verifique a inicialização do agente Hub e a inicialização bem-sucedida. Para mais detalhes, consulte Instale e valide as integrações gerenciadas Hub SDK.

> **Note**
>
> Para criar configurações de registro, consulte a PutRuntimeLogConfiguration API para obter detalhes.

> **Warning**
>
> A ativação de registros conta para a medição de cotas em camadas. O aumento dos níveis de registro resultará em maior volume de mensagens e custos adicionais.

Configurar configurações de log do Hub SDK

Defina as configurações de log do SDK do hub chamando a API para definir a configuração do log de tempo de execução.

Example amostra de solicitação de API

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

RuntimeLogConfigurations atributos

Os atributos a seguir são opcionais e podem ser configurados na `RuntimeLogConfigurations` API.

LogLevel

Define o nível mínimo de severidade para rastreamentos de tempo de execução. Valores: `DEBUG, ERROR, INFO, WARN`

Padrão: `WARN` (versão lançada)

LogFlushLevel

Determina o nível de severidade da descarga imediata de dados para o armazenamento local. Valores: `DEBUG, ERROR, INFO, WARN`

Padrão: `DISABLED`

LocalStoreLocation

Especifica o local de armazenamento para rastreamentos de tempo de execução. Padrão: `/var/log/awsiotmi`

- Registro ativo: `/var/log/awsiotmi/ManagedIntegrationsDeviceSdkHub.log`

- Registros girados: `/var/log/awsiotmi/ManagedIntegrationsDeviceSdkHub.N.log` (N indica a ordem de rotação)

LocalStoreFileRotationMaxBytes

Aciona a rotação do arquivo quando o arquivo atual excede o tamanho especificado.

> **Important**
>
> Para uma eficiência ideal, mantenha o tamanho do arquivo abaixo de 125 KB. Valores acima de 125 KB serão automaticamente limitados.

LocalStoreFileRotationMaxFiles,

Define o número máximo de arquivos de rotação permitidos pelo daemon de log.

UploadLog

Controla a transferência de rastreamento em tempo de execução para a nuvem. Os registros são armazenados no grupo `/aws/iotmanagedintegration` CloudWatch Registros.

Padrão: `false`.

UploadPeriodMinutes

Define a frequência dos uploads de rastreamento em tempo de execução. Padrão: 5

DeleteLocalStoreAfterUpload

Controla a exclusão do arquivo após o upload. Padrão: `true`

> **Note**
>
> Se definido como false, os arquivos enviados serão renomeados para: `/var/log/awsiotmi/ManagedIntegrationsDeviceSdkHub.uploaded.{uploaded_timestamp}`

Exemplo de arquivo de log

Veja o exemplo de um arquivo de CloudWatch registros abaixo:

Tipos de dispositivos Zigbee e Z-Wave compatíveis

Esta página lista os tipos de dispositivos conectados ao hub que foram testados com integrações gerenciadas e são compatíveis. As integrações gerenciadas oferecem suporte a ambos Configuração simples (SS) e Configuração guiada pelo usuário (UGS) para esses dispositivos.

Esta tabela lista os dispositivos Zigbee compatíveis.

Tipo de dispositivo Zigbee	Recursos com suporte
Lâmpada inteligente/Luz regulável/ Luz RGB	OnOff, LevelControl, ColorControl
Plugue inteligente	OnOff
Interruptor inteligente	OnOff
Faixa de LED	OnOff, LevelControl, ColorControl
Válvula de água	OnOff
Válvula de radiador	Termostato, temporizador OnOff
Termostato	Termostato,, FanControl, Temporizador OnOff
Abridor de porta de garagem	WindowCovering, OnOff, LevelControl
Alarme de fumaça	BooleanState,, OnOff, Temporizador Temperatu reMeasurement, Fumaça COAlarm
Sensor de movimento	BooleanState
Sensor de ocupação/presença humana	BooleanState, OccupancySensing
Sensor de porta e janela	BooleanState
Sensor de vazamento de água	BooleanState

Tipo de dispositivo Zigbee	Recursos com suporte
Sensor de vibração	BooleanState
Sensor de temperatura e umidade	TemperatureMeasurement, RelativeHumidityMeasurement

Esta tabela lista os dispositivos Z-Wave compatíveis.

Tipo de dispositivo Z-Wave	Recursos com suporte
Lâmpada inteligente/Luz regulável	OnOff, LevelControl
Plugue inteligente	OnOff
Controlador de porta de garagem	OnOff, LevelControl
Medidor de energia	ElectricalEnergyMeasurement, ElectricalPowerMeasurement
Pilha	LevelControl
Sirene	LevelControl
Sensor de movimento	BooleanState
Sensor de porta e janela	BooleanState
Sensor de vazamento de água	BooleanState
Sensor de temperatura	TemperatureMeasurement
sensor de CO	Fumaça COAlarm
Sensor de fumaça	Fumaça COAlarm

Execute integrações gerenciadas no Raspberry Pi

> **Note**
>
> Essa implementação do AWS IoT Hub SDK no Raspberry Pi é um projeto de demonstração destinado apenas para fins de aprendizado e teste e não deve ser usada em ambientes de produção. Para fins desta demonstração, defina as seguintes configurações para facilitar o desenvolvimento:
> AWS armazenamento de credenciais: somente para fins de demonstração, as credenciais e os certificados são armazenados em um local acessível para facilitar o teste e o desenvolvimento. Os ambientes de produção devem usar soluções de armazenamento seguro AWS Secrets Manager, como o Systems Manager Parameter Store. Eles devem implementar a criptografia em repouso e seguir as diretrizes AWS IoT de segurança.
> Privilégios de contêiner: a demonstração é executada com privilégios elevados para permitir acesso irrestrito aos recursos do host e simplificar os fluxos de trabalho de desenvolvimento. Na produção, os contêineres devem operar com os privilégios mínimos necessários.
> Configuração da ponte de rede: a demonstração usa uma configuração de ponte de rede que expõe o tráfego interno da rede para facilitar a depuração e o monitoramento. Em ambientes de produção, implemente o isolamento e a segmentação adequados da rede para evitar o acesso não autorizado ao tráfego interno da rede.
> Permissões do dispositivo USB: o acesso irrestrito ao dispositivo USB é ativado para facilitar a conexão dos periféricos de desenvolvimento e dos dispositivos de teste. Para produção, implemente controles e validação rígidos de dispositivos USB para evitar ataques de falsificação de dispositivos.
> Essas configurações permitem testes diretos e não devem ser usadas em ambientes de produção. Ao implantar na produção, siga as melhores práticas de segurança para evitar o comprometimento do sistema host e o acesso não autorizado às credenciais.

Como pré-requisito, você deve configurar o dongle USB Sonoff Zigbee antes de configurar o Raspberry Pi.

Firmware flash para dongle USB Sonoff Zigbee

Pré-requisitos

- Dongle USB Sonoff Zigbee

- Windows: Instale o driver universal CP21 0x do Windows

Atualize o firmware

1. Baixe o Zigbee Dongle Firmware Build 7.4.1.0.
2. Abra o Silabs Firmware Flasher.
3. Conecte o Sonoff Zigbee USB Dongle ao seu computador.
4. Role e encontre ZBDongle-E.
5. Selecione Conectar.
6. Aguarde até que o dispositivo se conecte.
7. Escolha Alterar firmware.
8. Selecione Carregar seu próprio firmware.
9. Encontre a localização do download do Zigbee Dongle Firmware Build 7.4.1.0 e selecione-o.
10. Clique em Instalar.
11. Aguarde a instalação do firmware.
12. Escolha Continuar quando a instalação estiver concluída.

O dongle agora está pronto para uso.

Escolha entre as opções listadas abaixo para executar o SDK do Hub de integrações gerenciadas em seu Raspberry Pi. As etapas de configuração e validação de ambas as abordagens estão listadas abaixo.

Tópicos

- Imagem do SDK do Hub de integrações gerenciadas no Raspberry Pi
- Integrações gerenciadas: contêiner Hub SDK Docker no Raspberry Pi
- Aplicativo de demonstração de integrações gerenciadas

Imagem do SDK do Hub de integrações gerenciadas no Raspberry Pi

> **Note**
>
> Essa implementação do AWS IoT Hub SDK no Raspberry Pi é um projeto de demonstração destinado apenas para fins de aprendizado e teste e não deve ser usada em ambientes de produção. Para fins desta demonstração, defina as seguintes configurações para facilitar o desenvolvimento:
> AWS armazenamento de credenciais: somente para fins de demonstração, as credenciais e os certificados são armazenados em um local acessível para facilitar o teste e o desenvolvimento. Os ambientes de produção devem usar soluções de armazenamento seguro AWS Secrets Manager, como o Systems Manager Parameter Store. Eles devem implementar a criptografia em repouso e seguir as diretrizes AWS IoT de segurança.
> Privilégios de contêiner: a demonstração é executada com privilégios elevados para permitir acesso irrestrito aos recursos do host e simplificar os fluxos de trabalho de desenvolvimento. Na produção, os contêineres devem operar com os privilégios mínimos necessários.
> Configuração da ponte de rede: a demonstração usa uma configuração de ponte de rede que expõe o tráfego interno da rede para facilitar a depuração e o monitoramento. Em ambientes de produção, implemente o isolamento e a segmentação adequados da rede para evitar o acesso não autorizado ao tráfego interno da rede.
> Permissões do dispositivo USB: o acesso irrestrito ao dispositivo USB é ativado para facilitar a conexão dos periféricos de desenvolvimento e dos dispositivos de teste. Para produção, implemente controles e validação rígidos de dispositivos USB para evitar ataques de falsificação de dispositivos.
> Essas configurações permitem testes diretos e não devem ser usadas em ambientes de produção. Ao implantar na produção, siga as melhores práticas de segurança para evitar o comprometimento do sistema host e o acesso não autorizado às credenciais.

Pré-requisitos

Preencha estes requisitos antes de implantar a imagem do Raspberry Pi:

- Baixe e instale o gerador de imagens Raspberry Pi.
- Obtenha um cartão SD.
- Configure um Raspberry Pi 5 com CPU quad-core de 2,4 GHz e 64 bits (8 GB de RAM).
- Conecte um dongle USB Sonoff Zigbee.

- Firmware flash para dongle USB Sonoff Zigbee.
- Conecte um dongle SLUSB001A da Silicon Labs.
- Cadastre-se para criar uma AWS conta.
- Instale a versão mais recente do a AWS CLI partir da Referência de AWS CLI Comandos de Integrações Gerenciadas.

Flash uma imagem do Raspberry Pi em um novo cartão SD

Atualize a imagem das integrações gerenciadas em seu cartão SD usando estas etapas:

1. Baixe a imagem do SDK do Raspberry Pi Hub de integrações gerenciadas.
2. Inicie o Raspberry Pi Imager em seu desktop.
3. Insira o cartão SD no leitor de cartão SD embutido do seu computador ou no leitor de cartão USB externo.
4. Selecione Escolher dispositivo → Raspberry Pi 5.
5. Selecione Escolher sistema operacional → Usar personalizado → Encontre o arquivo IotMI-HubSDK-RPi-Image -v1.0.0.img.gz → Abrir.
6. Selecione Escolher armazenamento → Selecione seu leitor de cartão SD.
7. Verifique se sua configuração corresponde à seguinte tela:

8. Clique em Next.
9. Defina as configurações de personalização do sistema operacional:
   - Nome do host: Selecione raspberrypi.
   - Nome de usuário e senha:
     - Ativar Definir nome de usuário e senha:
     - Para Nome de usuário:, insira`hub123456`.
     - Para Senha:, insira`sh123456`.
   - LAN sem fio:
     - Ative Configurar LAN sem fio.
     - Digite o SSID e a senha do seu roteador.

     Exemplo de configurações:

     - SSID: `iotmi-tplink`

    - Senha: `********` (mínimo de 8 caracteres)
  - Defina o país: para `US`.
- Defina as configurações locais:
  - Defina o fuso horário: para `America/Los Angeles`.
  - Defina o layout do teclado: para `US`.
- SSH:
  - Escolha a guia serviços.
  - Marque Ativar SSH.
  - Escolha Usar autenticação por senha.

10. Confirme todos os pop-ups para personalização do sistema operacional e eliminação de dados.
11. Aguarde até que o processo de escrita seja concluído.
12. Verifique a conclusão bem-sucedida com a seguinte tela:

13. Clique em Continue.
14. Remova o cartão SD e insira-o no seu Raspberry Pi.

Execute o Hub SDK no Raspberry Pi

Inicie os serviços Hub SDK em seu Raspberry Pi configurado:

1. Insira o cartão SD preparado no dispositivo Raspberry Pi 5.
2. Conecte o dongle USB Sonoff Zigbee e o dongle SLUSB001A da Silicon Labs ao Raspberry Pi.
3. Ligue o Raspberry Pi.
4. Certifique-se de que o Raspberry Pi e seu computador (do qual você usa o SSH) estejam na mesma rede.
5. Faça SSH no Raspberry Pi usando as credenciais que você definiu durante a implantação da imagem.

```
ssh username@hostname
```

6. Navegue até o diretório do SDK do hub:

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

7. Conclua a configuração de integração do Hub para adicionar materiais de autenticação e configuração.

> **Note**
>
> Você deve estar em YUL nossa DUB região para realizar essa etapa.

8. Execute o Hub SDK:

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

O sistema exibe a seguinte resposta para uma inicialização bem-sucedida do Hub SDK:

```
-------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----
```

Próximas etapas

Depois de iniciar com sucesso o Hub SDK, continue com a integração e o gerenciamento do dispositivo em. [Configuração guiada pelo usuário para integrar e operar dispositivos]

Integrações gerenciadas: contêiner Hub SDK Docker no Raspberry Pi

> **Note**
>
> Essa implementação do AWS IoT Hub SDK no Raspberry Pi é um projeto de demonstração destinado apenas para fins de aprendizado e teste e não deve ser usada em ambientes de produção. Para fins desta demonstração, defina as seguintes configurações para facilitar o desenvolvimento:
> AWS armazenamento de credenciais: somente para fins de demonstração, as credenciais e os certificados são armazenados em um local acessível para facilitar o teste e o desenvolvimento. Os ambientes de produção devem usar soluções de armazenamento seguro AWS Secrets Manager, como o Systems Manager Parameter Store. Eles devem implementar a criptografia em repouso e seguir as diretrizes AWS IoT de segurança.
> Privilégios de contêiner: a demonstração é executada com privilégios elevados para permitir acesso irrestrito aos recursos do host e simplificar os fluxos de trabalho de desenvolvimento. Na produção, os contêineres devem operar com os privilégios mínimos necessários.
> Configuração da ponte de rede: a demonstração usa uma configuração de ponte de rede que expõe o tráfego interno da rede para facilitar a depuração e o monitoramento. Em ambientes

de produção, implemente o isolamento e a segmentação adequados da rede para evitar o acesso não autorizado ao tráfego interno da rede.
Permissões do dispositivo USB: o acesso irrestrito ao dispositivo USB é ativado para facilitar a conexão dos periféricos de desenvolvimento e dos dispositivos de teste. Para produção, implemente controles e validação rígidos de dispositivos USB para evitar ataques de falsificação de dispositivos.
Essas configurações permitem testes diretos e não devem ser usadas em ambientes de produção. Ao implantar na produção, siga as melhores práticas de segurança para evitar o comprometimento do sistema host e o acesso não autorizado às credenciais.

Pré-requisitos

Os pré-requisitos a seguir são necessários para o contêiner docker.

- Baixe e instale o gerador de imagens Raspberry Pi.
- Obtenha um cartão SD.
- Configure um Raspberry Pi 5 com CPU quad-core de 2,4 GHz e 64 bits (8 GB de RAM).
- Conecte um dongle USB Sonoff Zigbee.
- Firmware flash para dongle USB Sonoff Zigbee.
- Conecte um dongle SLUSB001A da Silicon Labs.
- Cadastre-se para criar uma AWS conta.
- Instale a versão mais recente do a AWS CLI partir da Referência de AWS CLI Comandos de Integrações Gerenciadas.
- Acesso SSH ao Raspberry Pi com endereço IP ou nome do host.

Use o contêiner Hub SDK Docker de integrações gerenciadas no Raspberry Pi

1. Baixe integrações gerenciadas Raspberry Pi Hub SDK Docker.
2. Copie o arquivo para o Raspberry Pi usando o SCP:

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

3. Conecte-se ao Raspberry Pi via SSH:

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

4. Instale o Docker se não estiver presente:

```
# 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. Instale o Docker Compose se não estiver presente:

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

6. Extraia os arquivos do Hub SDK:

```
# Navigate to the home directory
cd

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

7. Navegue até o diretório hub-docker:

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

8. Conclua a configuração de integração do Hub para definir a autenticação e as configurações.

> **Note**
>
> Você deve estar em YUL nossa DUB região para realizar essa etapa.

9. Inicie o contêiner Docker:

```
# The first time it's called, it will build the container
```

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

Saída esperada:

```
[+] 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-\-\-\-
```

Depois de iniciar com sucesso o Hub SDK, continue com a integração e o gerenciamento do dispositivo em. Configuração guiada pelo usuário para integrar e operar dispositivos

Note

- Para acessar o shell bash do contêiner Docker, execute o seguinte comando:

  ```
  docker compose exec hubsdk bash
  ```

- Para reiniciar o contêiner após a reinicialização, execute o seguinte comando:

  ```
  docker compose up -d
  ```

- Para atualizar o SDK do Hub, substitua os binários na seguinte pasta:

  ```
  hub-docker/iotmi
  ```

- Para reiniciar o contêiner com segurança e, ao mesmo tempo, preservar os dados, faça:

  ```
  docker compose down
  docker compose up -d
  ```

```
docker compose logs -f
```

Aplicativo de demonstração de integrações gerenciadas

> **Note**
>
> Essa implementação do AWS IoT Hub SDK no Raspberry Pi é um projeto de demonstração destinado apenas para fins de aprendizado e teste e não deve ser usada em ambientes de produção. Para fins desta demonstração, defina as seguintes configurações para facilitar o desenvolvimento:
> AWS armazenamento de credenciais: somente para fins de demonstração, as credenciais e os certificados são armazenados em um local acessível para facilitar o teste e o desenvolvimento. Os ambientes de produção devem usar soluções de armazenamento seguro AWS Secrets Manager, como o Systems Manager Parameter Store. Eles devem implementar a criptografia em repouso e seguir as diretrizes AWS IoT de segurança.
> Privilégios de contêiner: a demonstração é executada com privilégios elevados para permitir acesso irrestrito aos recursos do host e simplificar os fluxos de trabalho de desenvolvimento. Na produção, os contêineres devem operar com os privilégios mínimos necessários.
> Configuração da ponte de rede: a demonstração usa uma configuração de ponte de rede que expõe o tráfego interno da rede para facilitar a depuração e o monitoramento. Em ambientes de produção, implemente o isolamento e a segmentação adequados da rede para evitar o acesso não autorizado ao tráfego interno da rede.
> Permissões do dispositivo USB: o acesso irrestrito ao dispositivo USB é ativado para facilitar a conexão dos periféricos de desenvolvimento e dos dispositivos de teste. Para produção, implemente controles e validação rígidos de dispositivos USB para evitar ataques de falsificação de dispositivos.
> Essas configurações permitem testes diretos e não devem ser usadas em ambientes de produção. Ao implantar na produção, siga as melhores práticas de segurança para evitar o comprometimento do sistema host e o acesso não autorizado às credenciais.

O aplicativo de demonstração é um aplicativo de demonstração baseado em React que apresenta recursos de integrações gerenciadas para gerenciamento de dispositivos domésticos inteligentes. Este aplicativo demonstra a integração, controle e monitoramento de dispositivos Z-Wave e Zigbee por meio de uma interface web moderna.

Pré-requisitos

- Cadastre-se para criar uma AWS conta.
- Crie um armário de credenciais e adicione o armário de credenciais ao seu hub.
- Configuração completa de integração do Hub.
- Node.js 18+ e npm.
- Instale a versão mais recente do a AWS CLI partir da Referência de AWS CLI Comandos de Integrações Gerenciadas.
- Navegador web moderno (Chrome, Firefox, Safari, Edge)

Instale e configure o aplicativo

1. Baixe o aplicativo de demonstração de integrações gerenciadas.
2. Extraia o pacote:

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

3. Instale as dependências:

```
npm install
```

4. Crie um `.env` arquivo no diretório raiz:

```
# 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. Crie e inicie o aplicativo:

```
npm start
```

6. Acesse o aplicativo em:

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

Para obter informações sobre preços, consulte a seção Integrações gerenciadas na página de preços do Gerenciamento de AWS IoT dispositivos.

Hub externo de integrações gerenciadas

Visão geral do processo externo do Hub SDK

O processo de desativação do hub remove um hub do sistema de Nuvem AWS gerenciamento. Quando a nuvem envia uma DeleteManagedThingsolicitação, o processo atinge dois objetivos principais:

Ações do lado do dispositivo:

- Redefinir o estado interno do hub
- Exclua todos os dados salvos localmente
- Prepare o dispositivo para uma possível reintegração futura

Ações do lado da nuvem:

- Remova todos os recursos de nuvem associados ao hub
- Desconexão completa da conta anterior

Normalmente, os clientes iniciam a desativação do hub quando:

- Alterando a conta associada ao hub
- Substituindo um hub existente por um novo dispositivo

O processo garante uma transição limpa e segura entre as configurações do hub, permitindo o gerenciamento contínuo de dispositivos e a flexibilidade da conta.

Pré-requisitos

- Você deve ter um hub integrado. Para obter instruções, consulte Configuração de integração do Hub.
- No `iotmi_config.json` arquivo localizado em/data/aws/iotmi/config/, verifique se isso é `iot_provisioning_state` exibido`PROVISIONED`.
- Confirme se os certificados e chaves permanentes referenciados no `iotmi_config.json` existem em seus caminhos especificados.
- Certifique-se de que o Agente HubOnboarding, o Provisionador e o proxy MQTT estejam configurados e em execução corretamente.
- Verifique se o hub não tem dispositivos secundários. Use a DeleteManagedThingAPI para remover todos os dispositivos secundários antes de continuar.

Processo externo do Hub SDK

Siga estas etapas para desconectar o hub:

Recupere o ID do hub_managed_thing

O `iotmi_config.json` arquivo é usado para armazenar o ID do item gerenciado para um hub de integrações gerenciadas. Esse identificador é uma informação essencial que permite que o hub se comunique com o serviço de integrações AWS IoT gerenciadas. O ID do item gerenciado é armazenado na seção rw (leitura-gravação) do arquivo JSON, abaixo do campo. `managed_thing_id` Isso é visto no exemplo de configuração a seguir:

```
{
      "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
            }
        }
}
```

Enviar comando para o hub externo

Use as credenciais da sua conta e execute o comando com o `managed_thing_id` recuperado na seção anterior:

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

Verifique se o hub foi desligado

Use as credenciais da sua conta e execute o comando com o `managed_thing_id` recuperado na seção anterior:

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

Cenários de sucesso e fracasso

Cenário de sucesso

Se o comando para desligar o hub for bem-sucedido, o seguinte exemplo de resposta é esperado:

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

Além disso, o exemplo a seguir `iotmi_config.json` seria observado se o comando de desligamento do hub fosse bem-sucedido. Verifique se a seção rw contém somente **`iot_provisioning_state`** e opcionalmente metadados. A ausência de metadados é aceitável. `iot_provisioning_state`deve ser NOT_PROVISIONED.

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

Cenário de falha

Se o comando para desligar o hub não tiver sido bem-sucedido, o seguinte exemplo de resposta é esperado:

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

- Se ProvisioningStatusestiver`DELETE_IN_PROGRESS`, siga as instruções em Recuperação do Hub.
- Caso contrário ProvisioningStatus`DELETE_IN_PROGRESS`, o comando para desconectar o hub falhou na nuvem de integrações gerenciadas ou não foi recebido pela nuvem de integrações gerenciadas. Siga as instruções na recuperação do Hub.
- Se a desativação não for bem-sucedida, seu `iotmi_config.json` arquivo terá a aparência do arquivo de amostra abaixo.

```
{
      "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
          }
      }
}
```

(Opcional) Depois de desativar o Hub SDK

> **Important**
>
> Os cenários a seguir listam as ações opcionais a serem tomadas após a falha do SDK do Hub ou se você quiser reintegrar seu hub após a desativação.

Volte a bordo

Se a desativação foi bem-sucedida, integre seu SDK do Hub seguindo a Etapa 3: Crie uma coisa gerenciada (provisionamento de frota) e o resto do processo de integração.

Recuperação de hub

Sucesso na desativação do hub de dispositivos e falha na desativação da nuvem

Se a chamada GetManagedThingda API não retornar a `Managed Thing resource not found` mensagem, mas o arquivo `iotmi_config.json` for removido. Consulte Cenário de sucesso para ver um exemplo de arquivo json.

Para se recuperar desse cenário, consulte Forçar exclusão.

Falha na desativação do hub de dispositivos

Esse cenário ocorre quando o arquivo não `iotmi_config.json` é removido corretamente. Consulte Cenário de falha para ver um exemplo de arquivo json.

Para se recuperar desse cenário, consulte Forçar exclusão. Se ainda não `iotmi_config.json` estiver desligado, o hub deve ser redefinido para as configurações de fábrica.

A desativação do hub de dispositivos e a desativação da nuvem falham

Nesse cenário, ainda não `iotmi_config.json` está desligado e o status do hub é ou`ACTIVATED`. `DISCOVERED`

Para se recuperar desse cenário, consulte Forçar exclusão. Se a exclusão forçada falhar ou ainda não `iotmi_config.json` estiver desligada, o hub deverá ser redefinido para as configurações de fábrica.

O hub está offline e o status do hub é DELETE_IN_PROGRESS

Nesse cenário, o hub fica off-line e a nuvem recebe um comando de desligamento.

Para se recuperar desse cenário, consulte Forçar exclusão.

Forçar exclusão

Para excluir recursos da nuvem sem uma desativação bem-sucedida do hub de dispositivos, siga estas etapas. Essa operação pode resultar em inconsistência entre os estados da nuvem e do dispositivo, potencialmente causando problemas com operações futuras.

Chame a DeleteManagedThing API com o parâmetro hub `managed_thing_id` e force:

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

Em seguida, chame a GetManagedThing API e verifique se ela retorna`Managed Thing resource not found`. Isso confirma que os recursos da nuvem foram excluídos.

> **Note**
>
> Essa abordagem não é recomendada, pois pode levar a inconsistências entre os estados da nuvem e do dispositivo. Geralmente, é melhor garantir uma desativação bem-sucedida do hub de dispositivos antes de tentar excluir os recursos da nuvem.

Middleware específico de protocolo

> **Important**
>
> A documentação e o código fornecidos aqui descrevem uma implementação de referência do middleware. Ele não é fornecido a você como parte do SDK.

O middleware específico do protocolo tem um papel fundamental na interação com as pilhas de protocolos subjacentes. Os componentes de integração e controle de dispositivos do SDK do Hub de integrações gerenciadas o usam para interagir com o dispositivo final.

O middleware executa as seguintes funções.

- Abstrai as pilhas APIs de protocolos de dispositivos de diferentes fornecedores, fornecendo um conjunto comum de. APIs
- Fornece gerenciamento de execução de software, como agendador de threads, gerenciamento de filas de eventos e cache de dados.

Arquitetura de middleware

O diagrama de blocos abaixo representa a arquitetura do middleware Zigbee. A arquitetura dos middlewares de outros protocolos, como o Z-Wave, também é semelhante.

O middleware específico do protocolo tem três componentes principais.

- ACS Zigbee DPK: O Zigbee Device Porting Kit (DPK) é usado para fornecer abstração do hardware e do sistema operacional subjacentes, permitindo assim a portabilidade. Basicamente, isso pode ser considerado como a camada de abstração de hardware (HAL), que fornece

um conjunto comum APIs para controlar e se comunicar com os rádios Zigbee de diferentes fornecedores. O middleware Zigbee contém a implementação da API DPK para a estrutura de aplicativos Zigbee da Silicon Labs.

- Serviço ACS Zigbee: O serviço Zigbee é executado como um daemon dedicado. Ele inclui um manipulador de API que atende às chamadas de API de aplicativos clientes por meio dos canais IPC. O AIPC é usado como o canal IPC entre o adaptador Zigbee e o serviço Zigbee. Ele fornece outras funcionalidades, como lidar com os dois async/sync comandos, lidar com eventos do HAL e usar o ACS Event Manager para registrar/publicar eventos.
- Adaptador ACS Zigbee: O adaptador Zigbee é uma biblioteca em execução no processo do aplicativo (nesse caso, o aplicativo é o plug-in CDMB). O adaptador Zigbee fornece um conjunto APIs que é consumido por aplicativos clientes, como plug-ins de CDMB/Provisioner protocolo, para controlar e se comunicar com o dispositivo final.

End-to-end exemplo de fluxo de comando de middleware

Aqui está um exemplo do fluxo de comando por meio do middleware Zigbee.

Aqui está um exemplo do fluxo de comando por meio do middleware Z-Wave.

Organização de código de middleware específico para protocolos

Esta seção contém informações sobre a localização do código de cada componente dentro do `IotManagedIntegrationsDeviceSDK-Middleware` repositório. Veja a seguir um exemplo da estrutura de pastas nesse repositório.

```
./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
```

Tópicos

Organização de código de middleware Zigbee

O seguinte mostra a organização do código de middleware de referência do Zigbee.

Tópicos

ACS Zigbee DPK

O código do Zigbee DPK está localizado dentro do diretório listado no exemplo abaixo:

```
./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
```

SDK Zigbee da Silicon Labs

O SDK do Silicon Labs é apresentado dentro da `IotManagedIntegrationsDeviceSDK-Middleware/example-iot-ace-z3-gateway` pasta. Essa camada ACS Zigbee DPK é implementada para este SDK do Silicon Labs.

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

Serviço ACS Zigbee

O código do Serviço Zigbee está localizado dentro da `IotManagedIntegrationsDeviceSDK-Middleware/example-iot-ace-general/middleware/zigbee/` pasta. As `include` subpastas `src` e neste local contêm todos os arquivos relacionados ao serviço ACS Zigbee.

```
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
```

Adaptador ACS Zigbee

O código do adaptador ACS Zigbee está localizado dentro da pasta. `IotManagedIntegrationsDeviceSDK-Middleware/example-iot-ace-general/middleware/zigbee/api` As `include` subpastas `src` e neste local contêm todos os arquivos relacionados à biblioteca ACS Zigbee Adaptor.

```
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
```

Organização de código de middleware Z-Wave

A seguir, mostramos a organização do código de middleware de referência do Z-wave.

Tópicos

- ACS Z-Wave DPK
- Silicon Labs ZWare e Zip Gateway
- Serviço ACS Z-Wave
- Adaptador ACS Z-Wave

ACS Z-Wave DPK

O código do Z-Wave DPK está localizado dentro da pasta. `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 e Zip Gateway

O código dos laboratórios Silicon ZWare e do Zip Gateway está localizado dentro da `IotManagedIntegrationsDeviceSDK-Middleware/example-iot-ace-z3-gateway` pasta. Essa camada ACS Z-Wave DPK é implementada para os gateways C e Zip Z-Wave. APIs

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

Serviço ACS Z-Wave

O código do serviço Z-Wave está localizado dentro da pasta listada na `IoTmanagedintegrationsMiddlewares/exampleiot-ace-zwave-mw/` pasta. As `include` pastas `src` e neste local contêm todos os arquivos relacionados ao serviço ACS Z-Wave.

```
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
```

Adaptador ACS Z-Wave

O código do adaptador ACS Zigbee está localizado dentro da pasta. `IotManagedIntegrationsDeviceSDK-Middleware/example-iot-ace-zwave-mw/cli/` A `include` pasta `src` e neste local contém todos os arquivos relacionados à biblioteca do adaptador ACS Z-Wave.

```
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
```

Integre o middleware com o SDK

A integração do middleware no novo hub é discutida nas seções a seguir.

Tópicos

- Integração da API do kit de portabilidade de dispositivos (DPK)
- Implementação de referência e organização do código

Integração da API do kit de portabilidade de dispositivos (DPK)

Para integrar o SDK de qualquer fornecedor de chipset ao middleware, uma interface de API padrão é fornecida pela camada intermediária do DPK (kit de portabilidade de dispositivos). Os provedores de serviços de integrações gerenciadas ou ODMs precisam implementá-las APIs com base no SDK do fornecedor suportado pelos chipsets de Zigbee/Z-wave/Wi Wi-Fi usados em seus hubs de IoT.

Implementação de referência e organização do código

Com exceção do middleware, todos os outros componentes do Device SDK, como as integrações gerenciadas Device Agent e Common Data Model Bridge (CDMB), podem ser usados sem modificações e só precisam ser compilados de forma cruzada.

A implementação do middleware é baseada no SDK do Silicon Labs para Zigbee e Z-Wave. Se os chipsets Z-Wave e Zigbee usados no novo hub forem suportados pelo SDK do Silicon Labs presente no middleware, o middleware de referência poderá ser usado sem modificações. Você só precisa fazer a compilação cruzada do middleware e ele poderá ser executado no novo hub.

O DPK (kit de portabilidade de dispositivos) APIs para Zigbee pode ser encontrado em `acehal_zigbee.c`, e a implementação de referência do DPK APIs está presente dentro da pasta. `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
```

O DPK APIs para Z-Wave pode ser encontrado em `acehal_zwave.c` e a implementação de referência do DPK APIs está presente dentro da pasta. `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
```

Como ponto de partida para implementar a camada DPK para o SDK de um fornecedor diferente, a implementação de referência pode ser usada e modificada. As duas modificações a seguir serão necessárias para oferecer suporte ao SDK de um fornecedor diferente:

1. Substitua o SDK do fornecedor atual pelo SDK do novo fornecedor no repositório.
2. Implemente o middleware DPK (kit de portabilidade de dispositivos) de APIs acordo com o SDK do novo fornecedor.

Integrações gerenciadas SDK para dispositivos finais

Crie uma plataforma de IoT que conecte dispositivos inteligentes a integrações gerenciadas e comandos de processos por meio de uma interface de controle unificada. O SDK do dispositivo final se integra ao firmware do seu dispositivo e fornece configuração simplificada com os componentes de ponta do SDK, além de conectividade segura e gerenciamento de AWS IoT Core dispositivos. AWS IoT Baixe a versão mais recente do SDK do dispositivo final no Console de gerenciamento da AWS

Este guia descreve como implementar o SDK do dispositivo final em seu firmware. Analise a arquitetura, os componentes e as etapas de integração para começar a criar sua implementação.

Tópicos

O que é o SDK do dispositivo final?

O que é o SDK do dispositivo final?

O SDK do dispositivo final é uma coleção de código-fonte, bibliotecas e ferramentas fornecidas pela AWS IoT. Criado para ambientes com recursos limitados, o SDK oferece suporte a dispositivos com apenas 512 KB de RAM e 4 MB de memória flash, como câmeras e purificadores de ar executados em Linux incorporado e sistemas operacionais em tempo real (RTOS). Baixe a versão mais recente do SDK do dispositivo final no AWS IoT Management Console.

Componentes principais

O SDK combina um agente MQTT para comunicação na nuvem, um manipulador de tarefas para gerenciamento de tarefas e uma integração gerenciada, o Data Model Handler. Esses componentes

trabalham juntos para fornecer conectividade segura e tradução automatizada de dados entre seus dispositivos e integrações gerenciadas.

Para obter os requisitos técnicos detalhados, consulte Referência técnica o.

Arquitetura e componentes do SDK do dispositivo final

Esta seção descreve a arquitetura do SDK do dispositivo final e como seus componentes interagem com suas funções C de baixo nível. O diagrama a seguir ilustra os componentes principais e seus relacionamentos na estrutura do SDK.

Componentes do SDK do dispositivo final

A arquitetura do SDK do dispositivo final contém esses componentes para integração de recursos de integrações gerenciadas:

Provisionado

: Cria recursos de dispositivos na nuvem de integrações gerenciadas, incluindo certificados de dispositivos e chaves privadas para comunicação segura com o MQTT. Essas credenciais estabelecem conexões confiáveis entre seu dispositivo e as integrações gerenciadas.

Agente MQTT

: Gerencia conexões MQTT por meio de uma biblioteca cliente C segura para threads. Esse processo em segundo plano manipula filas de comandos em ambientes de vários segmentos, com tamanhos de fila configuráveis para dispositivos com restrição de memória. As mensagens são roteadas por meio de integrações gerenciadas para processamento.

Manipulador de trabalhos

: Processa atualizações over-the-air (OTA) para firmware de dispositivos, patches de segurança e entrega de arquivos. Esse serviço integrado gerencia as atualizações de software para todos os dispositivos registrados.

Manipulador do modelo de dados

: Traduz as operações entre integrações gerenciadas e suas funções C de baixo nível usando AWS a implementação do Matter Data Model. Para obter mais informações, consulte a documentação do Matter em GitHub.

Chaves e certificados

Gerencia operações criptográficas por meio da API PKCS #11, suportando módulos de segurança de hardware e implementações de software, como o core. PKCS11 Essa API lida com operações de certificado para componentes como o Provisionee e o MQTT Agent durante conexões TLS.

Provisionado

O provisionado é um componente das integrações gerenciadas que permite o provisionamento da frota por reclamação. Com o provisionado, você provisiona seus dispositivos com segurança. O SDK cria os recursos necessários para o provisionamento de dispositivos, o que inclui o certificado do dispositivo e as chaves privadas que são obtidas da nuvem de integrações gerenciadas. Quando quiser provisionar seus dispositivos, ou se houver alguma alteração que possa exigir que você reprovisione seus dispositivos, você pode usar o provisionado.

Tópicos

Fluxo de trabalho do provisionado

O processo requer configuração no lado da nuvem e do dispositivo. Os clientes configuram os requisitos de nuvem, como endpoints personalizados, perfis de provisionamento e itens gerenciados. Na primeira inicialização do dispositivo, o provisionado:

1. Conecta-se ao endpoint de integrações gerenciadas usando um certificado de solicitação
2. Valida os parâmetros do dispositivo por meio de ganchos de provisionamento de frota
3. Obtém e armazena um certificado permanente e uma chave privada no dispositivo

4. O dispositivo usa o certificado permanente para se reconectar
5. Descobre e carrega recursos do dispositivo para integrações gerenciadas

Após o provisionamento bem-sucedido, o dispositivo se comunica diretamente com as integrações gerenciadas. O provisionado é ativado somente para tarefas de reprovisionamento.

Definição de variáveis de ambiente

Defina as seguintes AWS credenciais em seu ambiente de nuvem:

```
$ 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
```

Registre um endpoint personalizado

Use o comando RegisterCustomEndpointda API em seu ambiente de nuvem para criar um endpoint personalizado para device-to-cloud comunicação.

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

Exemplo de resposta

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

> **Note**
>
> Armazene o endereço do endpoint para configurar os parâmetros de provisionamento. Use a `GetCustomEndpoint` API para retornar informações do endpoint. Para obter mais informações, consulte GetCustomEndpointAPI e RegisterCustomEndpointAPI no Guia de referência da API de integrações gerenciadas.

Crie um perfil de aprovisionamento

Crie um perfil de aprovisionamento que defina o método de aprovisionamento da sua frota. Execute a CreateProvisioningProfileAPI em seu ambiente de nuvem para retornar um certificado de solicitação e uma chave privada para autenticação do dispositivo:

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

Exemplo de resposta

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

Você pode implementar a biblioteca de abstração da PKCS11 plataforma principal (PAL) para fazer com que a PKCS11 biblioteca principal funcione com seu dispositivo. As portas PKCS11 PAL principais devem fornecer um local para armazenar o certificado de solicitação e a chave privada. Usando esse recurso, você pode armazenar com segurança a chave privada e o certificado do dispositivo. Você pode armazenar a chave privada e o certificado em um módulo de segurança de hardware (HSM) ou em um módulo de plataforma confiável (TPM).

Crie uma coisa gerenciada

Registre seu dispositivo na nuvem de integrações gerenciadas usando a CreateManagedThingAPI. Inclua o número de série (SN) e o código universal do produto (UPC) do seu dispositivo:

```
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;"
```

Veja a seguir um exemplo de resposta da API.

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

A API retorna o ID do item gerenciado que pode ser usado para validação de provisionamento. Você precisará fornecer o número de série (SN) do dispositivo e o código universal do produto (UPC), que correspondem ao item gerenciado aprovado durante a transação de provisionamento. A transação retorna um resultado semelhante ao seguinte:

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

Provisionamento Wi-Fi do usuário do SDK

Fabricantes de dispositivos e provedores de soluções têm seu próprio serviço proprietário de provisionamento de Wi-Fi para receber e configurar credenciais de Wi-Fi. O serviço de provisionamento de Wi-Fi envolve o uso de aplicativos móveis dedicados, conexões Bluetooth Low Energy (BLE) e outros protocolos proprietários para transferir com segurança as credenciais de Wi-Fi para o processo de configuração inicial.

O consumidor do SDK do dispositivo final deve implementar o serviço de provisionamento Wi-Fi e o dispositivo pode se conectar a uma rede Wi-Fi.

Provisionamento de frota por reclamação

Usando o provisionado, o usuário final pode provisionar um certificado exclusivo e registrá-lo em integrações gerenciadas usando o provisionamento por solicitação.

O ID do cliente pode ser adquirido a partir da resposta do modelo de aprovisionamento ou do certificado do dispositivo. `<common name>“_”<serial number>`

Capacidades de coisas gerenciadas

O provisionado descobre os recursos do item gerenciado e, em seguida, carrega os recursos para as integrações gerenciadas. Ele disponibiliza os recursos para que aplicativos e outros serviços acessem. Dispositivos, outros clientes da Web e serviços podem atualizar os recursos usando o MQTT e o tópico reservado do MQTT ou o HTTP usando a API REST.

Over-the-Air atualizações

Visão geral da arquitetura OTA

O processo de atualização Over-the-Air (OTA) envolve vários componentes trabalhando juntos para fornecer atualizações de firmware aos seus dispositivos. O diagrama a seguir ilustra como uma solicitação de atualização OTA é tratada por meio da interação entre o SDK do dispositivo final, o SDK do Hub e o recurso.

A arquitetura de atualização OTA consiste nos seguintes componentes:

- Cliente: carrega documentos de trabalho em um bucket do S3 e inicia atualizações via API
- Serviço OTA: lida com a criação, validação e gerenciamento de tarefas
- AWS IoT Trabalhos: gerencia a execução e a entrega de trabalhos aos dispositivos
- Dispositivos: receba e aplique atualizações usando o Harmony SDK

Pré-requisitos

Antes de criar tarefas OTA, você deve configurar os seguintes pré-requisitos:

Configurar o acesso ao Amazon S3

Para habilitar as atualizações do OTA, você deve carregar seus documentos de trabalho em um bucket do Amazon S3 e configurar as permissões de acesso apropriadas:

1. Carregue seu documento de trabalho OTA em um bucket S3
2. Adicione uma política de bucket do Amazon S3 que conceda às integrações gerenciadas acesso aos seus documentos de trabalho:

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"
      ]
    }
  ]
}
```

Implementar tarefas Over-the-Air (OTA)

Você pode criar tarefas OTA de duas maneiras, dependendo dos requisitos de atualização e da estratégia de segmentação do dispositivo:

Atualizações únicas de tarefas OTA

Uma tarefa OTA única contém uma lista estática de destinos (`ManagedThings`) para realizar atualizações de OTA. Você pode adicionar até 100 alvos por vez. O fluxo de trabalho usa AWS IoT Jobs with Fleet Indexing enquanto mantém a camada de abstração de integrações gerenciadas.

Use o exemplo a seguir para criar uma tarefa OTA única:

```
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"}'
```

Atualizações contínuas de tarefas OTA

O fluxo de trabalho de agrupamento OTA (Over-the-Air) permite que você implante atualizações de firmware em grupos de dispositivos com base em atributos específicos, usando AWS IoT Jobs

with Fleet Indexing, mantendo a camada de abstração de integrações gerenciadas. As tarefas OTA contínuas usam uma sequência de caracteres de consulta em vez de alvos específicos. Todos os dispositivos que correspondem aos critérios de consulta passam por atualizações OTA, e os critérios de consulta são continuamente reavaliados. Os alvos correspondentes terão implantações de trabalho.

Configurar pré-requisitos

Antes de criar tarefas OTA contínuas, preencha estes pré-requisitos:

1. Crie uma coisa gerenciada chamando a CreateManagedThingAPI e realize o provisionamento da frota.
2. Adicione atributos de metadados às suas coisas gerenciadas para direcionamento de consultas.

Adicione atributos e metadados ao `ManagedThing` uso da UpdateManagedThingAPI:

```
aws iotmanagedintegrations update-managed-thing \
  --managed-thing-id "YOUR_MANAGED_THING_ID" \
  --meta-data '{"owner":"managedintegrations","version":"1.0"}'
```

Use o exemplo a seguir para criar uma tarefa OTA contínua:

```
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"}'
```

Entenda o fluxo de trabalho OTA

O fluxo de trabalho de atualização contínua do OTA segue estas etapas:

1. Você atualiza coisas gerenciadas com atributos usando a UpdateManagedThingAPI.
2. Crie um trabalho OTA com uma sequência de caracteres de consulta direcionada a atributos específicos do dispositivo.
3. O serviço OTA cria um Thing Group dinâmico AWS IoT Core com base nos atributos de consulta

4. O IoT Jobs executa atualizações em dispositivos correspondentes
5. Você monitora o progresso por meio da ListOtaTaskExecutionsAPI ou das notificações OTA por meio do stream do Kinesis (se ativado).

Diferenças entre integrações gerenciadas OTA e IoT Jobs

A distinção fundamental entre integrações gerenciadas OTA e IoT Jobs está na orquestração e automação de serviços. Integrações gerenciadas O OTA fornece uma solução de serviço único que elimina a complexidade da coordenação de vários serviços.

O que as integrações gerenciadas OTA faz automaticamente:

- Criação dinâmica de grupos de coisas: gera grupos de AWS IoT Core coisas automaticamente com base em seus critérios de consulta.
- Resolução alvo: traduz cadeias de caracteres de consulta (exemplo:`attributes.owner=managedintegrations`) em destinos reais do dispositivo.
- Integração de serviços: coordena perfeitamente entre AWS IoT Core trabalhos de IoT e serviços de indexação de frotas.
- Gerenciamento do ciclo de vida: gerencia todo o fluxo de trabalho do OTA, desde a criação até o monitoramento da execução.

O que o MI OTA elimina:

- Criando grupos de coisas em AWS IoT Core.
- Adicionar coisas aos grupos.
- Criação de empregos de IoT.

Integrações gerenciadas O OTA lida com todas as três operações internamente com base em sua string de consulta, descobrindo automaticamente dispositivos que atendem aos seus critérios, criando trabalhos de IoT nos bastidores e orquestrando o fluxo de trabalho completo do OTA sem exigir que você interaja diretamente com vários serviços. AWS

Configuração das configurações de tarefas OTA

Você pode criar configurações para atualizações OTA para controlar como as atualizações são lançadas nos dispositivos, definir condições de aborto e configurar tempos limite.

Exemplo: CreateOtaTaskConfiguration.

Use o exemplo a seguir para criar uma configuração de tarefa OTA:

```
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"
```

Aplicar configurações às tarefas OTA

Depois que a configuração for criada, você receberá uma `taskConfigurationId` que será adicionada à sua `CreateOtaTask` solicitação junto com configurações adicionais:

```
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"}'
```

Monitore as notificações OTA

Você pode monitorar as atualizações do OTA usando dois métodos diferentes:

Notificações push por meio do Kinesis Data Streams

Quando as notificações OTA são ativadas, os eventos de status de atualização são automaticamente enviados para seu stream do Kinesis. Isso fornece visibilidade em tempo real do progresso das atualizações de firmware em todos os dispositivos.

Monitore com ListOtaTaskExecutions API

Você pode usar a ListOtaTaskExecutionsAPI para verificar manualmente o status das atualizações do OTA para suas coisas gerenciadas:

```
aws iotmanagedintegrations list-ota-task-executions \
  --task-id "task-123456789" \
  --max-results 25
```

A resposta fornece um status de execução detalhado para cada coisa gerenciada:

```
{
  "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"
}
```

Essa API permite que você recupere o status detalhado de execução de cada item gerenciado direcionado por uma tarefa específica do OTA, incluindo registros de data e hora e status atual.

Processar documentos de trabalho

Quando você cria uma tarefa OTA, o manipulador de trabalhos executa as etapas a seguir no seu dispositivo. Quando uma atualização está disponível, ela solicita o documento do trabalho pelo MQTT.

1. Se inscreve nos tópicos de notificação do MQTT.
2. Chama a StartNextPendingJobExecutionAPI para trabalhos pendentes.
3. Recebe os documentos de trabalho disponíveis.
4. Processa atualizações com base nos tempos limite especificados.

Usando o manipulador de trabalhos, o aplicativo pode determinar se deve agir imediatamente ou esperar até um período de tempo limite especificado.

Implemente o agente OTA

Ao receber o documento de trabalho de integrações gerenciadas, você deve ter uma implementação de seu próprio agente OTA que processe o documento de trabalho, baixe atualizações e execute qualquer operação de instalação. O agente OTA precisa executar as seguintes etapas:

1. Analise documentos de trabalho para o firmware Amazon URLs S3.
2. Baixe as atualizações de firmware por meio de HTTP.
3. Verifique as assinaturas digitais.
4. Instale atualizações validadas.
5. Ligue `iotmi\_JobsHandler\_updateJobStatus` com `SUCCESS` ou `FAILED` status.

Quando seu dispositivo conclui com êxito a operação OTA, ele deve chamar a `iotmi\_JobsHandler\_updateJobStatus` API com o status de `JobSucceeded` Para relatar um trabalho bem-sucedido.

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

Gerador de código de modelo de dados

Saiba como usar o gerador de código para o modelo de dados. O código gerado pode ser usado para serializar e desserializar os modelos de dados que são trocados entre a nuvem e o dispositivo.

O repositório do projeto contém uma ferramenta de geração de código para criar manipuladores de modelos de dados de código C. Os tópicos a seguir descrevem o gerador de código e o fluxo de trabalho.

Tópicos

- Processo de geração de código
- Configuração do ambiente
- Gere código para dispositivos

Processo de geração de código

O gerador de código cria arquivos de origem C a partir de três entradas principais: AWS'implementação do Matter Data Model (arquivo.matter) da plataforma avançada Zigbee Cluster Library (ZCL), um plug-in Python que manipula o pré-processamento e modelos Jinja2 que definem a estrutura do código. Durante a geração, o plug-in Python processa seus arquivos.matter adicionando definições de tipo globais, organizando tipos de dados com base em suas dependências e formatando as informações para renderização de modelos.

A imagem a seguir descreve o gerador de código que cria os arquivos de origem C.

O SDK do dispositivo final inclui plug-ins Python e modelos Jinja2 que funcionam no projeto. codegen.pyconnectedhomeip Essa combinação gera vários arquivos C para cada cluster com base na entrada do arquivo.matter.

Os subtópicos a seguir descrevem esses arquivos.

- Plug-in Python
- Modelos Jinja2
- (Opcional) Esquema personalizado

Plug-in Python

O gerador de código,`codegen.py`, analisa os arquivos.matter e envia as informações como objetos Python para o plug-in. O arquivo do plug-in `iotmi_data_model.py` pré-processa esses dados e renderiza fontes com os modelos fornecidos. O pré-processamento inclui:

1. Adicionar informações não disponíveis em`codegen.py`, como tipos globais
2. Executando classificação topológica em tipos de dados para estabelecer a ordem de definição correta

> **Note**
>
> A classificação topológica garante que os tipos dependentes sejam definidos após suas dependências, independentemente da ordem original.

Modelos Jinja2

O SDK do dispositivo final fornece modelos Jinja2 personalizados para manipuladores de modelos de dados e funções C de baixo nível.

Modelos Jinja2

Modelo	Fonte gerada	Observações
`cluster.h.jinja`	`iotmi_device_<cluster>.h`	Cria arquivos de cabeçalho de função C de baixo nível.

Modelo	Fonte gerada	Observações
`cluster.c.jinja`	`iotmi_device_<cluster>.c`	Implemente e registre ponteiros de função de retorno de chamada com o Data Model Handler.
`cluster_type_helpers.h.jinja`	`iotmi_device_type_helpers_<cluster>.h`	Define protótipos de funções para tipos de dados.
`cluster_type_helpers.c.jinja`	`iotmi_device_type_helpers_<cluster>.c`	Gera protótipos de funções de tipo de dados para enumeraçõ es, bitmaps, listas e estruturas específicas do cluster.
`iot_device_dm_types.h.jinja`	`iotmi_device_dm_types.h`	Define os tipos de dados C para tipos de dados globais.
`iot_device_type_helpers_global.h.jinja`	`iotmi_device_type_helpers_global.h`	Define os tipos de dados C para operações globais.
`iot_device_type_helpers_global.c.jinja`	`iotmi_device_type_helpers_global.c`	Declara tipos de dados padrão, incluindo booleanos , inteiros, ponto flutuante , cadeias de caracteres, bitmaps, listas e estruturas.

(Opcional) Esquema personalizado

O SDK do dispositivo final combina o processo padronizado de geração de código com o esquema personalizado. Isso permite a extensão do Matter Data Model para seus dispositivos e software de dispositivos. Esquemas personalizados podem ajudar a descrever os recursos de device-to-cloud comunicação do dispositivo.

Para obter informações detalhadas sobre modelos de dados de integrações gerenciadas, incluindo formato, estrutura e requisitos, consulteModelo de dados de integrações gerenciadas.

Use a `codegen.py` ferramenta para gerar arquivos de origem C para o esquema personalizado, da seguinte forma:

> **Note**
>
> Cada cluster personalizado exige o mesmo ID de cluster para os três arquivos a seguir.

- Crie um esquema personalizado em um `JSON` formato que forneça uma representação de clusters para geração de relatórios de recursos para criar novos clusters personalizados na nuvem. Um arquivo de amostra está localizado em`codegen/custom_schemas/custom.SimpleLighting@1.0`.
- Crie o arquivo de definição ZCL (Zigbee Cluster Library) em `XML` formato que contenha as mesmas informações do esquema personalizado. Use a ferramenta ZAP para gerar seus arquivos Matter IDL a partir do ZCL XML. Um arquivo de amostra está localizado em`codegen/zcl/custom.SimpleLighting.xml`.
- A saída da ferramenta ZAP é `Matter IDL File (.matter)` e ela define os clusters Matter correspondentes ao seu esquema personalizado. Essa é a entrada da `codegen.py` ferramenta para gerar arquivos de origem C para o SDK do dispositivo final. Um arquivo de amostra está localizado em`codegen/matter_files/custom-light.matter`.

Para obter instruções detalhadas sobre como integrar modelos de dados de integrações gerenciadas personalizadas em seu fluxo de trabalho de geração de código, consulteGere código para dispositivos.

Configuração do ambiente

Saiba como configurar seu ambiente para usar o gerador `codegen.py` de código.

Tópicos

- Pré-requisitos
- Configure o ambiente

Pré-requisitos

Instale os seguintes itens antes de configurar seu ambiente:

- Git
- Python 3.10 ou posterior
- Poesia 1.2.0 ou superior

Configure o ambiente

Use o procedimento a seguir para configurar seu ambiente para usar o gerador de código codegen.py.

1. Baixe a versão mais recente do SDK do dispositivo final no Console de gerenciamento da AWS.
2. Configure o ambiente do Python. O projeto codegen é baseado em python e usa Poetry para gerenciamento de dependências.
   - Instale as dependências do projeto usando poesia no `codegen` diretório:

     ```
     poetry run poetry install --no-root
     ```
3. Configure seu repositório.
   a. Clone o connectedhomeiprepositório. Ele usa o `codegen.py` script localizado na `connectedhomeip/scripts/` pasta para geração de código. Para obter mais informações, consulte connectedhomeip on. GitHub

      ```
      git clone -b v1.4.0.0 https://github.com/project-chip/connectedhomeip.git
      ```
   b. Clone-o no mesmo nível da sua pasta `IoT-managed-integrations-End-Device-SDK` raiz. Sua estrutura de pastas deve corresponder ao seguinte:

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

> **Note**
>
> Você não precisa clonar submódulos recursivamente.

Gere código para dispositivos

Crie código C personalizado para seus dispositivos usando as ferramentas de geração de código de integrações gerenciadas. Esta seção descreve como gerar código a partir de arquivos de amostra incluídos no SDK ou de suas próprias especificações. Aprenda a usar os scripts de geração, entender o processo de fluxo de trabalho e criar um código que atenda aos requisitos do seu dispositivo.

Tópicos

Pré-requisitos

1. Python 3.10 ou superior.

2. Comece com um arquivo.matter para geração de código. O SDK do dispositivo final fornece dois arquivos de amostra no`codegen/matter_files folder`:

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

> **Note**
>
> Esses arquivos de amostra geram código para clusters de aplicativos de demonstração.

Gerar código

Execute este comando para gerar código na pasta out:

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

Gere código para arquivos.matter personalizados

Para gerar o código para um `.matter` arquivo específico ou fornecer seu próprio `.matter` arquivo, execute as tarefas a seguir.

Para gerar o código para arquivos.matter personalizados

1. Prepare seu arquivo.matter
2. Execute o comando de geração:

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

(Opcional) Para gerar código com esquema personalizado

1. Prepare seu esquema personalizado em formato `JSON`
2. Execute o comando de geração:

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

Os comandos acima usam vários componentes para transformar seu `.matter` arquivo em `C` código:

- `codegen.py`do projeto ConnectedHomeIP
- Plugin Python localizado em `codegen/py_scripts/iotmi_data_model.py`
- Modelos Jinja2 da pasta `codegen/py_scripts/templates`

O plug-in define as variáveis a serem passadas para os modelos Jinja2, que são então usados para gerar a saída final do código C. Adicionar a `--format` bandeira aplica o formato Clang ao código gerado.

fluxo de trabalho de geração de código

O processo de geração de código organiza suas estruturas de dados de arquivos.matter usando funções utilitárias e classificação topológica. `topsort.py` Isso garante a ordenação adequada dos tipos de dados e suas dependências.

Em seguida, o script combina as especificações do arquivo.matter com o processamento do plug-in Python para extrair e formatar as informações necessárias. Por fim, ele aplica a formatação do modelo Jinja2 para criar a saída final do código C.

Esse fluxo de trabalho garante que os requisitos específicos do dispositivo do arquivo.matter sejam traduzidos com precisão em um código C funcional que se integre ao sistema de integrações gerenciadas.

Função C de baixo nível APIs

Integre o código específico do seu dispositivo com integrações gerenciadas usando a função C de baixo nível fornecida. APIs Esta seção descreve as operações de API disponíveis para cada cluster no modelo de AWS dados para interações eficientes entre dispositivo e nuvem. Saiba como implementar funções de retorno de chamada, emitir eventos, notificar alterações de atributos e registrar clusters para os endpoints do seu dispositivo.

Os principais componentes da API incluem:

1. Estruturas de ponteiro de função de retorno de chamada para atributos e comandos
2. Funções de emissão de eventos
3. Funções de notificação de alteração de atributo
4. Funções de registro de cluster

Ao implementá-los APIs, você cria uma ponte entre as operações físicas do seu dispositivo e os recursos de nuvem de integrações gerenciadas, garantindo comunicação e controle contínuos.

A seção a seguir ilustra a API do OnOffcluster.

OnOff API de cluster

O OnOff.xmlcluster oferece suporte a esses atributos e comandos:.

- Atributos:
  - `OnOff (boolean)`
  - `GlobalSceneControl (boolean)`
  - `OnTime (int16u)`
  - `OffWaitTime (int16u)`
  - `StartUpOnOff (StartUpOnOffEnum)`
- Comandos:

- `Off : () -> Status`
- `On : () -> Status`
- `Toggle : () -> Status`
- `OffWithEffect : (EffectIdentifier: EffectIdentifierEnum, EffectVariant: enum8) -> Status`
- `OnWithRecallGlobalScene : () -> Status`
- `OnWithTimedOff : (OnOffControl: OnOffControlBitmap, OnTime: int16u, OffWaitTime: int16u) -> Status`

Para cada comando, fornecemos o ponteiro de função mapeado 1:1 que você pode usar para conectar sua implementação.

Todos os retornos de chamada para atributos e comandos são definidos em uma estrutura C com o nome do cluster.

Exemplo de estrutura C

```
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);
};
```

Além da estrutura C, as funções de relatório de alterações de atributos são definidas para todos os atributos.

```
/* 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);
```

As funções de relatório de eventos são definidas para todos os eventos específicos do cluster. Como o OnOff cluster não define nenhum evento, abaixo está um exemplo do `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);
```

Cada cluster também tem uma função de registro.

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

O ponteiro do usuário passado para a função de registro será passado para as funções de retorno de chamada.

Interações de recursos e dispositivos em integrações gerenciadas

Esta seção descreve a função da implementação da função C e a interação entre o dispositivo e o recurso do dispositivo de integrações gerenciadas.

Tópicos

Manipulando comandos remotos

Os comandos remotos são gerenciados pela interação entre o SDK do dispositivo final e o recurso. As ações a seguir descrevem um exemplo de como você pode acender uma lâmpada usando essa interação.

O cliente MQTT recebe a carga e passa para o Data Model Handler

> Quando você envia um comando remoto, o cliente MQTT recebe a mensagem das integrações gerenciadas no formato JSON. Em seguida, ele passa a carga para o manipulador do modelo de dados. Por exemplo, digamos que você queira usar integrações gerenciadas para acender uma lâmpada. A lâmpada tem um endpoint #1 que suporta o OnOff cluster. Nesse caso, quando você envia o comando para acender a lâmpada, as integrações gerenciadas enviam uma solicitação pelo MQTT para o dispositivo, informando que ele deseja invocar o comando On no endpoint #1.

O Data Model Handler verifica as funções de retorno de chamada e as invoca

> O Data Model Handler analisa a solicitação JSON. Se a solicitação contiver propriedades ou ações, o Data Model Handler encontrará os endpoints e invocará sequencialmente as funções de retorno de chamada correspondentes. Por exemplo, no caso da lâmpada, quando o Data Model Handler recebe a mensagem MQTT, ele verifica se a função de retorno de chamada correspondente ao comando On definido no OnOff cluster está registrada no endpoint #1.

A implementação do manipulador e da função C executa o comando

> O Data Model Handler chama as funções de retorno de chamada apropriadas encontradas e as invoca. A implementação da Função C então chama as funções de hardware correspondentes para controlar o hardware físico e retorna o resultado da execução. Por exemplo, no caso da lâmpada, o Data Model Handler chama a função de retorno de chamada e armazena o resultado da execução. Como resultado, a função de retorno de chamada liga a lâmpada.

O Data Model Handler retorna o resultado da execução

Depois que todas as funções de retorno de chamada forem chamadas, o Data Model Handler combina todos os resultados. Em seguida, ele empacota a resposta no formato JSON e publica o resultado na nuvem de integrações gerenciadas usando o cliente MQTT. No caso da lâmpada, a mensagem MQTT na resposta conterá o resultado de que a lâmpada foi ligada pela função de retorno de chamada.

Lidando com eventos não solicitados

Eventos não solicitados também são tratados pela interação entre o SDK do dispositivo final e o recurso. As ações a seguir descrevem como.

O dispositivo envia uma notificação para o Data Model Handler

Quando ocorre uma alteração de propriedade ou evento, como quando um botão físico é pressionado no dispositivo, a implementação da função C gera uma notificação de evento não solicitada e chama a função de notificação correspondente para enviar a notificação ao manipulador do modelo de dados.

O Data Model Handler traduz a notificação

O Data Model Handler manipula a notificação recebida e a traduz para o modelo de AWS dados.

O Data Model Handler publica notificação na nuvem

O Data Model Handler então publica um evento não solicitado na nuvem de integrações gerenciadas usando o cliente MQTT.

Comece a usar o SDK do dispositivo final

Siga estas etapas para executar o SDK do dispositivo final em um dispositivo Linux. Esta seção orienta você na configuração do ambiente, na configuração da rede, na implementação da função de hardware e na configuração do endpoint.

> **Important**
>
> Os aplicativos de demonstração no `examples` diretório e sua implementação de Platform Abstraction Layer (PAL) em `platform/posix` são apenas para referência. Não os use em ambientes de produção.

Analise cuidadosamente cada etapa do procedimento a seguir para garantir a integração adequada do dispositivo com integrações gerenciadas.

Integre o SDK do dispositivo final

1. Configurar EC2 instância da Amazon

   Faça login Console de gerenciamento da AWS e execute uma EC2 instância da Amazon usando uma Amazon Linux AMI. Consulte Comece a usar a Amazon EC2 no Guia do usuário do Amazon Elastic Container Registry.

2. Configurar o ambiente de construção

   Crie o código no Amazon Linux 2023/x86_64 como seu host de desenvolvimento. Instale as dependências de compilação necessárias:

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

3. (Opcional) Configurar a rede

   O SDK do dispositivo final é melhor usado com hardware físico. Se estiver usando a Amazon EC2, não siga esta etapa.

   Se você não estiver usando a Amazon EC2 antes de usar o aplicativo de amostra, inicialize a rede e conecte seu dispositivo a uma rede Wi-Fi disponível. Conclua a configuração da rede antes do provisionamento do dispositivo:

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

4. Configurar parâmetros de provisionamento

> **Note**
>
> Siga o Provisionee para obter o certificado de solicitação e a chave privada antes de prosseguir.

Modifique o arquivo de configuração `example/project_name/device_config.sh` com os seguintes parâmetros de provisionamento:

Parâmetros de provisionamento

Parâmetros macro	Description	Como obter essas informações
`IOTMI_ROOT_CA_PATH`	O arquivo raiz do certificado CA.	Você pode baixar esse arquivo na seção Baixar o certificado Amazon Root CA no guia do AWS IoT Core desenvolvedor.
`IOTMI_CLAIM_CERTIFICATE_PATH`	O caminho para o arquivo do certificado de solicitação.	Para obter o certificado de solicitação e a chave privada, crie um perfil de provisionamento usando a CreateProvisioningProfileAPI. Para instruções, consulte Crie um perfil de aprovisionamento.
`IOTMI_CLAIM_PRIVATE_KEY_PATH`	O caminho para o arquivo de chave privada da solicitação.	
`IOTMI_MANAGEDINTEGRATIONS_ENDPOINT`	URL do endpoint para integrações gerenciadas.	Para obter o endpoint de integrações gerenciadas, use a RegisterCustomEndpointAPI. Para instruções, consulte Registre um endpoint personalizado.

Parâmetros macro	Description	Como obter essas informações
IOTMI_MAN AGEDINTEG RATIONS_E NDPOINT_PORT	O número da porta para o endpoint de integraçõ es gerenciadas	Por padrão, a porta 8883 é usada para operações de publicação e assinatura do MQTT. A porta 443 está configurada para a extensão TLS de negociação de protocolo de camada de aplicativo (ALPN) que os dispositivos usam.

5. Crie e execute os aplicativos de demonstração

   Esta seção demonstra dois aplicativos de demonstração do Linux: uma câmera de segurança simples e um purificador de ar, ambos usados CMake como sistema de construção.

   a. Aplicação simples de câmera de segurança

      Para criar e executar o aplicativo, execute estes comandos:

```
>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
```

      Esta demonstração implementa funções C de baixo nível para uma câmera simulada com controlador de sessão RTC e clusters de gravação. Conclua o fluxo mencionado em Fluxo de trabalho do provisionado antes de executar.

      Exemplo de saída do aplicativo de demonstração:

```
[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. Aplicação simples de purificador de ar

Para criar e executar o aplicativo, execute os seguintes comandos:

```
>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
```

Esta demonstração implementa funções C de baixo nível para um purificador de ar simulado com 2 terminais e os seguintes clusters compatíveis:

Clusters compatíveis para terminais de purificadores de ar

Endpoint	Clusters
Ponto final #1: Purificador de ar	OnOff

Endpoint	Clusters
	Controle do ventilador
	Monitoramento do filtro HEPA
	Monitoramento de filtro de carbono ativado
Ponto final #2: Sensor de qualidade do ar	Qualidade do ar
	Medição da concentração de dióxido de carbono
	Medição da concentração de formaldeído
	Medição da concentração de Pm25
	Medição da concentração de Pm1
	Medição da concentração total de compostos orgânicos voláteis

A saída é semelhante à do aplicativo de demonstração da câmera, com diferentes clusters suportados.

6. Próximas etapas:

As integrações gerenciadas, o SDK do dispositivo final e os aplicativos de demonstração agora estão em execução em suas instâncias da Amazon EC2 . Isso permite que você desenvolva e teste seus aplicativos em seu próprio hardware físico. Com essa configuração, você pode aproveitar o serviço de integrações gerenciadas para controlar seus AWS IoT dispositivos.

a. Desenvolva funções de retorno de chamada de hardware

Antes de implementar as funções de retorno de chamada do hardware, entenda como a API funciona. Este exemplo usa o On/Off cluster e o OnOff atributo para controlar a função de um dispositivo. Para obter detalhes da API, consulteFunção C de baixo nível 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. Configure endpoints e conecte funções de retorno de chamada de hardware

Depois de implementar as funções, crie endpoints e registre seus retornos de chamada. Complete as seguintes tarefas:

i. Crie um agente de dispositivo.

A. Crie um agente de dispositivo usando `iotmiDev_Agent_new()` antes de invocar qualquer outra função do SDK.

B. No mínimo, sua configuração deve incluir os parâmetros thingID e clientID.

C. Use a `iotmiDev_Agent_initDefaultConfig()` função para definir valores padrão razoáveis para parâmetros como tamanhos de fila e pontos finais máximos.

D. Ao terminar de usar os recursos, libere-os com a `iotmiDev_Agent_free()` função. Isso evita vazamentos de memória e garante o gerenciamento adequado de recursos em seu aplicativo.

ii. Preencha os ponteiros da função de retorno de chamada para cada estrutura de cluster que você deseja oferecer suporte.

iii. Configure endpoints e registre seus clusters compatíveis.

Crie endpoints com`iotmiDev_Agent_addEndpoint()`, o que requer:

A. Um ID de endpoint exclusivo.

B. Um nome descritivo de endpoint

C. Um ou mais tipos de dispositivos que estão em conformidade com as definições do modelo AWS de dados.

D. Depois de criar endpoints, registre os clusters usando as funções de registro específicas do cluster apropriadas.

E. Cada registro de cluster exige funções de retorno de chamada para atributos e comandos. O sistema passa o ponteiro de contexto do usuário para os retornos de chamada para manter o estado entre as chamadas.

```
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. Use o manipulador de trabalhos para obter o documento do trabalho

   i. Inicie uma chamada para seu aplicativo OTA:

```
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. Ligue `iotmi_JobsHandler_start` para inicializar o manipulador de trabalhos.

iii. Ligue `iotmi_JobsHandler_getJobDocument` para recuperar o documento do trabalho das integrações gerenciadas.

iv. Quando o documento de tarefas for obtido com sucesso, escreva sua operação OTA personalizada na `processOTA` função e retorne um `JobSucceeded` status.

```
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." ) );

}
```

Porte o SDK do dispositivo final para o seu dispositivo

Porte o SDK do dispositivo final para a plataforma do seu dispositivo. Siga estas etapas para conectar seus dispositivos ao Gerenciamento de AWS IoT dispositivos.

Baixe e verifique o SDK do dispositivo final

1. Baixe a versão mais recente do SDK do dispositivo final no console de integrações gerenciadas.
2. Verifique se sua plataforma está na lista de plataformas suportadas emReferência: Plataformas suportadas.

   > **Note**
   >
   > O SDK do dispositivo final foi testado nas plataformas especificadas. Outras plataformas podem funcionar, mas não foram testadas.

3. Extraia (descompacte) os arquivos do SDK no seu espaço de trabalho.
4. Configure seu ambiente de compilação com as seguintes configurações:
   - Caminhos do arquivo de origem
   - Diretórios de arquivos de cabeçalho
   - Bibliotecas necessárias
   - Sinalizadores de compilador e vinculador
5. Antes de portar a camada de abstração da plataforma (PAL), certifique-se de que as funcionalidades básicas da sua plataforma estejam inicializadas. As funcionalidades incluem:

- Tarefas do sistema operacional
- Periféricos
- Interfaces de rede
- Requisitos específicos da plataforma

Porte o PAL para o seu dispositivo

1. Crie um novo diretório para suas implementações específicas da plataforma no diretório da plataforma existente. Por exemplo, se você usa Freertos, crie um diretório em. `platform/freertos`

   Example Estrutura de diretórios do SDK

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

2. Copie os arquivos de implementação de referência POSIX (.c e .h) da pasta posix para o novo diretório da plataforma. Esses arquivos fornecem um modelo para as funções que você precisará implementar.

   - Gerenciamento de memória flash para armazenamento de credenciais
   - Implementação do PKCS #11
   - Interface de transporte de rede
   - Sincronização de horário
   - Funções de reinicialização e redefinição do sistema

- Mecanismos de registro em log
- Configurações específicas do dispositivo

3. Configure a autenticação Transport Layer Security (TLS) com MBed TLS.

   - Use a implementação do POSIX fornecida se você já tiver uma versão do MBed TLS que corresponda à versão do SDK na sua plataforma.
   - Com uma versão diferente do TLS, você implementa os ganchos de transporte para sua pilha TLS com pilha. TCP/IP

4. Compare a configuração mBedTLS da sua plataforma com os requisitos do SDK em. `platform/posix/mbedtls/mbedtls_config.h` Certifique-se de que todas as opções necessárias estejam ativadas.

5. O SDK depende do CoreMQTT para interagir com a nuvem. Portanto, você deve implementar uma camada de transporte de rede que use a seguinte estrutura:

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

   Para obter mais informações, consulte a documentação da interface de transporte no site do FreeRTOS.

6. (Opcional) O SDK usa a API PCKS #11 para lidar com operações de certificado. O CorePKCS é uma implementação do PKCS #11 não específica de hardware para prototipagem. Recomendamos que você use criptoprocessadores seguros, como Trusted Platform Module (TPM), Hardware Security Module (HSM) ou Secure Element em seu ambiente de produção:

   - Analise o exemplo de implementação do PKCS #11 que usa o sistema de arquivos Linux para gerenciamento de credenciais em. `platform/posix/corePKCS11-mbedtls`
   - Implemente a camada PAL PKCS #11 em`commonDependencies/core_pkcs11/corePKCS11/source/include/core_pkcs11.h`.
   - Implemente o sistema de arquivos Linux em`platform/posix/corePKCS11-mbedtls/source/iotmi_pal_Pkcs11Operations.c`.

- Implemente a função de armazenamento e carregamento do seu tipo de armazenamento `emplatform/include/iotmi_pal_Nvm.h`.
- Implemente o acesso padrão ao arquivo `emplatform/posix/source/iotmi_pal_Nvm.c`.

Para obter instruções detalhadas de portabilidade, consulte Como portar a PKCS11 biblioteca principal no Guia do Usuário do FreeRTOS.

7. Adicione as bibliotecas estáticas do SDK ao seu ambiente de compilação:

   - Configure os caminhos da biblioteca para resolver quaisquer problemas de vinculadores ou conflitos de símbolos
   - Verifique se todas as dependências estão vinculadas corretamente

Teste sua porta

Você pode usar o aplicativo de exemplo existente para testar sua porta. A compilação deve ser concluída sem erros ou avisos.

> **Note**
>
> Recomendamos que você comece com o aplicativo multitarefa mais simples possível. O aplicativo de exemplo fornece um equivalente multitarefa.

1. Encontre o aplicativo de exemplo em `examples/[device_type_sample]`.
2. Converta o `main.c` arquivo em seu projeto e adicione uma entrada para chamar a função main () existente.
3. Verifique se você pode compilar o aplicativo de demonstração com sucesso.

Referência técnica

Tópicos

- Referência: Plataformas suportadas
- Referência: Requisitos técnicos
- Referência: API comum

Referência: Plataformas suportadas

A tabela a seguir mostra as plataformas compatíveis com o SDK.

Plataformas compatíveis

Plataforma	Arquitetura	Sistema operacional
Linux x86_64	x86_64	Linux
Ambarella	Armv (8) AArch64	Linux
Amebad	Armv8-M de 32 bits	FreeRTOS
ESP32S3	Xtensa LX7 de 32 bits	FreeRTOS

Referência: Requisitos técnicos

A tabela a seguir mostra os requisitos técnicos do SDK, incluindo o espaço de RAM. O SDK do dispositivo final em si requer cerca de 5 a 10 MB de espaço ROM ao usar a mesma configuração.

Espaço RAM

SDK e componentes	Requisitos de espaço (bytes usados)
O próprio SDK do dispositivo final	180 KB
Fila de comandos padrão do agente MQTT	480 bytes (pode ser configurado)
Fila de entrada padrão do MQTT Agent	320 bytes (pode ser configurado)

Referência: API comum

Esta seção é uma lista de operações de API que não são específicas de um cluster.

```
/* 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);
```

Segurança em integrações gerenciadas para AWS IoT Device Management

A segurança na nuvem AWS é a maior prioridade. Como AWS cliente, você se beneficia de data centers e arquiteturas de rede criados para atender aos requisitos das organizações mais sensíveis à segurança.

A segurança é uma responsabilidade compartilhada entre você AWS e você. O Modelo de Responsabilidade Compartilhada descreve isso como segurança da nuvem e segurança na nuvem:

- Segurança da nuvem — AWS é responsável por proteger a infraestrutura que executa AWS os serviços no Nuvem AWS. AWS também fornece serviços que você pode usar com segurança. Auditores terceirizados testam e verificam regularmente a eficácia de nossa segurança como parte dos Programas de Conformidade Programas de AWS de . Para saber mais sobre os programas de conformidade que se aplicam às integrações gerenciadas, consulte AWS Serviços no escopo do programa de conformidade AWS .
- Segurança na nuvem — Sua responsabilidade é determinada pelo AWS serviço que você usa. Você também é responsável por outros fatores, incluindo a confidencialidade de seus dados, os requisitos da empresa e as leis e regulamentos aplicáveis.

Essa documentação ajuda você a entender como aplicar o modelo de responsabilidade compartilhada ao usar integrações gerenciadas. Os tópicos a seguir mostram como configurar integrações gerenciadas para atender aos seus objetivos de segurança e conformidade. Você também aprenderá a usar outros AWS serviços que ajudam a monitorar e proteger seus recursos de integrações gerenciadas.

Tópicos

Proteção de dados em integrações gerenciadas

O modelo de responsabilidade AWS compartilhada modelo de de se aplica à proteção de dados em integrações gerenciadas para AWS IoT Device Management. Conforme descrito neste modelo, AWS é responsável por proteger a infraestrutura global que executa todos os Nuvem AWS. Você é responsável por manter o controle sobre o conteúdo hospedado nessa infraestrutura. Você também é responsável pelas tarefas de configuração e gerenciamento de segurança dos Serviços da AWS que usa. Para obter mais informações sobre a privacidade de dados, consulte as Data Privacy FAQ. Para obter mais informações sobre a proteção de dados na Europa, consulte a postagem do blog AWS Shared Responsibility Model and RGPD no Blog de segurança da AWS .

Para fins de proteção de dados, recomendamos que você proteja Conta da AWS as credenciais e configure usuários individuais com Centro de Identidade do AWS IAM ou AWS Identity and Access Management (IAM). Dessa maneira, cada usuário receberá apenas as permissões necessárias para cumprir suas obrigações de trabalho. Recomendamos também que você proteja seus dados das seguintes formas:

- Use uma autenticação multifator (MFA) com cada conta.
- Use SSL/TLS para se comunicar com AWS os recursos. Exigimos TLS 1.2 e recomendamos TLS 1.3.
- Configure a API e o registro de atividades do usuário com AWS CloudTrail. Para obter informações sobre o uso de CloudTrail trilhas para capturar AWS atividades, consulte Como trabalhar com CloudTrail trilhas no Guia AWS CloudTrail do usuário.
- Use soluções de AWS criptografia, juntamente com todos os controles de segurança padrão Serviços da AWS.
- Use serviços gerenciados de segurança avançada, como o Amazon Macie, que ajuda a localizar e proteger dados sensíveis armazenados no Amazon S3.
- Se você precisar de módulos criptográficos validados pelo FIPS 140-3 ao acessar AWS por meio de uma interface de linha de comando ou de uma API, use um endpoint FIPS. Para obter mais informações sobre os endpoints FIPS disponíveis, consulte Federal Information Processing Standard (FIPS) 140-3.

É altamente recomendável que nunca sejam colocadas informações confidenciais ou sigilosas, como endereços de e-mail de clientes, em tags ou campos de formato livre, como um campo Nome. Isso inclui quando você trabalha com integrações gerenciadas para AWS IoT Device Management ou outros Serviços da AWS usando o console, a API ou AWS SDKs. AWS CLI Quaisquer dados

inseridos em tags ou em campos de texto de formato livre usados para nomes podem ser usados para logs de faturamento ou de diagnóstico. Se você fornecer um URL para um servidor externo, é fortemente recomendável que não sejam incluídas informações de credenciais no URL para validar a solicitação nesse servidor.

Criptografia de dados em repouso para integrações gerenciadas

Integrações gerenciadas para AWS IoT Device Management criptografar dados confidenciais de clientes em repouso, por padrão, usando chaves de criptografia.

Há dois tipos de chaves de criptografia que são usadas para proteger dados confidenciais para clientes de integrações gerenciadas:

Chaves gerenciadas pelo cliente (CMK)

As integrações gerenciadas oferecem suporte ao uso de chaves simétricas gerenciadas pelo cliente que você pode criar, possuir e gerenciar. Você tem controle total sobre essas chaves do KMS, incluindo estabelecer e manter suas políticas de chaves, políticas do IAM e concessões, além de habilitá-las e desabilitá-las, alternar seu material criptográfico, adicionar etiquetas, criar aliases que fazem referência às chaves do KMS e programar a exclusão de chaves do KMS.

AWS chaves de propriedade

As integrações gerenciadas usam essas chaves por padrão para criptografar automaticamente os dados confidenciais do cliente. Você não pode visualizar, gerenciar ou auditar seu uso. Você não precisa realizar nenhuma ação ou alterar nenhum programa para proteger as chaves que criptografam seus dados. Por padrão, a criptografia de dados em repouso ajuda a reduzir a sobrecarga operacional e a complexidade envolvidas na proteção de dados confidenciais. Ao mesmo tempo, ela permite que você crie aplicações seguras que atendam aos rigorosos requisitos regulatórios e de conformidade de criptografia.

A chave de criptografia padrão usada é a AWS chave própria. Como alternativa, a API opcional para atualizar sua chave de criptografia é `PutDefaultEncryptionConfiguration`.

Para obter mais informações sobre os tipos de chaves de AWS KMS criptografia, consulte AWS KMS chaves.

AWS KMS uso para integrações gerenciadas

As integrações gerenciadas criptografam e descriptografam todos os dados do cliente usando criptografia de envelope. Esse tipo de criptografia pegará seus dados de texto simples e os

criptografará com uma chave de dados. Em seguida, uma chave de criptografia chamada chave de empacotamento criptografará a chave de dados original usada para criptografar seus dados de texto sem formatação. Na criptografia de envelope, chaves de empacotamento adicionais podem ser usadas para criptografar chaves de empacotamento existentes que estejam mais próximas em graus de separação da chave de dados original. Como a chave de dados original é criptografada por uma chave de empacotamento armazenada separadamente, você pode armazenar a chave de dados original e os dados criptografados em texto simples no mesmo local. Um chaveiro é usado para gerar, criptografar e descriptografar chaves de dados, além da chave de empacotamento usada para criptografar e descriptografar a chave de dados.

> **Note**
>
> O SDK AWS de criptografia de banco de dados fornece criptografia de envelope para sua implementação de criptografia no lado do cliente. Para obter mais informações sobre o SDK AWS de criptografia de banco de dados, consulte O que é o SDK AWS de criptografia de banco de dados?

Para obter mais informações sobre criptografia de envelope, chaves de dados, chaves de embalagem e chaveiros, consulte Criptografia de envelope, chave de dados, chave de embalagem e chaveiros.

As integrações gerenciadas exigem que os serviços usem sua chave gerenciada pelo cliente para as seguintes operações internas:

- Envie `DescribeKey` solicitações AWS KMS para verificar se o ID simétrico da chave gerenciada pelo cliente foi fornecido ao fazer a rotação das chaves de dados.
- Envie `GenerateDataKeyWithoutPlaintext` solicitações AWS KMS para gerar chaves de dados criptografadas pela chave gerenciada pelo cliente.
- Envie `ReEncrypt*` solicitações para AWS KMS recriptografar as chaves de dados pela chave gerenciada pelo cliente.
- Envie `Decrypt` solicitações para AWS KMS decifrar dados usando sua chave gerenciada pelo cliente.

Tipos de dados criptografados usando chaves de criptografia

As integrações gerenciadas usam chaves de criptografia para criptografar vários tipos de dados armazenados em repouso. A lista a seguir descreve os tipos de dados criptografados em repouso usando chaves de criptografia:

- Eventos do conector de nuvem para nuvem (C2C), como descoberta de dispositivos e atualização de status de dispositivos.
- Criação de uma coisa gerenciada que representa o dispositivo físico e um perfil de dispositivo contendo os recursos de um tipo específico de dispositivo. Para obter mais informações sobre um dispositivo e um perfil de dispositivo, consulte Dispositivo Dispositivo e.
- Notificações de integrações gerenciadas sobre vários aspectos da implementação do seu dispositivo. Para obter mais informações sobre notificações de integrações gerenciadas, consulteConfigurar notificações de integrações gerenciadas.
- Informações de identificação pessoal (PII) de um usuário final, como material de autenticação do dispositivo, número de série do dispositivo, nome do usuário final, identificador do dispositivo e Amazon Resource Name (arn) do dispositivo.

Como as integrações gerenciadas usam as principais políticas em AWS KMS

Para rotação de chaves de filiais e chamadas assíncronas, as integrações gerenciadas exigem uma política de chaves para usar sua chave de criptografia. Uma política chave é usada pelos seguintes motivos:

- Autorize programaticamente o uso de uma chave de criptografia para outros diretores. AWS

Para obter um exemplo de uma política de chaves usada para gerenciar o acesso à sua chave de criptografia em integrações gerenciadas, consulte Crie uma chave de criptografia

> **Note**
>
> Para uma chave AWS própria, não é necessária uma política de chaves, pois a chave AWS própria é de propriedade AWS e você não pode visualizá-la, gerenciá-la ou usá-la. As integrações gerenciadas usam a AWS chave própria por padrão para criptografar automaticamente os dados confidenciais de seus clientes.

Além de usar políticas de chaves para gerenciar sua configuração de criptografia com AWS KMS chaves, as integrações gerenciadas usam políticas do IAM. Para obter mais informações sobre as políticas do IAM, consulte Políticas e permissões em AWS Identity and Access Management.

Crie uma chave de criptografia

Você pode criar uma chave de criptografia usando o Console de gerenciamento da AWS ou AWS KMS APIs o.

Para criar uma chave de criptografia

Siga as etapas para criar uma chave KMS no Guia do AWS Key Management Service desenvolvedor.

Política de chave

Uma declaração de política fundamental controla o acesso a uma AWS KMS chave. Cada AWS KMS chave conterá somente uma política de chaves. Essa política de chaves determina quais AWS diretores podem usar a chave e como eles podem usá-la. Para obter mais informações sobre como gerenciar o acesso e o uso de AWS KMS chaves usando declarações de política de chaves, consulte Gerenciando o acesso usando políticas.

Veja a seguir um exemplo de uma declaração de política fundamental que você pode usar para gerenciar o acesso e o uso das AWS KMS chaves armazenadas em suas Conta da AWS integrações gerenciadas:

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

Para obter mais informações sobre armazenamentos de chaves, consulte Armazenamentos de chaves.

Atualizando a configuração de criptografia

A capacidade de atualizar perfeitamente sua configuração de criptografia é fundamental para gerenciar sua implementação de criptografia de dados para integrações gerenciadas. Ao iniciar a integração com integrações gerenciadas, você será solicitado a selecionar sua configuração de criptografia. Suas opções serão as chaves de AWS propriedade padrão ou criarão sua própria AWS KMS chave.

Console de gerenciamento da AWS

Para atualizar sua configuração de criptografia no Console de gerenciamento da AWS, abra a página inicial do AWS IoT serviço e navegue até Integração gerenciada para controle unificado > Configurações > Criptografia. Na janela Configurações de criptografia, você pode atualizar sua configuração de criptografia selecionando uma nova AWS KMS chave para proteção adicional de criptografia. Escolha Personalizar configurações de criptografia (avançadas) para selecionar uma AWS KMS chave existente ou você pode escolher Criar uma AWS KMS chave para criar sua própria chave gerenciada pelo cliente.

Comandos da API

Há dois APIs usados para gerenciar sua configuração de criptografia de AWS KMS chaves em integrações gerenciadas: `PutDefaultEncryptionConfiuration` e. `GetDefaultEncryptionConfiguration`

Para atualizar a configuração de criptografia padrão, ligue para`PutDefaultEncryptionConfiuration`. Para obter mais informações sobre `PutDefaultEncryptionConfiuration`, consulte PutDefaultEncryptionConfiuration.

Para ver a configuração de criptografia padrão, ligue para`GetDefaultEncryptionConfiguration`. Para obter mais informações sobre `GetDefaultEncryptionConfiguration`, consulte GetDefaultEncryptionConfiguration.

Gerenciamento de identidade e acesso para integrações gerenciadas

AWS Identity and Access Management (IAM) é uma ferramenta AWS service (Serviço da AWS) que ajuda o administrador a controlar com segurança o acesso aos AWS recursos. Os administradores do IAM controlam quem pode ser autenticado (conectado) e autorizado (tem permissões) para usar

recursos de integrações gerenciadas. O IAM é um AWS service (Serviço da AWS) que você pode usar sem custo adicional.

Tópicos

Público

A forma como você usa AWS Identity and Access Management (IAM) difere com base na sua função:

- Usuário do serviço: solicite permissões ao administrador caso você não consiga acessar os recursos (consulte Solução de problemas de identidade e acesso de integrações gerenciadas).
- Administrador do serviço: determine o acesso do usuário e envie solicitações de permissão (consulte Como as integrações gerenciadas funcionam com o IAM).
- Administrador do IAM: escreva políticas para gerenciar o acesso (consulte Exemplos de políticas baseadas em identidade para integrações gerenciadas).

Autenticação com identidades

A autenticação é a forma como você faz login AWS usando suas credenciais de identidade. Você deve estar autenticado como usuário do IAM ou assumindo uma função do IAM. Usuário raiz da conta da AWS

Você pode fazer login como uma identidade federada usando credenciais de uma fonte de identidade como Centro de Identidade do AWS IAM (IAM Identity Center), autenticação de login único ou credenciais. Google/Facebook Para ter mais informações sobre como fazer login, consulte Como fazer login em sua Conta da AWS no Guia do usuário do Início de Sessão da AWS .

Para acesso programático, AWS fornece um SDK e uma CLI para assinar solicitações criptograficamente. Para ter mais informações, consulte AWS Signature Version 4 para solicitações de API no Guia do usuário do IAM.

Conta da AWS usuário root

Ao criar um Conta da AWS, você começa com uma identidade de login chamada usuário Conta da AWS raiz que tem acesso completo a todos Serviços da AWS os recursos. É altamente recomendável não usar o usuário-raiz em tarefas diárias. Para ver as tarefas que exigem credenciais de usuário-raiz, consulte Tarefas que exigem credenciais de usuário-raiz no Guia do usuário do IAM.

Identidade federada

Como prática recomendada, exija que os usuários humanos usem a federação com um provedor de identidade para acessar Serviços da AWS usando credenciais temporárias.

Uma identidade federada é um usuário do seu diretório corporativo, provedor de identidade da web ou Directory Service que acessa Serviços da AWS usando credenciais de uma fonte de identidade. As identidades federadas assumem perfis que oferecem credenciais temporárias.

Para o gerenciamento de acesso centralizado, recomendamos Centro de Identidade do AWS IAM. Para obter mais informações, consulte O que é o IAM Identity Center? no Guia do usuário do Centro de Identidade do AWS IAM .

Usuários e grupos do IAM

Usuário do IAM é uma identidade com permissões específicas para uma única pessoa ou aplicação. É recomendável usar credenciais temporárias, em vez de usuários do IAM com credenciais de longo prazo. Para obter mais informações, consulte Exigir que usuários humanos usem a federação com um provedor de identidade para acessar AWS usando credenciais temporárias no Guia do usuário do IAM.

Um grupo do IAM especifica um conjunto de usuários do IAM e facilita o gerenciamento de permissões para grandes conjuntos de usuários. Para ter mais informações, consulte Casos de uso para usuários do IAM no Guia do usuário do IAM.

Perfis do IAM

Um perfil do IAM é uma identidade com permissões específicas que oferece credenciais temporárias. Você pode assumir uma função mudando de um usuário para uma função do IAM (console) ou

chamando uma operação de AWS API AWS CLI ou. Para obter mais informações, consulte Métodos para assumir um perfil no Manual do usuário do IAM.

As funções do IAM são úteis para acesso de usuários federados, permissões temporárias de usuários do IAM, acesso entre contas, acesso entre serviços e aplicativos executados na Amazon. EC2 Consulte mais informações em Acesso a recursos entre contas no IAM no Guia do usuário do IAM.

Gerenciar o acesso usando políticas

Você controla o acesso AWS criando políticas e anexando-as a AWS identidades ou recursos. Uma política define permissões quando associada a uma identidade ou recurso. AWS avalia essas políticas quando um diretor faz uma solicitação. A maioria das políticas é armazenada AWS como documentos JSON. Para ter mais informações sobre documentos de política JSON, consulte Visão geral das políticas de JSON no Guia do usuário do IAM.

Usando políticas, os administradores especificam quem tem acesso ao quê definindo qual entidade principal pode realizar ações em quais recursos e sob quais condições.

Por padrão, usuários e perfis não têm permissões. Um administrador do IAM cria políticas do IAM e as adiciona a funções, que os usuários podem acabar assumindo. As políticas do IAM definem permissões, independentemente do método usado para realizar a operação.

Políticas baseadas em identidade

As políticas baseadas em identidade são documentos de políticas de permissão JSON que você pode anexar a uma identidade (usuário, grupo de usuários ou perfil). Essas políticas controlam quais ações as identidades podem realizar, em quais recursos e sob quais condições. Para saber como criar uma política baseada em identidade, consulte Definir permissões personalizadas do IAM com as políticas gerenciadas pelo cliente no Guia do Usuário do IAM.

As políticas baseadas em identidade podem ser políticas em linha (incorporadas diretamente em uma única identidade) ou políticas gerenciadas (políticas autônomas anexadas a várias identidades). Para saber como escolher entre uma política gerenciada e políticas em linha, consulte Escolher entre políticas gerenciadas e políticas em linha no Guia do usuário do IAM.

Políticas baseadas em recursos

Políticas baseadas em recursos são documentos de políticas JSON que você anexa a um recurso. Os exemplos incluem políticas de confiança de perfil do IAM e políticas de bucket do Amazon S3.

Em serviços compatíveis com políticas baseadas em recursos, os administradores de serviço podem usá-las para controlar o acesso a um recurso específico. É necessário especificar uma entidade principal em uma política baseada em recursos.

Políticas baseadas em recursos são políticas em linha localizadas nesse serviço. Você não pode usar políticas AWS gerenciadas do IAM em uma política baseada em recursos.

Outros tipos de política

AWS oferece suporte a tipos de políticas adicionais que podem definir o máximo de permissões concedidas por tipos de políticas mais comuns:

- Limites de permissões: definem o número máximo de permissões que uma política baseada em identidade pode conceder a uma entidade do IAM. Para obter mais informações, consulte Limites de permissões para entidades do IAM no Guia do usuário do IAM.
- Políticas de controle de serviço (SCPs) — Especifique as permissões máximas para uma organização ou unidade organizacional em AWS Organizations. Para obter mais informações, consulte Políticas de controle de serviço no Guia do usuário do AWS Organizations .
- Políticas de controle de recursos (RCPs) — Defina o máximo de permissões disponíveis para recursos em suas contas. Para obter mais informações, consulte Políticas de controle de recursos (RCPs) no Guia AWS Organizations do usuário.
- Políticas de sessão: políticas avançadas passadas como um parâmetro durante a criação de uma sessão temporária para uma função ou um usuário federado. Para obter mais informações, consulte Políticas de sessão no Guia do usuário do IAM.

Vários tipos de política

Quando vários tipos de política são aplicáveis a uma solicitação, é mais complicado compreender as permissões resultantes. Para saber como AWS determinar se uma solicitação deve ser permitida quando vários tipos de políticas estão envolvidos, consulte Lógica de avaliação de políticas no Guia do usuário do IAM.

AWS políticas gerenciadas para integrações gerenciadas

Para adicionar permissões a usuários, grupos e funções, é mais fácil usar políticas AWS gerenciadas do que escrever políticas você mesmo. É necessário tempo e experiência para criar políticas

gerenciadas pelo cliente do IAM que fornecem à sua equipe apenas as permissões de que precisam. Para começar rapidamente, você pode usar nossas políticas AWS gerenciadas. Essas políticas abrangem casos de uso comuns e estão disponíveis na sua Conta da AWS. Para obter mais informações sobre políticas AWS gerenciadas, consulte políticas AWS gerenciadas no Guia do usuário do IAM.

AWS os serviços mantêm e atualizam as políticas AWS gerenciadas. Você não pode alterar as permissões nas políticas AWS gerenciadas. Os serviços ocasionalmente acrescentam permissões adicionais a uma política gerenciada pela AWS para oferecer suporte a novos recursos. Esse tipo de atualização afeta todas as identidades (usuários, grupos e funções) em que a política está anexada. É mais provável que os serviços atualizem uma política gerenciada pela AWS quando um novo recurso for iniciado ou novas operações se tornarem disponíveis. Os serviços não removem as permissões de uma política AWS gerenciada, portanto, as atualizações de políticas não violarão suas permissões existentes.

Além disso, AWS oferece suporte a políticas gerenciadas para funções de trabalho que abrangem vários serviços. Por exemplo, a política ReadOnlyAccess AWS gerenciada fornece acesso somente de leitura a todos os AWS serviços e recursos. Quando um serviço lança um novo recurso, AWS adiciona permissões somente de leitura para novas operações e recursos. Para obter uma lista e descrições das políticas de perfis de trabalho, consulte Políticas gerenciadas pela AWS para perfis de trabalho no Guia do usuário do IAM.

AWS política gerenciada: AWSIo TManaged IntegrationsFullAccess

É possível anexar a política `AWSIoTManagedIntegrationsFullAccess` às suas identidades do IAM.

Essa política concede permissões de acesso total às integrações gerenciadas e serviços relacionados. Para ver essa política no Console de gerenciamento da AWS, consulte AWSIoTManagedIntegrationsFullAccess.

Detalhes de permissões

Esta política inclui as seguintes permissões:

- `iotmanagedintegrations`— Fornece acesso total a integrações gerenciadas e serviços relacionados para os usuários, grupos e funções do IAM aos quais você adiciona essa política.

- `iam`— Permite que os usuários, grupos e funções do IAM atribuídos criem uma função vinculada ao serviço em um. Conta da AWS

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 política gerenciada: AWS Io TManaged IntegrationsRolePolicy

É possível anexar a política `AWS IoTManagedIntegrationsRolePolicy` às suas identidades do IAM.

Essa política concede às integrações gerenciadas permissão para publicar CloudWatch registros e métricas da Amazon em seu nome.

Para ver essa política no Console de gerenciamento da AWS, consulte AWSIoTManagedIntegrationsRolePolicy.

Detalhes das permissões

Esta política inclui as seguintes permissões.

- `logs`— Oferece a capacidade de criar grupos de CloudWatch registros da Amazon e transmitir registros para os grupos.
- `cloudwatch`— Oferece a capacidade de publicar CloudWatch métricas da Amazon. Para obter mais informações sobre CloudWatch as métricas da Amazon, consulte Métricas na 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"
          ]
        }
      }
    }
  ]
}
```

Integrações gerenciadas: atualizações das políticas AWS gerenciadas

Veja detalhes sobre as atualizações das políticas AWS gerenciadas para integrações gerenciadas desde que esse serviço começou a rastrear essas alterações. Para receber alertas automáticos sobre alterações nessa página, assine o feed RSS na página de histórico de documentos de integrações gerenciadas.

Alteração	Descrição	Data
As integrações gerenciadas começaram a monitorar as mudanças	As integrações gerenciadas começaram a monitorar as mudanças em suas políticas AWS gerenciadas.	03 de março de 2025

Como as integrações gerenciadas funcionam com o IAM

Antes de usar o IAM para gerenciar o acesso às integrações gerenciadas, saiba quais recursos do IAM estão disponíveis para uso com integrações gerenciadas.

Recursos do IAM que você pode usar com integrações gerenciadas

Recurso do IAM	Suporte a integrações gerenciadas
Políticas baseadas em identidade	Sim
Políticas baseadas em recurso	Não
Ações de políticas	Sim
Recursos de políticas	Sim
Chaves de condição de políticas	Sim
ACLs	Não
ABAC (tags em políticas)	Não
Credenciais temporárias	Sim
Permissões de entidade principal	Sim
Perfis de serviço	Sim
Perfis vinculados a serviço	Sim

Para ter uma visão de alto nível de como as integrações gerenciadas e outros AWS serviços funcionam com a maioria dos recursos do IAM, consulte AWS os serviços que funcionam com o IAM no Guia do usuário do IAM.

Políticas baseadas em identidade para integrações gerenciadas

Compatível com políticas baseadas em identidade: sim

As políticas baseadas em identidade são documentos de políticas de permissões JSON que podem ser anexados a uma identidade, como usuário do IAM, grupo de usuários ou perfil. Essas

políticas controlam quais ações os usuários e perfis podem realizar, em quais recursos e em que condições. Para saber como criar uma política baseada em identidade, consulte Definir permissões personalizadas do IAM com as políticas gerenciadas pelo cliente no Guia do Usuário do IAM.

Com as políticas baseadas em identidade do IAM, é possível especificar ações e recursos permitidos ou negados, assim como as condições sob as quais as ações são permitidas ou negadas. Para saber mais sobre todos os elementos que podem ser usados em uma política JSON, consulte Referência de elemento de política JSON do IAM no Guia do usuário do IAM.

Exemplos de políticas baseadas em identidade para integrações gerenciadas

Para ver exemplos de políticas baseadas em identidade de integrações gerenciadas, consulte. Exemplos de políticas baseadas em identidade para integrações gerenciadas

Políticas baseadas em recursos em integrações gerenciadas

Compatibilidade com políticas baseadas em recursos: não

Políticas baseadas em recursos são documentos de políticas JSON que você anexa a um recurso. São exemplos de políticas baseadas em recursos as políticas de confiança de perfil do IAM e as políticas de bucket do Amazon S3. Em serviços compatíveis com políticas baseadas em recursos, os administradores de serviço podem usá-las para controlar o acesso a um recurso específico. Para o atributo ao qual a política está anexada, a política define quais ações uma entidade principal especificado pode executar nesse atributo e em que condições. É necessário especificar uma entidade principal em uma política baseada em recursos. Os diretores podem incluir contas, usuários, funções, usuários federados ou. Serviços da AWS

Para permitir o acesso entre contas, é possível especificar uma conta inteira ou as entidades do IAM em outra conta como a entidade principal em uma política baseada em recursos. Consulte mais informações em Acesso a recursos entre contas no IAM no Guia do usuário do IAM.

Ações políticas para integrações gerenciadas

Compatível com ações de políticas: sim

Os administradores podem usar políticas AWS JSON para especificar quem tem acesso ao quê. Ou seja, qual entidade principal pode executar ações em quais recursos e em que condições.

O elemento `Action` de uma política JSON descreve as ações que podem ser usadas para permitir ou negar acesso em uma política. Incluem ações em uma política para conceder permissões para executar a operação associada.

Para ver uma lista de ações de integrações gerenciadas, consulte Ações definidas por integrações gerenciadas na Referência de Autorização de Serviço.

As ações de política em integrações gerenciadas usam o seguinte prefixo antes da ação:

```
iot-mi
```

Para especificar várias ações em uma única declaração, separe-as com vírgulas.

```
"Action": [
      "iot-mi:action1",
      "iot-mi:action2"
         ]
```

Para ver exemplos de políticas baseadas em identidade de integrações gerenciadas, consulte. Exemplos de políticas baseadas em identidade para integrações gerenciadas

Recursos de políticas para integrações gerenciadas

Compatível com recursos de políticas: sim

Os administradores podem usar políticas AWS JSON para especificar quem tem acesso ao quê. Ou seja, qual entidade principal pode executar ações em quais recursos e em que condições.

O elemento de política JSON `Resource` especifica o objeto ou os objetos aos quais a ação se aplica. Como prática recomendada, especifique um recurso usando seu nome do recurso da Amazon (ARN). Para ações que não oferecem compatibilidade com permissões em nível de recurso, use um curinga (*) para indicar que a instrução se aplica a todos os recursos.

```
"Resource": "*"
```

Para ver uma lista dos tipos de recursos de integrações gerenciadas e seus ARNs, consulte Recursos definidos por integrações gerenciadas na Referência de Autorização de Serviço. Para saber com quais ações você pode especificar o ARN de cada recurso, consulte Ações definidas por integrações gerenciadas.

Para ver exemplos de políticas baseadas em identidade de integrações gerenciadas, consulte. Exemplos de políticas baseadas em identidade para integrações gerenciadas

Chaves de condição de política para integrações gerenciadas

Compatível com chaves de condição de política específicas de serviço: sim

Os administradores podem usar políticas AWS JSON para especificar quem tem acesso ao quê. Ou seja, qual entidade principal pode executar ações em quais recursos e em que condições.

O elemento `Condition` especifica quando as instruções são executadas com base em critérios definidos. É possível criar expressões condicionais que usem agentes de condição, como “igual a” ou “menor que”, para fazer a condição da política corresponder aos valores na solicitação. Para ver todas as chaves de condição AWS globais, consulte as chaves de contexto de condição AWS global no Guia do usuário do IAM.

Para ver uma lista de chaves de condição de integrações gerenciadas, consulte Chaves de condição para integrações gerenciadas na Referência de autorização de serviço. Para saber com quais ações e recursos você pode usar uma chave de condição, consulte Ações definidas por integrações gerenciadas.

Para ver exemplos de políticas baseadas em identidade de integrações gerenciadas, consulte. Exemplos de políticas baseadas em identidade para integrações gerenciadas

ACLs em integrações gerenciadas

Suportes ACLs: Não

As listas de controle de acesso (ACLs) controlam quais diretores (membros da conta, usuários ou funções) têm permissões para acessar um recurso. ACLs são semelhantes às políticas baseadas em recursos, embora não usem o formato de documento de política JSON.

ABAC com integrações gerenciadas

Compatível com ABAC (tags em políticas): parcial

O controle de acesso por atributo (ABAC) é uma estratégia de autorização que define permissões com base em atributos chamados de tags. Você pode anexar tags a entidades e AWS recursos do IAM e, em seguida, criar políticas ABAC para permitir operações quando a tag do diretor corresponder à tag no recurso.

Para controlar o acesso baseado em tags, forneça informações sobre as tags no elemento de condição de uma política usando as `aws:ResourceTag/key-name`, `aws:RequestTag/key-name` ou chaves de condição `aws:TagKeys`.

Se um serviço for compatível com as três chaves de condição para cada tipo de recurso, o valor será Sim para o serviço. Se um serviço for compatível com as três chaves de condição somente para alguns tipos de recursos, o valor será Parcial

Para obter mais informações sobre o ABAC, consulte Definir permissões com autorização do ABAC no Guia do usuário do IAM. Para visualizar um tutorial com etapas para configurar o ABAC, consulte Usar controle de acesso baseado em atributos (ABAC) no Guia do usuário do IAM.

Usando credenciais temporárias com integrações gerenciadas

Compatível com credenciais temporárias: sim

As credenciais temporárias fornecem acesso de curto prazo aos AWS recursos e são criadas automaticamente quando você usa a federação ou troca de funções. AWS recomenda que você gere credenciais temporárias dinamicamente em vez de usar chaves de acesso de longo prazo. Para ter mais informações, consulte Credenciais de segurança temporárias no IAM e Serviços da Serviços da AWS que funcionam com o IAM no Guia do usuário do IAM.

Permissões principais entre serviços para integrações gerenciadas

Compatibilidade com o recurso de encaminhamento de sessões de acesso (FAS): sim

As sessões de acesso direto (FAS) usam as permissões do principal chamando um AWS service (Serviço da AWS), combinadas com a solicitação AWS service (Serviço da AWS) de fazer solicitações aos serviços posteriores. Para obter detalhes da política ao fazer solicitações de FAS, consulte Sessões de acesso direto.

Funções de serviço para integrações gerenciadas

Compatível com perfis de serviço: sim

O perfil de serviço é um perfil do IAM que um serviço assume para executar ações em seu nome. Um administrador do IAM pode criar, modificar e excluir um perfil de serviço do IAM. Para obter mais informações, consulte Criar um perfil para delegar permissões a um AWS service (Serviço da AWS) no Guia do Usuário do IAM.

> **Warning**
>
> Alterar as permissões de uma função de serviço pode interromper a funcionalidade de integrações gerenciadas. Edite as funções de serviço somente quando as integrações gerenciadas fornecerem orientação para fazer isso.

Funções vinculadas a serviços para integrações gerenciadas

Compatibilidade com perfis vinculados a serviços: sim

Uma função vinculada ao serviço é um tipo de função de serviço vinculada a um. AWS service (Serviço da AWS) O serviço pode assumir o perfil de executar uma ação em seu nome. As funções vinculadas ao serviço aparecem em você Conta da AWS e são de propriedade do serviço. Um administrador do IAM pode visualizar, mas não editar as permissões para perfis vinculados ao serviço.

Para obter detalhes sobre como criar ou gerenciar perfis vinculados a serviços, consulte Serviços da AWS que funcionam com o IAM. Encontre um serviço na tabela que inclua um `Yes` na coluna Perfil vinculado ao serviço. Escolha o link Sim para visualizar a documentação do perfil vinculado a serviço desse serviço.

Exemplos de políticas baseadas em identidade para integrações gerenciadas

Por padrão, usuários e funções não têm permissão para criar ou modificar recursos de integrações gerenciadas. Para conceder permissão aos usuários para executar ações nos recursos que eles precisam, um administrador do IAM pode criar políticas do IAM.

Para aprender a criar uma política baseada em identidade do IAM ao usar esses documentos de política em JSON de exemplo, consulte Criar políticas do IAM (console) no Guia do usuário do IAM.

Para obter detalhes sobre ações e tipos de recursos definidos por integrações gerenciadas, incluindo o formato do ARNs para cada um dos tipos de recursos, consulte Ações, recursos e chaves de condição para integrações gerenciadas na Referência de Autorização de Serviço.

Tópicos

- Práticas recomendadas de política
- Usando o console de integrações gerenciadas

- Permitir que os usuários visualizem suas próprias permissões

Práticas recomendadas de política

As políticas baseadas em identidade determinam se alguém pode criar, acessar ou excluir recursos de integrações gerenciadas em sua conta. Essas ações podem incorrer em custos para sua Conta da AWS. Ao criar ou editar políticas baseadas em identidade, siga estas diretrizes e recomendações:

- Comece com as políticas AWS gerenciadas e avance para as permissões de privilégios mínimos — Para começar a conceder permissões aos seus usuários e cargas de trabalho, use as políticas AWS gerenciadas que concedem permissões para muitos casos de uso comuns. Eles estão disponíveis no seu Conta da AWS. Recomendamos que você reduza ainda mais as permissões definindo políticas gerenciadas pelo AWS cliente que sejam específicas para seus casos de uso. Para obter mais informações, consulte Políticas gerenciadas pela AWS ou Políticas gerenciadas pela AWS para funções de trabalho no Guia do usuário do IAM.
- Aplique permissões de privilégio mínimo: ao definir permissões com as políticas do IAM, conceda apenas as permissões necessárias para executar uma tarefa. Você faz isso definindo as ações que podem ser executadas em recursos específicos sob condições específicas, também conhecidas como permissões de privilégio mínimo. Para obter mais informações sobre como usar o IAM para aplicar permissões, consulte Políticas e permissões no IAM no Guia do usuário do IAM.
- Use condições nas políticas do IAM para restringir ainda mais o acesso: é possível adicionar uma condição às políticas para limitar o acesso a ações e recursos. Por exemplo, é possível escrever uma condição de política para especificar que todas as solicitações devem ser enviadas usando SSL. Você também pode usar condições para conceder acesso às ações de serviço se elas forem usadas por meio de uma ação específica AWS service (Serviço da AWS), como CloudFormation. Para obter mais informações, consulte Elementos da política JSON do IAM: condição no Guia do usuário do IAM.
- Use o IAM Access Analyzer para validar suas políticas do IAM a fim de garantir permissões seguras e funcionais: o IAM Access Analyzer valida as políticas novas e existentes para que elas sigam a linguagem de política do IAM (JSON) e as práticas recomendadas do IAM. O IAM Access Analyzer oferece mais de cem verificações de política e recomendações práticas para ajudar a criar políticas seguras e funcionais. Para obter mais informações, consulte Validação de políticas do IAM Access Analyzer no Guia do Usuário do IAM.
- Exigir autenticação multifator (MFA) — Se você tiver um cenário que exija usuários do IAM ou um usuário root, ative Conta da AWS a MFA para obter segurança adicional. Para exigir MFA quando as operações de API forem chamadas, adicione condições de MFA às suas políticas. Para obter

mais informações, consulte Configuração de acesso à API protegido por MFA no Guia do Usuário do IAM.

Para obter mais informações sobre as práticas recomendadas do IAM, consulte Práticas recomendadas de segurança no IAM no Guia do usuário do IAM.

Usando o console de integrações gerenciadas

Para acessar o console de integrações gerenciadas, você deve ter um conjunto mínimo de permissões. Essas permissões devem permitir que você liste e visualize detalhes sobre os recursos de integrações gerenciadas em seu Conta da AWS. Caso crie uma política baseada em identidade mais restritiva que as permissões mínimas necessárias, o console não funcionará como pretendido para entidades (usuários ou perfis) com essa política.

Você não precisa permitir permissões mínimas do console para usuários que estão fazendo chamadas somente para a API AWS CLI ou para a AWS API. Em vez disso, permita o acesso somente a ações que correspondam à operação de API que estiverem tentando executar.

Para garantir que usuários e funções ainda possam usar o console de integrações gerenciadas, anexe também as integrações gerenciadas *ConsoleAccess* ou a política *ReadOnly* AWS gerenciada às entidades. Para obter informações, consulte Adicionar permissões a um usuário no Guia do usuário do IAM.

Permitir que os usuários visualizem suas próprias permissões

Este exemplo mostra como criar uma política que permita que os usuários do IAM visualizem as políticas gerenciadas e em linha anexadas a sua identidade de usuário. Essa política inclui permissões para concluir essa ação no console ou programaticamente usando a API AWS CLI ou 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": "*"
        }
    ]
}
```

Solução de problemas de identidade e acesso de integrações gerenciadas

Use as informações a seguir para ajudá-lo a diagnosticar e corrigir problemas comuns que você pode encontrar ao trabalhar com integrações gerenciadas e IAM.

Tópicos

- Não estou autorizado a realizar uma ação em integrações gerenciadas
- Não estou autorizado a realizar iam: PassRole
- Quero permitir que pessoas de fora da minha acessem meus Conta da AWS recursos de integrações gerenciadas

Não estou autorizado a realizar uma ação em integrações gerenciadas

Se você receber uma mensagem de erro informando que não tem autorização para executar uma ação, suas políticas deverão ser atualizadas para permitir que você realize a ação.

O erro do exemplo a seguir ocorre quando o usuário do IAM `mateojackson` tenta usar o console para visualizar detalhes sobre um atributo *`my-example-widget`* fictício, mas não tem as permissões `iot-mi:`*`GetWidget`* fictícias.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform: iot-
mi:GetWidget on resource: my-example-widget
```

Nesse caso, a política do usuário `mateojackson` deve ser atualizada para permitir o acesso ao recurso *`my-example-widget`* usando a ação `iot-mi:`*`GetWidget`*.

Se precisar de ajuda, entre em contato com seu AWS administrador. Seu administrador é a pessoa que forneceu suas credenciais de login.

Não estou autorizado a realizar iam: PassRole

Se você receber um erro informando que não está autorizado a realizar a `iam:PassRole` ação, suas políticas devem ser atualizadas para permitir que você passe uma função para integrações gerenciadas.

Alguns Serviços da AWS permitem que você passe uma função existente para esse serviço em vez de criar uma nova função de serviço ou uma função vinculada ao serviço. Para fazer isso, é preciso ter permissões para passar o perfil para o serviço.

O exemplo de erro a seguir ocorre quando um usuário do IAM chamado `marymajor` tenta usar o console para realizar uma ação em integrações gerenciadas. No entanto, a ação exige que o serviço tenha permissões concedidas por um perfil de serviço. Mary não tem permissões para passar o perfil para o serviço.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

Nesse caso, as políticas de Mary devem ser atualizadas para permitir que ela realize a ação `iam:PassRole`.

Se precisar de ajuda, entre em contato com seu AWS administrador. Seu administrador é a pessoa que forneceu suas credenciais de login.

Quero permitir que pessoas de fora da minha acessem meus Conta da AWS recursos de integrações gerenciadas

É possível criar um perfil que os usuários de outras contas ou pessoas fora da organização podem usar para acessar seus recursos. É possível especificar quem é confiável para assumir o perfil. Para serviços que oferecem suporte a políticas baseadas em recursos ou listas de controle de acesso (ACLs), você pode usar essas políticas para conceder às pessoas acesso aos seus recursos.

Para saber mais, consulte:

- Para saber se as integrações gerenciadas oferecem suporte a esses recursos, consulteComo as integrações gerenciadas funcionam com o IAM.
- Para saber como fornecer acesso aos seus recursos em todos os Contas da AWS que você possui, consulte Como fornecer acesso a um usuário do IAM em outro Conta da AWS que você possui no Guia do usuário do IAM.
- Para saber como fornecer acesso aos seus recursos a terceiros Contas da AWS, consulte Como fornecer acesso Contas da AWS a terceiros no Guia do usuário do IAM.
- Para saber como conceder acesso por meio da federação de identidades, consulte Conceder acesso a usuários autenticados externamente (federação de identidades) no Guia do usuário do IAM.
- Para saber a diferença entre perfis e políticas baseadas em recurso para acesso entre contas, consulte Acesso a recursos entre contas no IAM no Guia do usuário do IAM.

Usando funções vinculadas a serviços para integrações gerenciadas

As integrações gerenciadas para gerenciamento de AWS IoT dispositivos usam funções AWS Identity and Access Management vinculadas a serviços (IAM). Uma função vinculada a serviços é um tipo exclusivo de função do IAM vinculada diretamente às integrações gerenciadas. As funções vinculadas ao serviço são predefinidas por integrações gerenciadas e incluem todas as permissões que o serviço exige para chamar outros AWS serviços em seu nome.

Uma função vinculada ao serviço facilita a configuração de integrações gerenciadas porque você não precisa adicionar manualmente as permissões necessárias. As integrações gerenciadas para gerenciamento de AWS IoT dispositivos definem as permissões de suas funções vinculadas ao serviço e, a menos que definido de outra forma, somente integrações gerenciadas podem assumir suas funções. As permissões definidas incluem a política de confiança e a política de permissões, que não pode ser anexada a nenhuma outra entidade do IAM.

Um perfil vinculado ao serviço poderá ser excluído somente após excluir seus atributos relacionados. Isso protege seus recursos de integrações gerenciadas porque você não pode remover inadvertidamente a permissão para acessar os recursos.

Para obter informações sobre outros serviços que oferecem suporte a funções vinculadas a serviços, consulte AWS Serviços que funcionam com IAM e procure os serviços que têm Sim na coluna Funções vinculadas ao serviço. Escolha um Sim com um link para visualizar a documentação do perfil vinculado a esse serviço.

Permissões de função vinculadas a serviços para integrações gerenciadas

As integrações gerenciadas para gerenciamento de AWS IoT dispositivos usam a função vinculada ao serviço chamada AWSServiceRoleForIoTManagedIntegrações — fornece integrações gerenciadas para permissão de gerenciamento de AWS IoT dispositivos para publicar registros e métricas em seu nome.

A função vinculada ao serviço de AWSService RoleForIo TManaged Integrações confia nos seguintes serviços para assumir a função:

- `iotmanagedintegrations.amazonaws.com`

A política de permissões de função nomeada AWSIo TManaged IntegrationsServiceRolePolicy permite que as integrações gerenciadas concluam as seguintes ações nos recursos especificados:

- Ação: `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"
          ]
        }
      }
    }
  ]
}
```

Você deve configurar permissões para permitir que seus usuários, grupos ou perfis criem, editem ou excluam um perfil vinculado ao serviço. Para obter mais informações, consulte Service-linked role permissions (Permissões de nível vinculado a serviços) no Guia do usuário do IAM.

Criação de uma função vinculada a serviços para integrações gerenciadas

Não é necessário criar manualmente um perfil vinculado ao serviço.
Quando você causa um tipo de evento, como chamar os comandos `PutRuntimeLogConfigurationCreateEventLogConfiguration`, ou `RegisterCustomEndpoint` API na, na ou na AWS API Console de gerenciamento da AWS AWS CLI, as integrações gerenciadas criam a função vinculada ao serviço para você. Para obter mais informações sobre `PutRuntimeLogConfigurationCreateEventLogConfiguration`, ou`RegisterCustomEndpoint`, consulte PutRuntimeLogConfigurationCreateEventLogConfiguration, ou RegisterCustomEndpoint.

Se excluir esse perfil vinculado ao serviço e precisar criá-lo novamente, será possível usar esse mesmo processo para recriar o perfil em sua conta. Quando você causa um tipo de evento, como chamar os comandos`PutRuntimeLogConfiguration`, ou da `RegisterCustomEndpoint` API`CreateEventLogConfiguration`, as integrações gerenciadas criam a função vinculada ao serviço para você novamente. Como alternativa, você pode entrar em contato com AWS o Suporte ao Cliente por meio do AWS Support Center Console. Para obter mais informações sobre os planos de AWS suporte, consulte Compare os planos de suporte da AWS.

Você também pode usar o console do IAM para criar uma função vinculada ao serviço com o caso de uso de IoT ManagedIntegrations — Managed Role. Na AWS CLI ou na AWS API, crie uma função vinculada ao serviço com o nome do `iotmanagedintegrations.amazonaws.com` serviço. Para obter mais informações, consulte Criar um perfil vinculado a serviço no Guia do usuário do IAM. Se você excluir essa função vinculada ao serviço, será possível usar esse mesmo processo para criar a função novamente.

Editando uma função vinculada a serviços para integrações gerenciadas

as integrações gerenciadas não permitem que você edite a função vinculada ao serviço AWSService RoleForIo TManaged Integrations. Depois que criar um perfil vinculado ao serviço, você não poderá alterar o nome do perfil, pois várias entidades podem fazer referência a ele. No entanto, será possível editar a descrição do perfil usando o IAM. Para obter mais informações, consulte Editar um perfil vinculado ao serviço no Guia do usuário do IAM.

Excluindo uma função vinculada a serviços para integrações gerenciadas

Se você não precisar mais usar um recurso ou serviço que requer um perfil vinculado ao serviço, é recomendável excluí-lo. Dessa forma, você não tem uma entidade não utilizada que não seja

monitorada ativamente ou mantida. No entanto, você deve limpar os recursos de seu perfil vinculado ao serviço antes de excluí-lo manualmente.

> **Note**
>
> Se as integrações gerenciadas estiverem usando a função quando você tentar excluir os recursos, a exclusão poderá falhar. Se isso acontecer, espere alguns minutos e tente a operação novamente.

Como excluir manualmente o perfil vinculado ao serviço usando o IAM

Use o console do IAM AWS CLI, o ou a AWS API para excluir a função vinculada ao serviço AWSService RoleForIo TManaged Integrations. Para obter mais informações, consulte Excluir um perfil vinculado ao serviço no Guia do usuário do IAM.

Regiões suportadas para funções vinculadas a serviços de integrações gerenciadas

As integrações gerenciadas para gerenciamento de AWS IoT dispositivos oferecem suporte ao uso de funções vinculadas a serviços em todas as regiões em que o serviço está disponível. Para obter mais informações, consulte Regiões e endpoints da AWS.

Use AWS Secrets Manager para proteção de dados para fluxos de trabalho C2C

AWS Secrets Manager é um serviço de armazenamento secreto que você pode usar para proteger credenciais de banco de dados, chaves de API e outras informações secretas. Em Seguida, no seu código, é possível substituir credenciais codificadas por uma chamada de API para o Secrets Manager. Isso ajuda a garantir que o segredo não possa ser comprometido por alguém que esteja examinando seu código, pois o segredo não está ali. Para obter uma visão geral, consulte o Guia do usuário do AWS Secrets Manager.

O Secrets Manager criptografa segredos usando AWS Key Management Service chaves. Para obter mais informações, consulte Criptografia e descriptografia de dados no AWS Key Management Service.

Integrações gerenciadas para AWS IoT Device Management integrações para que você possa armazenar seus dados no Secrets Manager e usar o ID secreto em suas configurações. AWS Secrets Manager

Como as integrações gerenciadas usam segredos

A Autorização Aberta (OAuth) é um padrão aberto para autorização de acesso delegado, permitindo que os usuários concedam aos sites ou aplicativos acesso às suas informações em outros sites sem compartilhar suas senhas. É uma forma segura de aplicativos de terceiros acessarem os dados do usuário em nome do usuário, fornecendo uma alternativa mais segura ao compartilhamento de senhas.

Em OAuth, um ID de cliente e um segredo de cliente são credenciais que identificam e autenticam um aplicativo cliente quando ele solicita um token de acesso.

Integrações gerenciadas para AWS IoT Device Management usuários se OAuth comunicarem com clientes que usam os fluxos de trabalho C2C. Os clientes precisam fornecer o ID e o segredo do cliente para se comunicarem. Os clientes de integrações gerenciadas armazenarão um ID e um segredo do cliente em suas AWS contas, e as integrações gerenciadas lerão o ID e o segredo do cliente em nossa conta de cliente.

Como criar um segredo

Para criar um segredo, siga as etapas em Criar um AWS Secrets Manager segredo no Guia do AWS Secrets Manager usuário.

Você deve criar seu segredo com uma AWS KMS chave gerenciada pelo cliente para que as integrações gerenciadas leiam o valor secreto. Para obter mais informações, consulte Permissões para a AWS KMS chave no Guia AWS Secrets Manager do usuário.

Você também deve usar as políticas do IAM na seção a seguir.

Conceda acesso a integrações gerenciadas AWS IoT Device Management para recuperar o segredo

Para permitir que as integrações gerenciadas recuperem o valor secreto do Secrets Manager, inclua as seguintes permissões na política de recursos para o segredo ao criá-lo.

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"

        }
      }
  } ]
}
```

Adicione a seguinte declaração à política da sua chave gerenciada pelo cliente AWS KMS .

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/*"
            ]
        }

    ]
}
```

Validação de conformidade para integrações gerenciadas

Para saber se um AWS service (Serviço da AWS) está dentro do escopo de programas de conformidade específicos, consulte Serviços da AWS Escopo por Programa de Conformidade Serviços da AWS e escolha o programa de conformidade em que você está interessado. Para obter informações gerais, consulte Programas de AWS conformidade Programas AWS de .

Você pode baixar relatórios de auditoria de terceiros usando AWS Artifact. Para obter mais informações, consulte Baixar relatórios em AWS Artifact .

Sua responsabilidade de conformidade ao usar Serviços da AWS é determinada pela confidencialidade de seus dados, pelos objetivos de conformidade de sua empresa e pelas leis e regulamentações aplicáveis. Para obter mais informações sobre sua responsabilidade de conformidade ao usar Serviços da AWS, consulte a documentação AWS de segurança.

Use integrações gerenciadas com endpoints de interface VPC

Você pode estabelecer uma conexão privada entre sua Amazon VPC e integrações AWS IoT gerenciadas criando uma interface de endpoint Amazon VPC. Os endpoints de interface são alimentados por AWS PrivateLink, uma tecnologia que permite acessar serviços de forma privada usando endereços IP privados. A AWS PrivateLink restringe todo o tráfego de rede entre suas integrações gerenciadas de VPC e IoT à rede Amazon. Você não precisa de um gateway de internet, dispositivo NAT ou conexão VPN.

Você não precisa usar AWS PrivateLink, mas é recomendado. Para obter mais informações sobre AWS PrivateLink endpoints de VPC, consulte Como acessar AWS serviços por meio do Guia AWS PrivateLink.AWS PrivateLink

Tópicos

- Considerações sobre integrações AWS IoT gerenciadas (VPC endpoints)
- Criação de uma interface VPC endpoint para integrações gerenciadas AWS IoT
- Testando seu VPC endpoint
- Controle do acesso aos serviços por meio de VPC endpoints
- Preços
- Limitações

Considerações sobre integrações AWS IoT gerenciadas (VPC endpoints)

Antes de configurar uma interface VPC endpoint para integrações AWS IoT gerenciadas, analise as propriedades e limitações do endpoint da interface no Guia.AWS PrivateLink

AWS IoT As integrações gerenciadas permitem fazer chamadas para todas as ações de API da sua VPC por meio de endpoints de VPC de interface.

Endpoints compatíveis

AWS IoT As integrações gerenciadas oferecem suporte a VPC endpoints para as seguintes interfaces de serviço:

- API do plano de controle: `com.amazonaws.region.iotmanagedintegrations.api`

Endpoints não suportados

Os seguintes endpoints de integrações AWS IoT gerenciadas não oferecem suporte a endpoints de VPC:

- Endpoints MQTT: os dispositivos MQTT são normalmente implantados em ambientes de usuário final e não em ambientes internos, tornando a integração desnecessária. AWS VPCs AWS PrivateLink
- OAuth endpoints de retorno de chamada: muitas plataformas de terceiros não operam na AWS infraestrutura, reduzindo os benefícios do AWS PrivateLink suporte a fluxos. OAuth

Disponibilidade

AWS IoT Os endpoints de VPC de integrações gerenciadas estão disponíveis nas seguintes regiões: AWS

- Canadá (Central): ca-central-1
- Europa (Irlanda): eu-west-1

Outras regiões serão suportadas à medida que as integrações AWS IoT gerenciadas expandirem sua disponibilidade.

Compatibilidade com pilha dupla

AWS IoT Integrações gerenciadas Os endpoints VPC oferecem suporte IPv4 tanto ao tráfego quanto ao tráfego. IPv6 Você pode criar VPC endpoints com os seguintes tipos de endereço IP:

- IPv4: atribui IPv4 endereços às interfaces de rede de endpoints
- IPv6: atribui IPv6 endereços às interfaces de rede de endpoints (requer apenas IPv6 sub-redes)
- Dualstack: atribui IPv6 endereços IPv4 e endereços às interfaces de rede de endpoints

Criação de uma interface VPC endpoint para integrações gerenciadas AWS IoT

Você pode criar um VPC endpoint para o serviço de integrações AWS IoT gerenciadas usando o Amazon VPC Console ou o (CLI). AWS CLI AWS

Para criar uma interface VPC endpoint para integrações AWS IoT gerenciadas (console)

1. Abra o console da Amazon VPC no console da Amazon VPC.
2. No painel de navegação, escolha Endpoints.
3. Escolha Criar endpoint.
4. Em Categoria do serviço, escolha Serviços do AWS .
5. Em Nome do serviço, selecione o nome do serviço que corresponde à sua AWS região. Por exemplo:
   - `com.amazonaws.ca-central-1.iotmanagedintegrations.api`
   - `com.amazonaws.eu-west-1.iotmanagedintegrations.api`
6. Para VPC, selecione a VPC a partir da qual você acessará as integrações gerenciadas. AWS IoT
7. Para Configurações adicionais, a opção Ativar nome DNS é selecionada por padrão. Recomendamos que você mantenha essa configuração. Isso garante que as solicitações para os endpoints de serviço público de integrações AWS IoT gerenciadas sejam resolvidas para seu endpoint da Amazon VPC.
8. Em Sub-redes, selecione as sub-redes nas quais serão criadas as interfaces de rede de endpoint. Você só pode selecionar uma sub-rede por zona de disponibilidade.
9. Em IP address type (Tipo de endereço IP), escolha uma das seguintes opções:

- IPv4: atribua IPv4 endereços às interfaces de rede do endpoint
- IPv6: atribua IPv6 endereços às interfaces de rede do endpoint (suportado somente se todas as sub-redes selecionadas forem somente -s) IPv6
- Dualstack: atribua IPv6 endereços IPv4 e endereços às interfaces de rede do endpoint

10 Em Grupos de segurança, selecione os grupos de segurança para associar às interfaces de rede do endpoint. As regras do grupo de segurança devem permitir a comunicação entre a interface de rede do endpoint e os recursos em sua VPC que se comunicam com o serviço.

11 Em Política, escolha Acesso total para permitir todas as operações de todos os diretores em todos os recursos no endpoint da interface. Para restringir o acesso, escolha Personalizado e especifique uma política.

12 (Opcional) Para adicionar uma tag, escolha Add new tag (Adicionar nova tag) e insira a chave e o valor da tag.

13 Escolha Criar endpoint.

Para criar uma interface VPC endpoint para integrações gerenciadas de IoT (AWS CLI)

Use o create-vpc-endpointcomando e especifique o ID da VPC, o tipo de endpoint da VPC (interface), o nome do serviço, as sub-redes que usarão o endpoint e os grupos de segurança a serem associados às interfaces de rede do endpoint.

```
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
```

Testando seu VPC endpoint

Depois de criar seu VPC endpoint, você pode testar a conexão fazendo chamadas de API para integrações AWS IoT gerenciadas a partir de uma instância EC2 em sua VPC.

Pré-requisitos

- Uma EC2 instância em uma sub-rede privada dentro da sua VPC

- Permissões apropriadas do IAM para operações de integrações AWS IoT gerenciadas
- Regras de grupo de segurança que permitem tráfego HTTPS (porta 443) para o VPC endpoint

Testar a conexão

1. Conecte-se à sua EC2 instância da Amazon na sub-rede privada.
2. Verifique a resolução de DNS para o nome DNS privado:

```
dig api.iotmanagedintegrations.region.api.aws
```

3. Teste a conectividade HTTPS:

```
curl -v https://api.iotmanagedintegrations.region.api.aws
```

4. Faça uma chamada à API de integrações AWS IoT gerenciadas:

```
aws iot-managed-integrations list-destinations \
  --region region \
  --endpoint-url https://api.iotmanagedintegrations.region.api.aws
```

`region`Substitua pela sua AWS região (por exemplo,`ca-central-1`).

Controle do acesso aos serviços por meio de VPC endpoints

Uma política de VPC endpoint é uma política de recursos do IAM que você anexa a uma interface VPC endpoint ao criar ou modificar o endpoint. Se você não anexar uma política quando criar um endpoint, anexaremos uma política padrão que permita o acesso total ao serviço. Uma política de endpoint não substitui políticas de usuário do IAM nem políticas de serviço específicas. É uma política separada para controlar o acesso do endpoint ao serviço especificado.

Políticas de endpoint devem ser gravadas em formato JSON. Para obter mais informações, consulte Controlar o acesso a serviços com VPC endpoints no Guia do usuário da Amazon VPC.

Exemplo: política de VPC endpoint para ações de integrações gerenciadas AWS IoT

Veja a seguir um exemplo de uma política de endpoint para integrações AWS IoT gerenciadas. Essa política permite que os usuários se conectem a integrações AWS IoT gerenciadas por meio do VPC endpoint para acessar destinos, mas nega o acesso aos armários de credenciais.

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

Exemplo: política de VPC endpoint que restringe o acesso a uma função específica do IAM

A política de VPC endpoint a seguir permite acesso a integrações AWS IoT gerenciadas somente para diretores do IAM que tenham a função específica do IAM em sua cadeia de confiança. Todos os outros diretores do IAM têm acesso negado.

```
{
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "*",
```

```
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:PrincipalArn": "arn:aws:iam::123456789012:role/
IoTManagedIntegrationsVPCRole"
                }
            }
        }
    ]
}
```

Preços

São cobradas taxas padrão pela criação e uso de uma interface VPC endpoint com AWS IoT integrações gerenciadas. Para obter mais informações, consulte Preços do AWS PrivateLink.

Limitações

- A CreateAccountAssociationAPI foi projetada para funcionar OAuth com serviços de nuvem de terceiros, o que exige que a solicitação saia da rede Amazon. Isso é importante para os clientes AWS PrivateLink que usam conter seu tráfego na VPC, pois AWS PrivateLink não é possível fornecer end-to-end contenção completa para essa chamada de API.
- Os VPC endpoints para integrações AWS IoT gerenciadas não estão disponíveis em. AWS GovCloud (US) Regions

Para as limitações gerais do VPC endpoint, consulte Propriedades e limitações do endpoint da interface no Guia do usuário do Amazon VPC.

Conecte-se a integrações gerenciadas para endpoints AWS IoT Device Management FIPS

AWS IoT fornece um endpoint de plano de controle que suporta o Federal Information Processing Standard (FIPS) 140-2. Os endpoints compatíveis com FIPS são diferentes dos endpoints padrão. AWS Para interagir com integrações gerenciadas de forma compatível com FIPS, você deve usar os endpoints descritos abaixo com seu cliente compatível com FIPS. AWS IoT Device Management O AWS IoT console não é compatível com FIPS.

As seções a seguir descrevem como acessar o AWS IoT endpoint compatível com FIPS usando a API REST, um SDK ou o. AWS CLI

Endpoints do ambiente de gerenciamento

Os endpoints do plano de controle compatíveis com FIPS que suportam as operações de integrações gerenciadas e seus AWS CLI comandos relacionados estão listados em FIPS Endpoints by Service. Em FIPS Endpoints by Service, encontre o AWS IoT Device Management serviço de integrações gerenciadas e procure o endpoint para o seu. Região da AWS

Para usar o endpoint compatível com FIPS ao acessar as operações de integrações gerenciadas, use o AWS SDK ou a API REST com o endpoint apropriado para você. Região da AWS

Para usar o endpoint compatível com FIPS ao executar comandos CLI de integrações gerenciadas, adicione ao comando o --endpoint parâmetro com o endpoint apropriado para você. Região da AWS

Monitoramento de integrações gerenciadas

O monitoramento é uma parte importante da manutenção da confiabilidade, disponibilidade e desempenho das integrações gerenciadas e de suas outras soluções da AWS. A AWS fornece as seguintes ferramentas de monitoramento para observar integrações gerenciadas, relatar quando algo está errado e realizar ações automáticas quando apropriado:

- AWS CloudTrailcaptura chamadas de API e eventos relacionados feitos por ou em nome de sua AWS conta e entrega os arquivos de log para um bucket do Amazon S3 que você especificar. Você pode identificar quais usuários e contas ligaram AWS, o endereço IP de origem a partir do qual as chamadas foram feitas e quando elas ocorreram. Para obter mais informações, consulte o Guia do usuário do AWS CloudTrail.

Registro de chamadas de API de integrações gerenciadas usando AWS CloudTrail

As integrações gerenciadas são AWS CloudTrailintegradas com um serviço que fornece um registro das ações realizadas por um usuário, função ou um AWS service (Serviço da AWS). CloudTrail captura todas as chamadas de API para integrações gerenciadas como eventos. As chamadas capturadas incluem chamadas do console de integrações gerenciadas e chamadas de código para as operações da API de integrações gerenciadas. Usando as informações coletadas por CloudTrail, você pode determinar a solicitação feita às integrações gerenciadas, o endereço IP a partir do qual a solicitação foi feita, quando foi feita e detalhes adicionais.

Cada entrada de log ou evento contém informações sobre quem gerou a solicitação. As informações de identidade ajudam a determinar o seguinte:

- Se a solicitação foi feita com credenciais de usuário raiz ou credenciais de usuário.
- Se a solicitação foi feita em nome de um usuário do Centro de Identidade do IAM.
- Se a solicitação foi feita com credenciais de segurança temporárias de um perfil ou de um usuário federado.
- Se a solicitação foi feita por outro AWS service (Serviço da AWS).

CloudTrail está ativo Conta da AWS quando você cria a conta e você tem acesso automático ao histórico de CloudTrail eventos. O histórico de CloudTrail eventos fornece um registro visível,

pesquisável, baixável e imutável dos últimos 90 dias de eventos de gerenciamento registrados em um. Região da AWS Para obter mais informações, consulte Trabalhando com o histórico de CloudTrail eventos no Guia AWS CloudTrail do usuário. Não há CloudTrail cobrança pela visualização do histórico de eventos.

Para um registro contínuo dos eventos dos Conta da AWS últimos 90 dias, crie uma trilha ou um armazenamento de dados de eventos do CloudTrailLake.

CloudTrail trilhas

Uma trilha permite CloudTrail entregar arquivos de log para um bucket do Amazon S3. Todas as trilhas criadas usando o Console de gerenciamento da AWS são multirregionais. Só é possível criar uma trilha de região única ou de várias regiões usando a AWS CLI. É recomendável criar uma trilha multirregional porque você captura todas as atividades Regiões da AWS em sua conta. Ao criar uma trilha de região única, é possível visualizar somente os eventos registrados na Região da AWS da trilha. Para obter mais informações sobre trilhas, consulte Criar uma trilha para a Conta da AWS e Criar uma trilha para uma organização no Guia do usuário do AWS CloudTrail .

Você pode entregar uma cópia dos seus eventos de gerenciamento em andamento para o bucket do Amazon S3 sem nenhum custo CloudTrail criando uma trilha. No entanto, existem taxas de armazenamento do Amazon S3. Para obter mais informações sobre CloudTrail preços, consulte AWS CloudTrail Preços. Para receber informações sobre a definição de preços do Amazon S3, consulte Definição de preços do Amazon S3.

CloudTrail Armazenamentos de dados de eventos em Lake

CloudTrail O Lake permite que você execute consultas baseadas em SQL em seus eventos. CloudTrail O Lake converte eventos existentes no formato JSON baseado em linhas para o formato Apache ORC. O ORC é um formato colunar de armazenamento otimizado para recuperação rápida de dados. Os eventos são agregados em armazenamentos de dados de eventos, que são coleções imutáveis de eventos baseados nos critérios selecionados com a aplicação de seletores de eventos avançados. Os seletores que aplicados a um armazenamento de dados de eventos controlam quais eventos persistem e estão disponíveis para consulta. Para obter mais informações sobre o CloudTrail Lake, consulte Trabalhando com o AWS CloudTrail Lake no Guia AWS CloudTrail do Usuário.

CloudTrail Os armazenamentos e consultas de dados de eventos em Lake incorrem em custos. Ao criar um armazenamento de dados de eventos, você escolhe a opção de preço que deseja usar para ele. A opção de preço determina o custo para a ingestão e para o armazenamento de

eventos, e o período de retenção padrão e máximo para o armazenamento de dados de eventos. Para obter mais informações sobre CloudTrail preços, consulte AWS CloudTrail Preços.

Eventos de gestão em CloudTrail

Os eventos de gerenciamento fornecem informações sobre as operações de gerenciamento que são realizadas nos recursos do seu Conta da AWS. Também são conhecidas como operações de ambiente de gerenciamento. Por padrão, CloudTrail registra eventos de gerenciamento.

As integrações gerenciadas registram as seguintes operações do plano de controle de integrações gerenciadas CloudTrail como eventos de gerenciamento.

- `CreateCloudConnector`
- `UpdateCloudConnector`
- `GetCloudConnector`
- `DeleteCloudConnector`
- `ListCloudConnectors`
- `CreateConnectorDestination`
- `UpdateConnectorDestination`
- `GetConnectorDestination`
- `DeleteConnectorDestination`
- `ListConnectorDestinations`
- `CreateAccountAssociation`
- `UpdateAccountAssociation`
- `GetAccountAssociation`
- `DeleteAccountAssociation`
- `ListAccountAssociations`
- `StartAccountAssociationRefresh`
- `ListManagedThingAccountAssociations`
- `RegisterAccountAssociation`
- `DeregisterAccountAssociation`
- `SendConnectorEvent`
- `ListDeviceDiscoveries`

- `ListDiscoveredDevices`

Exemplos de evento

Um evento representa uma única solicitação de qualquer fonte e inclui informações sobre a operação de API solicitada, a data e a hora da operação, os parâmetros da solicitação e assim por diante. CloudTrail os arquivos de log não são um rastreamento de pilha ordenado das chamadas públicas de API, portanto, os eventos não aparecem em nenhuma ordem específica.

O exemplo a seguir mostra um CloudTrail evento que demonstra uma operação de `CreateCloudConnector` API bem-sucedida.

CloudTrail Evento bem-sucedido com a operação **`CreateCloudConnector`** da API.


```
{
    "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"
}
```

O exemplo a seguir mostra um CloudTrail evento que demonstra uma operação de `ListDiscoveredDevices` API bem-sucedida.

CloudTrail Evento bem-sucedido com a operação **`ListDiscoveredDevices`** da API.

```
{
    "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"
}
```


Para obter informações sobre o conteúdo do CloudTrail registro, consulte o conteúdo do CloudTrail registro no Guia AWS CloudTrail do usuário.

Histórico de documentos para as integrações gerenciadas Guia do desenvolvedor

A tabela a seguir descreve as versões da documentação para integrações gerenciadas.

Alteração	Descrição	Data
Versão de disponibilidade geral	Versão de disponibilidade geral do Guia do desenvolvedor de integrações gerenciadas	25 de junho de 2025
Versão prévia inicial	Versão prévia inicial do Guia do desenvolvedor de integrações gerenciadas	3 de março de 2025

As traduções são geradas por tradução automática. Em caso de conflito entre o conteúdo da tradução e da versão original em inglês, a versão em inglês prevalecerá.