aws

개발자 가이드

에 대한 관리형 통합 AWS IoT Device Management

에 대한 관리형 통합 AWS IoT Device Management: 개발자 가이드

Table of Contents

관리형 통합이란 무엇입니까 AWS IoT Device Management?

용 관리형 통합은 IoT 솔루션 공급자가 수백 개 제조업체의 IoT 디바이스 제어 및 관리를 통합하는 AWS IoT Device Management 데 도움이 됩니다. 관리형 통합을 사용하면 디바이스 공급업체 또는 연결 프로토콜에 관계없이 디바이스 설정 워크플로를 자동화하고 여러 디바이스에서 상호 운용성을 지원할 수 있습니다. 관리형 통합을 사용하면 단일 사용자 인터페이스와 APIs 세트를 사용하여 다양한 디바이스를 제어, 관리 및 운영할 수 있습니다.

주제

지원되는 리전

에 대한 관리형 통합 AWS IoT Device Management 은 다음 리전에서 지원됩니다.

- 캐나다(중부)
- 유럽(아일랜드)

관리형 통합을 처음 사용하시나요?

관리형 통합을 처음 사용하는 경우 먼저 다음 섹션을 읽는 것이 좋습니다.

- 관리형 통합 설정
- 에 대한 관리형 통합 시작하기 AWS IoT Device Management

관리형 통합 개요

다음 이미지는 관리형 통합에 대한 개략적인 개요를 제공합니다.

관리형 통합 용어

관리형 통합에는 자체 디바이스 구현을 관리하기 위해 이해하는 데 중요한 여러 개념과 용어가 있습니다. 다음 섹션에서는 관리형 통합에 대한 이해를 높이기 위해 이러한 주요 개념과 용어를 간략하게 설명합니다.

일반 관리형 통합 용어

관리형 통합에 대해 이해해야 할 중요한 개념은 사물과 비교한 관리 AWS IoT Core 형 사물입니다.

- AWS IoT Core 사물: AWS IoT Core 사물은 디지털 표현을 제공하는 AWS IoT Core 구문입니다. 개발자는 정책, 데이터 스토리지, 규칙, 작업, MQTT 주제 및 데이터 스토리지로의 디바이스 상태 전달을 관리해야 합니다. AWS IoT Core 사물에 대한 자세한 내용은 를 사용하여 디바이스 관리를 AWS IoT참조하세요.
- 관리형 통합 관리형 사물: 관리형 사물을 사용하면 디바이스 상호 작용을 간소화하는 추상화를 제공하며 개발자가 규칙, 작업, MQTT 주제 및 정책과 같은 항목을 생성할 필요가 없습니다.

Cloud-to-cloud 용어

관리형 통합과 통합되는 물리적 디바이스는 타사 클라우드 공급자에서 비롯될 수 있습니다. 이러한 디바이스를 관리형 통합에 온보딩하고 타사 클라우드 공급자와 통신하기 위해 다음 용어는 이러한 워크플로를 지원하는 몇 가지 주요 개념을 다룹니다.

- Cloud-to-cloud(C2C) 커넥터: C2C 커넥터는 관리형 통합과 타사 클라우드 공급자 간의 연결을 설정합니다.
- 타사 클라우드 공급자: 관리형 통합 외부에서 제조 및 관리되는 디바이스의 경우 타사 클라우드 공급자는 최종 사용자를 위해 이러한 디바이스를 제어할 수 있으며 관리형 통합은 디바이스 명령과 같은 다양한 워크플로를 위해 타사 클라우드 공급자와 통신합니다.

데이터 모델 용어

관리형 통합은 데이터 모델을 사용하여 디바이스 간의 데이터 및 end-to-end 통신을 구성합니다. 다음 용어는 이러한 두 데이터 모델을 이해하기 위한 몇 가지 주요 개념을 다룹니다.

- 디바이스: 전체 기능 세트를 제공하기 위해 여러 노드가 함께 작동하는 물리적 디바이스(예: 비디오 도어벨)를 나타내는 개체입니다.

- 엔드포인트: 엔드포인트는 독립 실행형 기능(링거, 모션 감지, 비디오 도어벨의 조명)을 캡슐화합니다.
- 기능: 엔드포인트에서 기능을 사용할 수 있도록 하는 데 필요한 구성 요소를 나타내는 엔터티입니다(비디오 도어벨의 종 기능에 버튼 또는 조명과 차임).
- 작업: 디바이스의 기능과의 상호 작용을 나타내는 엔터티입니다(벨을 울리거나 문 앞에 있는 사람 보기).
- 이벤트: 디바이스의 기능에서 이벤트를 나타내는 개체입니다. 디바이스는 이벤트를 전송하여 인시던트/경보, 센서의 활동 등을 보고할 수 있습니다(예: 문에 충격/고리가 있음).
- 속성: 디바이스 상태의 특정 속성을 나타내는 엔터티입니다(벨이 울리고, 포치 조명이 켜져 있고, 카메라가 레코딩 중입니다).
- 데이터 모델: 데이터 계층은 애플리케이션의 기능을 지원하는 데 도움이 되는 데이터 및 동사 요소에 해당합니다. 애플리케이션은 디바이스와 상호 작용하려는 의도가 있을 때 이러한 데이터 구조에서 작동합니다. 자세한 내용은 GitHub 웹 사이트의 connectedhomeip를 참조하세요.
- 스키마: 스키마는 JSON 형식의 데이터 모델을 나타냅니다.

관리형 통합 설정

다음 섹션에서는에서 관리형 통합을 사용하기 위한 초기 설정을 안내합니다 AWS IoT Device Management.

주제

에 가입 AWS 계정

이 없는 경우 다음 단계를 AWS 계정완료하여 생성합니다.

에 가입하려면 AWS 계정

1. https://portal.aws.amazon.com/billing/signup을 엽니다.
2. 온라인 지시 사항을 따릅니다.

   등록 절차 중 전화 또는 텍스트 메시지를 받고 전화 키패드로 확인 코드를 입력하는 과정이 있습니다.

   에 가입하면 AWS 계정AWS 계정 루트 사용자이 생성됩니다. 루트 사용자에게는 계정의 모든 AWS 서비스 및 리소스에 액세스할 권한이 있습니다. 보안 모범 사례는 사용자에게 관리 액세스 권한을 할당하고, 루트 사용자만 사용하여 루트 사용자 액세스 권한이 필요한 작업을 수행하는 것입니다.

AWS 는 가입 프로세스가 완료된 후 확인 이메일을 보냅니다. 언제든지 https://aws.amazon.com/으로 이동하고 내 계정을 선택하여 현재 계정 활동을 보고 계정을 관리할 수 있습니다.

관리자 액세스 권한이 있는 사용자 생성

에 가입한 후 일상적인 작업에 루트 사용자를 사용하지 않도록 관리 사용자를 AWS 계정보호 AWS IAM Identity Center, AWS 계정 루트 사용자활성화 및 생성합니다.

보안 AWS 계정 루트 사용자

1. 루트 사용자를 선택하고 AWS 계정 이메일 주소를 입력하여 계정 소유자AWS Management Console로에 로그인합니다. 다음 페이지에서 비밀번호를 입력합니다.

   루트 사용자를 사용하여 로그인하는 데 도움이 필요하면 AWS 로그인 User Guide의 루트 사용자로 로그인을 참조하세요.
2. 루트 사용자의 다중 인증(MFA)을 활성화합니다.

   지침은 IAM 사용 설명서의 AWS 계정 루트 사용자(콘솔)에 대한 가상 MFA 디바이스 활성화를 참조하세요.

관리자 액세스 권한이 있는 사용자 생성

1. IAM Identity Center를 활성화합니다.

   지침은 AWS IAM Identity Center 사용 설명서의 AWS IAM Identity Center설정을 참조하세요.
2. IAM Identity Center에서 사용자에게 관리 액세스 권한을 부여합니다.

   를 자격 증명 소스 IAM Identity Center 디렉터리 로 사용하는 방법에 대한 자습서는 사용 AWS IAM Identity Center 설명서의 기본값으로 사용자 액세스 구성을 IAM Identity Center 디렉터리 참조하세요.

관리 액세스 권한이 있는 사용자로 로그인

- IAM IDentity Center 사용자로 로그인하려면 IAM Identity Center 사용자를 생성할 때 이메일 주소로 전송된 로그인 URL을 사용합니다.

  IAM Identity Center 사용자를 사용하여 로그인하는 데 도움이 필요하면 사용 설명서의 AWS 액세스 포털에 로그인을 참조하세요. AWS 로그인

추가 사용자에게 액세스 권한 할당

1. IAM Identity Center에서 최소 권한 적용 모범 사례를 따르는 권한 세트를 생성합니다.

   지침은AWS IAM Identity Center 사용 설명서의 Create a permission set를 참조하세요.
2. 사용자를 그룹에 할당하고, 그룹에 Single Sign-On 액세스 권한을 할당합니다.

   지침은 AWS IAM Identity Center 사용 설명서의 Add groups를 참조하세요.

에 대한 관리형 통합 시작하기 AWS IoT Device Management

다음 섹션에서는 관리형 통합 사용을 시작하기 위해 수행해야 하는 단계를 간략하게 설명합니다.

주제

디바이스 유형

관리형 통합은 다양한 유형의 디바이스를 관리합니다. 각 디바이스는 다음 세 가지 범주 중 하나에 속합니다.

- 직접 연결 디바이스:이 유형의 디바이스는 관리형 통합 엔드포인트에 직접 연결됩니다. 일반적으로 이러한 디바이스는 직접 연결을 위한 관리형 통합 엔드 디바이스 SDK를 포함하는 디바이스 제조업체에서 빌드하고 관리합니다.
- 허브 연결 디바이스: 이러한 디바이스는 디바이스 검색, 온보딩 및 제어 기능을 관리하는 관리형 통합 Hub SDK를 실행하는 허브를 통해 관리형 통합에 연결됩니다. 최종 사용자는 버튼 누름 시작 또는 바코드 스캔을 사용하여 이러한 디바이스를 온보딩할 수 있습니다.

  허브에 연결된 디바이스 온보딩에는 다음 두 가지 워크플로가 지원됩니다.

  - 최종 사용자가 버튼을 눌러 디바이스 검색을 시작합니다.
  - 디바이스 연결을 수행하기 위한 바코드 기반 스캔
- Cloud-to-cloud(C2C) 디바이스: 디바이스 제어를 위해 자체 클라우드 인프라와 브랜드 모바일 애플리케이션을 유지 관리하는 공급업체가 설계하고 관리하는 디바이스입니다. 관리형 통합 고객은 사전 구축된 C2C 커넥터 카탈로그에 액세스하거나 자체 커넥터를 생성하여 통합 인터페이스를 통해 여러 타사 공급업체 클라우드에서 작동하는 IoT 솔루션을 개발할 수 있습니다.

  최종 사용자가 C2C 디바이스를 처음 켜는 경우 디바이스 기능과 메타데이터를 얻으려면 관리형 통합을 위해 해당 타사 클라우드 공급자와 프로비저닝해야 합니다. 프로비저닝 워크플로를 완료한 후 관리형 통합은 최종 사용자를 대신하여 클라우드 디바이스 및 타사 클라우드 공급자와 통신할 수 있습니다.

> **Note**
>
> 허브는 위에 나열된 특정 디바이스 유형이 아닙니다. 목적은 스마트 홈 디바이스의 컨트롤러 역할을 수행하고 관리형 통합과 타사 클라우드 공급자 간의 연결을 용이하게 하는 것입니다. 이 역할은 위에 나열된 디바이스 유형과 허브로 사용할 수 있습니다.

암호화 키 구성

보안은 최종 사용자, 관리형 통합 및 타사 클라우드 간에 라우팅되는 데이터에 매우 중요합니다. 디바이스 데이터를 보호하기 위해 지원하는 방법 중 하나는 데이터를 라우팅하기 위해 보안 암호화 키를 활용하는 end-to-end 암호화입니다.

관리형 통합의 고객은 암호화 키를 사용하기 위해 다음과 같은 두 가지 옵션을 사용할 수 있습니다.

- 기본 관리형 통합 관리형 암호화 키를 사용합니다.
- 생성한 AWS KMS key 를 제공합니다.

AWS KMS 서비스에 대한 자세한 내용은 키 관리 서비스(KMS)를 참조하세요.

관리형 통합 API 참조 가이드에서 PutDefaultEncryptionConfiguration API를 호출하면 사용하려는 암호화 키 옵션을 업데이트할 수 있는 액세스 권한이 부여됩니다. 기본적으로 관리형 통합은 기본 관리형 통합 관리형 암호화 키를 사용합니다. PutDefaultEncryptionConfiguration API를 사용하여 언제든지 암호화 키 구성을 업데이트할 수 있습니다.

또한 GetDefaultEncryptionConfiguration API 명령을 호출하면 기본 또는 지정된 리전에 있는 AWS 계정의 암호화 구성에 대한 정보가 반환됩니다.

온보딩 기법

온보딩 유형은 다음과 같습니다.

직접 연결된 디바이스 온보딩

직접 연결된 디바이스를 온보딩하는 단계는 프로비저닝 담당자 섹션을 참조하세요.

허브 온보딩

허브 온보딩 단계는 허브를 관리형 통합에 온보딩 섹션을 참조하세요.

허브 연결 디바이스 온보딩

허브 연결 디바이스를 온보딩하는 단계는 디바이스 온보딩 및 허브에서 작동 섹션을 참조하세요.

Cloud-to-cloud 디바이스 온보딩

타사 클라우드 공급업체에서 관리형 통합으로 클라우드 디바이스를 온보딩하는 단계는 C2C(Cloud-to-Cloud) 커넥터 사용 섹션을 참조하세요.

디바이스 프로비저닝

디바이스 프로비저닝은 디바이스 온보딩 프로세스를 용이하게 하고, 전체 디바이스 수명 주기를 감독하며, 관리형 통합의 다른 측면이 액세스할 수 있는 디바이스 정보를 위한 중앙 집중식 리포지토리를 설정합니다. 관리형 통합은 허브 디바이스를 통해 간접적으로 연결된 디바이스 소프트웨어 개발 키트(SDK) 또는 commercial-off-the-shelf품(COTS) 디바이스를 통해 직접 연결된 자사 고객 디바이스를 수용하여 다양한 디바이스 유형을 관리하기 위한 통합 인터페이스를 제공합니다.

관리형 통합의 각 디바이스에는 디바이스 유형에 관계없이 라는 전역적으로 고유한 식별자가 있습니다managedThingId. 이 식별자는 전체 디바이스 수명 주기 동안 디바이스의 온보딩 및 관리에 사용됩니다. 관리형 통합으로 완전히 관리되며 모든의 모든 관리형 통합에서 해당 특정 디바이스에 고유합니다 AWS 리전. 디바이스가 관리형 통합에 처음 추가되면이 식별자가 생성되어 관리형 통합의 관리형 사물에 연결됩니다. 관리형 사물은 물리적 디바이스의 모든 디바이스 메타데이터를 미러링하기 위해 관리형 통합 내에서 물리적 디바이스를 디지털 방식으로 표현한 것입니다. 타사 디바이스의 경우 물리적 디바이스를 나타내는 관리형 통합에 managedThingId 저장된 외에도 타사 클라우드와 관련된 별도의 고유 식별자가 있을 수 있습니다.

프로비저닝되는 디바이스는 온보딩 흐름의 단계에 따라 상태가 다를 수 있습니다. 다음 목록은 각 프로비저닝 상태를 설명합니다.

- ACTIVATED: 디바이스를 찾았으며 명령 및 제어를 사용할 수 있습니다.
- DISCOVERED: 디바이스를 찾았지만 명령 및 제어를 아직 사용할 수 없습니다.
- UNASSOCIATED: 관리형 사물이 생성되었지만 추가 작업을 검색해야 합니다. AWS 클라우드 또는 AWS IoT 관리형 통합 컨트롤러(허브)에서 연결할 수 없음
- PRE_ASSOCIATED: 관리형 사물이 생성되었으며 전원이 켜지거나 연결되면 자동으로 검색할 수 있습니다. AWS 클라우드 또는 AWS IoT 관리형 통합 컨트롤러(허브)에서는 연결할 수 없습니다.
- DELETE_IN_PROGRESS: 비동기 삭제 프로세스가 시작되었습니다.
- 삭제됨: 디바이스가에서 삭제되었습니다 AWS 클라우드.
- ISOLATED: 이전에 검색되거나 활성화된 관리형 사물로 더 이상 연결할 수 없습니다. 예를 들어 커넥터 연결이 모두 삭제된 타사 클라우드용 디바이스가 있습니다.

다음 온보딩 흐름은 관리형 통합으로 허브를 프로비저닝하기 위한 것입니다.

허브를 관리형 통합에 온보딩: 디바이스 인증, 통신 및 설정을 처리하기 위해 함께 작동하는 코어 프로비저너 및 프로토콜별 플러그인을 설정합니다.

관리형 통합으로 허브 연결 디바이스를 프로비저닝하기 위해 다음 온보딩 흐름이 제공됩니다.

- 단순 설정(SS): 최종 사용자는 IoT 디바이스의 전원을 켜고 디바이스 제조업체 애플리케이션을 사용하여 QR 코드를 스캔합니다. 그런 다음 디바이스가 관리형 통합 클라우드에 등록되고 IoT 허브에 연결됩니다.
- 제로 터치 설정(ZTS): 디바이스가 공급망의 업스트림에 사전 연결되어 있습니다. 예를 들어 최종 사용자가 디바이스 QR 코드를 스캔하는 대신이 단계는 이전에 완료되어 디바이스를 고객 계정에 미리 연결합니다.
- 사용자 안내 설정(UGS): 최종 사용자는 디바이스의 전원을 켜고 대화형 단계에 따라 관리형 통합에 온보딩합니다. 여기에는 IoT 허브에서 버튼 누르기, 디바이스 제조업체 앱 사용 또는 허브와 디바이스 모두에서 버튼 누르기가 포함될 수 있습니다. 단순 설정이 실패하면이 방법을 사용할 수 있습니다.

> **Note**
>
> 관리형 통합의 디바이스 프로비저닝 워크플로는 디바이스의 온보딩 요구 사항에 구애받지 않습니다. 관리형 통합은 디바이스 유형 또는 디바이스 프로토콜에 관계없이 디바이스를 온보딩하고 관리하기 위한 간소화된 사용자 인터페이스를 제공합니다.

디바이스 및 디바이스 프로파일 수명 주기

디바이스 및 디바이스 프로파일의 수명 주기를 관리하면 디바이스 플릿이 안전하고 효율적으로 실행될 수 있습니다.

주제

장치

초기 온보딩 중에 관리형 통합은 관리형 사물이라는 물리적 디바이스의 디지털 트윈을 생성합니다. 관리형 사물에는 모든 리전의 관리형 통합에서 디바이스를 식별하기 위한 글로벌 고유 식별자를 `managedThingID` 제공하는가 있습니다. 디바이스는 관리형 통합 또는 타사 디바이스용 타사 클라우드와의 실시간 통신을 위해 프로비저닝 중에 로컬 허브와 페어링됩니다. 디바이스는와 같은 관리형 사물에 대한 퍼블릭 APIs의 `owner` 파라미터로 식별되는 소유자와도 연결됩니다`GetManagedThing`. 디바이스는 디바이스 유형에 따라 해당 디바이스 프로파일에 연결됩니다.

> **Note**
>
> 물리적 디바이스를 서로 다른 고객에게 여러 번 프로비저닝하면 레코드가 여러 개 있을 수 있습니다.

디바이스 수명 주기는 `CreateManagedThing` API를 사용하여 관리형 통합에서 관리형 사물을 생성하는 것으로 시작되며 고객이 `DeleteManagedThing` API를 사용하여 관리형 사물을 삭제하면 종료됩니다. 디바이스의 수명 주기는 다음 퍼블릭 APIs에 의해 관리됩니다.

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

디바이스 프로필

디바이스 프로파일은 전구 또는 도어벨과 같은 특정 유형의 디바이스를 나타냅니다. 제조업체와 연결되며 디바이스의 기능이 포함되어 있습니다. 디바이스 프로파일은 관리형 통합을 통한 디바이스 연결 설정 요청에 필요한 인증 자료를 저장합니다. 사용되는 인증 자료는 디바이스 바코드입니다.

디바이스 제조 프로세스 중에 제조업체는 디바이스 프로파일을 관리형 통합에 등록할 수 있습니다. 이를 통해 제조업체는 온보딩 및 프로비저닝 워크플로 중에 관리형 통합에서 디바이스에 필요한 재료를 얻을 수 있습니다. 디바이스 프로파일의 메타데이터는 물리적 디바이스에 저장되거나 디바이스 레이블 지정에 인쇄됩니다. 디바이스 프로파일의 수명 주기는 제조업체가 관리형 통합에서 디바이스 프로파일을 삭제할 때 종료됩니다.

데이터 모델

데이터 모델은 시스템 내에서 데이터가 구성되는 방법의 조직 계층 구조를 나타냅니다. 또한 전체 디바이스 구현에서 end-to-end 통신을 지원합니다. 관리형 통합에는 두 가지 데이터 모델이 사용됩니다. 관리형 통합 데이터 모델 및 Matter 데이터 모델의 AWS 구현입니다. 유사성이 있지만 다음 주제에 설명된 미묘한 차이도 있습니다.

타사 디바이스의 경우 두 데이터 모델 모두 최종 사용자, 관리형 통합 및 타사 클라우드 공급자 간의 통신에 사용됩니다. 두 데이터 모델의 디바이스 명령 및 디바이스 이벤트와 같은 메시지를 변환하기 위해 Cloud-to-Cloud Connector 기능이 활용됩니다.

주제

관리형 통합 데이터 모델

관리형 통합 데이터 모델은 최종 사용자와 관리형 통합 간의 모든 통신을 관리합니다.

디바이스 계층 구조

`endpoint` 및 `capability` 데이터 요소는 관리형 통합 데이터 모델의 디바이스를 설명하는 데 사용됩니다.

`endpoint`

는 기능에서 제공하는 논리적 인터페이스 또는 서비스를 `endpoint` 나타냅니다.

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

`Capability`

는 디바이스 기능을 `capability` 나타냅니다.

```
{
```

```
    "$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[]
}
```

capability 데이터 요소의 경우 해당 항목을 구성하는 세 가지 항목인 property, 및 action가 있습니다event. 디바이스와 상호 작용하고 모니터링하는 데 사용할 수 있습니다.

- 속성: 조광 가능한 조명의 현재 밝기 수준 속성과 같이 디바이스가 보유한 상태입니다.
  - ```
    {
        "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

    }
    ```
- 작업: 문 잠금 시 문 잠금과 같이 수행할 수 있는 작업입니다. 작업은 응답과 결과를 생성할 수 있습니다.
  - ```
    {
        "name": { "$ref": "/schema-versions/definition/aws.name@1.0" }, //required
        "parameters": Map<String name, JSONNode value>,
        "responseCode": HTTPResponseCode,
        "errors": {
            "code": "string",
            "message": "string"
        }
    }
    ```
- 이벤트: 기본적으로 과거 상태 전환에 대한 레코드입니다. 는 현재 상태를 property 나타내지만 이벤트는 과거의 저널이며 단조롭게 증가하는 카운터, 타임스탬프 및 우선 순위를 포함합니다. 이를 통해 상태 전환을 캡처할 수 있을 뿐만 아니라 로 쉽게 달성할 수 없는 데이터 모델링도 가능합니다property.

- ```
  {
      "name": { "$ref": "/schema-versions/definition/aws.name@1.0" },        //
  required
      "parameters": Map<String name, JSONNode value>
  }
  ```

AWS Matter 데이터 모델 구현

Matter Data Model을 AWS 구현하면 관리형 통합과 타사 클라우드 공급자 간의 모든 통신이 관리됩니다.

자세한 내용은 Matter 데이터 모델: 개발자 리소스를 참조하세요.

디바이스 계층 구조

디바이스를 설명하는 데 사용되는 데이터 요소는 `endpoint` 및 입니다`cluster`.

`endpoint`

는 기능에서 제공하는 논리적 인터페이스 또는 서비스를 `endpoint` 나타냅니다.

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

`cluster`

는 디바이스 기능을 `cluster` 나타냅니다.

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

`cluster` 데이터 요소에는 해당 항목을 구성하는 세 가지 항목인 `attribute`, 및 `command`가 있습니다`event`. 디바이스와 상호 작용하고 모니터링하는 데 사용할 수 있습니다.
```

- 속성: 조광 가능한 조명의 현재 밝기 수준 속성과 같이 디바이스가 보유한 상태입니다.
  - ```
    {
        "id" (hexadecimalString): (JsonNode) value
    }
    ```
- 명령: 문 잠금 시 문 잠금과 같이 수행할 수 있는 작업입니다. 명령은 응답과 결과를 생성할 수 있습니다.
  - ```
    "id": {
        "fieldId": "fieldValue",
        ...
        "responseCode": HTTPResponseCode,
        "errors": {
            "code": "string",
            "message": "string"
        }
    }
    ```
- 이벤트: 기본적으로 과거 상태 전환에 대한 레코드입니다. 는 현재 상태를 `attributes` 나타내지만 이벤트는 과거의 저널이며 단조롭게 증가하는 카운터, 타임스탬프 및 우선 순위를 포함합니다. 이를 통해 상태 전환을 캡처할 수 있을 뿐만 아니라 로 쉽게 달성할 수 없는 데이터 모델링도 가능합니다`attributes`.
  - ```
    "id": {
        "fieldId": "fieldValue",
        ...
    }
    ```

데이터 모델 스키마

관리형 통합은 기능 및 유형 정의라는 두 가지 스키마 유형을 지원합니다. 사용자 지정 데이터 모델을 생성하는 경우 JSON 스키마 문서를 사용하여 두 스키마 유형 중 하나를 정의합니다. 각 스키마 문서에는 50,000자 제한이 있습니다.

기능 스키마

기능은 엔드포인트 내의 특정 기능을 나타내는 기본 구성 요소입니다. 기능을 사용하면 속성, 작업 및 이벤트를 사용하여 디바이스 상태 및 동작을 모델링할 수 있습니다. 속성을 사용하면 선언적 데이터 유형으로 디바이스의 상태 속성을 유연하게 모델링할 수 있습니다. 작업 및 이벤트는 실행할 수 있는 명령과 보고할 수 있는 신호를 포함하여 디바이스의 동작을 모델링합니다.
    ```

다음은 기능 스키마의 상위 수준 구조를 표시합니다.

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

작업

디바이스의 기능과의 상호 작용을 나타내는 개체입니다. 예를 들어 벨을 울리거나 문 앞에 있는 사람을 확인합니다.

Event

디바이스의 기능에서 이벤트를 나타내는 개체입니다. 디바이스는 이벤트를 전송하여 인시던트, 경보 또는 출입구의 충격과 같은 센서 활동을 보고할 수 있습니다.

속성

디바이스 상태의 특정 속성을 나타내는 개체입니다. 예를 들어 벨이 울리거나 포치 라이트가 켜져 있습니다.

각 기능에는 고유한 네임스페이스 식별자, 버전 정보 및 용도에 대한 설명이 포함되어 있습니다. 스키마 문서는 의미 체계 버전 관리를 사용하여 이전 버전과의 호환성을 유지하면서 새로운 기능을 활성화합니다.

자세한 내용은 기능 정의 스키마 단원을 참조하십시오.

유형 정의 스키마

유형 정의는 재사용성과 구성성을 지원하는 선언적 구조화된 데이터 유형입니다. 정보의 형식을 지정하고 제한하는 방법을 정의합니다. 유형 정의를 사용하여 IoT 솔루션 전체에서 표준화된 데이터 형식을 생성합니다.

각 유형 정의에는 다음이 포함됩니다.

- 고유한 네임스페이스 식별자
- Title
- 설명
- 데이터 형식 및 제약 조건을 정의하는 속성

유형은 정수 또는 정의된 제한이 있는 문자열과 같은 단순 프리미티브이거나 열거 또는 여러 필드가 있는 사용자 지정 객체와 같은 복잡한 구조일 수 있습니다. 유형 정의는 JSON 스키마 구문을 사용하여 최소값 및 최대값, 문자열 길이, 허용 가능한 패턴을 포함한 제약 조건을 지정합니다.

자세한 내용은 유형 정의 스키마 단원을 참조하십시오.

기능 정의 스키마

기능은 시스템 내에서 기능이 작동하는 방식에 대한 명확한 계약을 제공하는 선언적 JSON 문서를 사용하여 문서화됩니다.

기능의 경우 필수 요소는 `$id`, `nameextrinsicId`, `extrinsicVersion` 및 다음 섹션 중 하나 이상에 있는 하나 이상의 요소입니다.

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

기능의 선택적 요소는 `$ref`, `title`, `description`, 및 `version` `$defs` 입니다`extrinsicProperties`. 기능의 경우는를 참조`$ref`해야 합니다`aws.capability`.

다음 섹션에서는 기능 정의에 사용되는 스키마를 자세히 설명합니다.

$id(필수)

$id 요소는 스키마 정의를 식별합니다. 다음 구조를 따라야 합니다.

- `/schema-versions/` URI 접두사로 시작
- `capability` 스키마 유형 포함
- URI 경로 구분자로 슬래시(/) 사용
- 조각을 마침표로 구분하여 스키마 자격 증명 포함(`.`)
- @ 문자를 사용하여 스키마 ID와 버전을 구분합니다.
- 마침표(`.`)를 사용하여 버전 조각을 분리하는 셈버 버전으로 끝납니다.

스키마 자격 증명은 3~12자 길이의 루트 네임스페이스로 시작한 다음 선택적 하위 네임스페이스와 이름으로 시작해야 합니다.

셈버 버전에는 메이저 버전(최대 3자리), 마이너 버전(최대 3자리) 및 선택적 패치 버전(최대 4자리)이 포함됩니다.

> **Note**
>
> 예약된 네임스페이스 `aws` 또는를 사용할 수 없습니다. `matter`

Example$id 예

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

$ref

`$ref` 요소는 시스템 내의 기존 기능을 참조합니다. `$id` 요소와 동일한 제약 조건을 따릅니다.

> **Note**
>
> 형식 정의 또는 기능은 `$ref` 파일에 제공된 값과 함께 존재해야 합니다.

Example예제 $ref

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

이름(필수)

이름 요소는 스키마 문서의 엔터티 이름을 나타내는 문자열입니다. 종종 약어가 포함되어 있으며 다음 규칙을 따라야 합니다.

- 영숫자, 마침표(.), 슬래시(/), 하이픈(-) 및 공백만 포함
- 문자로 시작
- 최대 64자

이름 요소는 Amazon Web Services 콘솔 UI 및 설명서에 사용됩니다.

Example예제 이름

```
Door Lock
```

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

제목

제목 요소는 스키마 문서로 표시되는 개체에 대한 설명 문자열입니다. 모든 문자를 포함할 수 있으며 설명서에 사용됩니다. 기능 제목의 최대 길이는 256자입니다.

Example제목 예

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

설명

description 요소는 스키마 문서로 표시되는 개체에 대한 자세한 설명을 제공합니다. 모든 문자를 포함할 수 있으며 설명서에 사용됩니다. 기능 설명의 최대 길이는 2048자입니다.

Example예제 설명

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

version

version 요소는 옵션입니다. 스키마 문서의 버전을 나타내는 문자열입니다. 다음과 같은 제약이 있습니다.

- 다음 버전 조각을 . (마침표)로 구분하여 semver 형식을 사용합니다.
  - MAJOR 버전, 최대 3자리
  - MINOR 버전, 최대 3자리
  - PATCH 버전(선택 사항), 최대 4자리
- 길이는 3~12자입니다.

Example예제 버전

```
1.0
```

```
1.12
```

```
1.4.1
```

기능 버전 작업

기능은 변경 불가능한 버전이 지정된 엔터티입니다. 모든 변경 사항은 새 버전을 생성해야 합니다. 시스템은 다음과 같은 MAJOR.MINOR.PATCH 형식의 의미 체계 버전 관리를 사용합니다.

- 이전 버전과 호환되지 않는 API 변경 시 메이저 버전 증가
- 이전 버전과 호환되는 방식으로 기능을 추가할 때 마이너 버전이 증가합니다.
- 기능에 영향을 주지 않는 사소한 추가를 수행하면 패치 버전이 증가합니다.

Matter 클러스터에서 파생된 기능은 버전 1.4에서 기준이 지정되며 각 Matter 릴리스는 시스템으로 가져올 것으로 예상됩니다. Matter 버전은 메이저 및 마이너 수준의 셈버를 모두 사용하므로 관리형 통합은 패치 버전만 사용할 수 있습니다.

Matter에 대한 패치 버전을 추가할 때는 해당 Matter가 순차 개정을 사용한다는 점을 고려해야 합니다. 모든 PATCH 버전은 Matter 사양에 설명된 개정을 준수해야 하며 이전 버전과 호환되어야 합니다.

이전 버전과 호환되지 않는 문제를 해결하려면 Connectivity Standards Alliance(CSA)와 협력하여 사양의 문제를 해결하고 새 개정을 릴리스해야 합니다.

AWS관리형 기능은의 초기 버전과 함께 릴리스되었습니다`1.0`. 이를 통해 세 가지 수준의 버전을 모두 사용할 수 있습니다.

extrinsicVersion(필수)

시스템 외부에서 관리되는 버전을 나타내는 문자열입니다 AWS IoT . Matter 기능의 경우는에 `extrinsicVersion` 매핑됩니다. `revision`

문자열화된 정수 값으로 표시되며 길이는 1~10자리 숫자일 수 있습니다.

Example예제 버전

```
7
```

```
1567
```

extrinsicId(필수)

`extrinsicId` 요소는 Amazon Web Services IoT 시스템 외부에서 관리되는 식별자를 나타냅니다. Matter 기능의 경우 컨텍스트에 따라 `clusterId`, `fieldId`, `attributeIdcommandIdeventId`, 또는에 매핑됩니다.

는 문자열화된 10진수 정수(1~10자리) 또는 문자열화된 16진수 정수(0x 또는 0X 접두사 뒤에 1~8자리 16진수)일 `extrinsicId` 수 있습니다.

> **Note**
>
> AWS의 경우 공급업체 ID(VID)는 0x1577이고 Matter의 경우 0입니다. 시스템은 사용자 지정 스키마가 이러한 예약된 VIDs 기능에 사용하지 않도록 합니다.

Example extrinsicIds 예제

```
0018
0x001A
0x15771002
```

$defs

`$defs` 섹션은 JSON 스키마에서 허용하는 스키마 문서 내에서 참조할 수 있는 하위 스키마의 맵입니다. 이 맵에서 키는 로컬 참조 정의에 사용되며 값은 JSON 스키마를 제공합니다.

> **Note**
>
> 시스템은 `$defs` 유효한 맵이고 각 하위 스키마가 유효한 JSON 스키마인 만 적용합니다. 추가 규칙은 적용되지 않습니다.

정의 작업 시 다음 제약 조건을 따릅니다.

- 정의 이름에 URI 친화적 문자만 사용
- 각 값이 유효한 하위 스키마인지 확인
- 스키마 문서 크기 제한에 맞는 하위 스키마 수를 포함합니다.

extrinsicProperties

`extrinsicProperties` 요소에는 외부 시스템에 정의되어 있지만 데이터 모델 내에서 유지되는 속성 세트가 포함되어 있습니다. Matter 기능의 경우 ZCL 클러스터, 속성, 명령 또는 이벤트 내에서 모델링되지 않았거나 부분적으로 모델링된 다양한 요소에 매핑됩니다.

외부 속성은 다음 제약 조건을 따라야 합니다.

- 속성 이름은 공백이나 특수 문자 없이 영숫자여야 합니다.
- 속성 값은 모든 JSON 스키마 값일 수 있습니다.
- 최대 20개의 속성

시스템은 `extrinsicProperties`, , `access`, 등 다양한 `apiMaturitycliclicliFunctionName`를 지원합니다. 이러한 속성은 ACL에서 데이터 모델로의 변환 AWS (및 그 반대)을 용이하게 합니다.

> **Note**
>
> 외부 속성은 기능의 `action`, `eventproperty`, 및 `struct` 필드 요소에 대해 지원되지만 기능 또는 클러스터 자체에는 지원되지 않습니다.

시스템 지원 외부 속성

시스템은 ZCL로 또는 ZCL에서 변환하는 `extrinsicProperties` 동안 다음과 같이 모델링되지 않았거나 부분적으로 모델링된 클러스터, 속성, 명령 또는 이벤트 속성을 추적합니다.

`access`

각 액세스 객체에는 다음이 포함됩니다.

- `op` - 값이 `read`, `write`또는 인 `enum` 로 모델링된 작업 `invoke`
- `privilege` - 값이 `view`, , `proxy_view`, `operate` `manage`또는 `enum`인 로 모델링된 권한 `administer`
- `role` - 연산자 역할을 나타내는 경계 없는 문자열

apiMaturity

성숙도를 나타내는 무한 일반 문자열입니다. ZCL에서 값이 stable, internal, 또는 enum인 로 모델링됩니다provisional. deprecated

side

값이 either, server및 인 열거형으로 모델링됨 client

부울 속성

다음 속성은 부울 플래그입니다.

- isFabricScoped
- isFabricSensitive
- mustUseAtomicWrite
- mustUseTimedInvoke

문자열 속성

다음 속성은 무한 문자열로 표시됩니다.

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

변환 고려 사항

ZCL 변환의 경우 `extrinsicProperties`는 처리 없이 맵에 저장됩니다. 검색을 사용하는 사용자 지정 스키마는 ZCL 변환을 거치지 않습니다. 그러나 향후 사용자 지정 스키마에 대한 ZCL 변환을 구현하려는 경우 무제한 일반 문자열 유형을 모두 모델링`extrinsicProperties`하고 열거형, 패턴(정규식) 및 길이와 같은 제약 조건을 정의해야 합니다. 이렇게 준비하면 변환 중에 이러한 속성을 적절하게 처리할 수 있습니다.

반대로 커넥터 변환 AWS 의 경우 커넥터 형식에 이러한 세부 정보가 필요하지 않으므로가 전혀 `extrinsicProperties` 포함되지 않습니다.

속성

속성은 디바이스 관리형 기능 상태를 나타냅니다. 각 상태는 키-값 페어로 정의되며, 여기서 키는 상태 이름을 설명하고 값은 상태 정의를 설명합니다.

속성을 사용할 때는 다음 제약 조건을 따르세요.

- 속성 이름에는 공백이나 특수 문자 없이 영숫자만 사용
- 스키마 문서 크기 제한에 맞는 모든 수의 속성을 포함합니다.

속성 작업

기능 내의 속성은 관리형 통합으로 구동되는 디바이스의 특정 상태를 나타내는 기본 요소입니다. 디바이스의 현재 조건 또는 구성을 나타냅니다. 스마트 홈 시스템은 이러한 속성이 정의되고 구조화되는 방식을 표준화하여 다양한 제조업체의 디바이스가 효과적으로 통신할 수 있도록 하여 원활하고 상호 운용 가능한 경험을 제공합니다.

기능 속성의 경우 필수 요소는 `extrinsicId` 및 입니다`value`. 기능 속성의 선택적 요소는 `description`, `retrievable`, 및 `mutable` `reportable` 입니다`extrinsicProperties`.

값

빌더가 JSON 스키마 준수 제약 조건을 적용하여이 속성의 데이터 유형을 정의할 수 있는 무제한 구조입니다.

값을 정의할 때 다음 제약 조건을 따릅니다.

- 간단한 유형의 경우 `type` 및 `maxLength` 또는와 같은 기타 기본 JSON 스키마 제약 조건을 사용합니다. `maximum`

- 복합 유형의 경우 , 또는 oneOfallOf를 사용합니다anyOf. 시스템에서 not 키워드를 지원하지 않습니다.
- 모든 글로벌 유형을 참조하려면 유효한 검색 가능한 참조와 $ref 함께를 사용하세요.
- null 가능성의 경우 부울 플래그와 함께 null 가능한 속성을 제공하여 OpenAPI 유형 스키마 정의를 따릅니다(truenull이 허용되는 값인 경우).

예시

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

검색 가능

상태를 읽을 수 있는지 여부를 설명하는 부울입니다.

상태의 가독성 측면은 디바이스의 기능 구현으로 연기됩니다. 디바이스는 지정된 상태를 읽을 수 있는지 여부를 결정합니다. 상태의이 측면은 아직 기능 보고서에서 보고하도록 지원되지 않으므로 시스템 내에서 적용되지 않습니다.

예: true 또는 false

Mutable

상태가 쓰기 가능한지 여부를 설명하는 부울입니다.

상태의 쓰기 가능성 측면은 디바이스의 기능 구현으로 연기됩니다. 디바이스는 지정된 상태가 쓰기 가능한지 여부를 결정합니다. 상태의이 측면은 아직 기능 보고서에서 보고하도록 지원되지 않으므로 시스템 내에서 적용되지 않습니다.

예: true 또는 false

보고 가능

상태가 변경될 때 디바이스에서 상태를 보고하는지 여부를 설명하는 부울입니다.

상태의 보고 가능성 측면은 디바이스의 기능 구현으로 연기됩니다. 디바이스는 지정된 상태가 보고 가능한지 여부를 결정합니다. 상태의이 측면은 아직 기능 보고서에서 보고하도록 지원되지 않으므로 시스템 내에서 적용되지 않습니다.

예: `true` 또는 `false`

작업

작업은 요청-응답 모델을 따르는 스키마 관리형 작업입니다. 각 작업은 디바이스 구현 작업을 나타냅니다.

작업을 구현할 때 다음 제약 조건을 따릅니다.

- 작업 배열에 고유한 작업만 포함
- 스키마 문서 크기 제한에 맞는 모든 수의 작업을 포함합니다.

작업

작업은 관리형 통합 시스템에서 디바이스 기능과 상호 작용하고 제어하기 위한 표준화된 방법입니다. 필요한 요청 또는 응답 파라미터를 모델링하기 위해 구조화된 형식으로 완성된 디바이스에서 실행할 수 있는 특정 명령 또는 작업을 나타냅니다. 이러한 작업은 사용자 의도와 디바이스 작업 간의 연결 역할을 하여 다양한 유형의 스마트 디바이스에서 일관되고 신뢰할 수 있는 제어를 가능하게 합니다.

작업의 경우 필수 요소는 `name` 및 입니다`extrinsicId`. 선택적 요소는 `description`, `extrinsicProperties` `request` 및 입니다`response`.

설명

설명의 최대 길이 제약 조건은 1536자입니다.

요청

요청 섹션은 선택 사항이며 요청 파라미터가 없는 경우 생략할 수 있습니다. 생략하면의 이름만 사용하여 페이로드 없이 요청을 보낼 수 있습니다`Action`. 이는 조명 켜기 또는 끄기와 같은 간단한 작업에 사용됩니다.

복잡한 작업에는 추가 파라미터가 필요합니다. 예를 들어 카메라 영상 스트리밍 요청에는 사용할 스트리밍 프로토콜 또는 스트림을 특정 디스플레이 디바이스로 전송할지 여부에 대한 파라미터가 포함될 수 있습니다.

작업 요청의 경우 필수 요소는 입니다`parameters`. 선택적 요소는 `description`, `extrinsicId`및 입니다`extrinsicProperties`.

요청 설명

설명은 섹션 3.5와 동일한 형식을 따르며 최대 길이는 2,048자입니다.

응답

관리형 통합에서는 SendManagedThingCommand API를 통해 전송된 모든 작업 요청에 대해 요청이 디바이스에 도달하고 비동기 응답이 다시 발생할 것으로 예상합니다. 작업 응답은이 응답의 구조를 정의합니다.

작업 요청의 경우 필수 요소는 입니다`parameters`. 선택적 요소는 `name`, `description`, `extrinsicId`, 및 `extrinsicProperties` `errors` 입니다`responseCode`.

응답 설명

설명은와 동일한 형식을 따르며 최대 길이는 2설명,048자입니다.

응답 이름

이름은 다음과 같은 추가 세부 정보와 이름(필수)함께와 동일한 형식을 따릅니다.

- 응답의 기존 이름은 작업 이름`Response`에를 추가하여 파생됩니다.
- 다른 이름을 사용하려면이 `name` 요소에 제공할 수 있습니다. 응답에 `name`이 제공된 경우이 값은 기존 이름보다 우선합니다.

오류

요청을 처리하는 동안 오류가 있는 경우 응답에 제공된 고유한 메시지의 무한 배열입니다.

제약 조건:

- 메시지 항목은 다음 필드가 있는 JSON 객체로 선언됩니다.
  - `code`: 1~64자 길이의 영숫자 및 _ (밑줄)을 포함하는 문자열
  - `message`: 무한 문자열 값

Example오류 메시지 예제

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

응답 코드

요청이 처리된 방식을 보여주는 정수 코드입니다. 시스템 내에서 균일성을 허용하려면 디바이스 코드가 HTTP 서버 응답 상태 코드 사양을 사용하여 코드를 반환하는 것이 좋습니다.

제약 조건: 100~599 범위의 정수 값입니다.

요청 또는 응답 파라미터

파라미터 섹션은 이름 및 하위 스키마 페어의 맵으로 정의됩니다. 스키마 문서에 적합할 수 있는 경우 요청 파라미터 내에 원하는 수의 파라미터를 정의할 수 있습니다.

파라미터 이름에는 영숫자만 사용할 수 있습니다. 공백 또는 기타 문자는 허용되지 않습니다.

파라미터 필드

의 필수 요소는 `extrinsicId` 및 `parameter`입니다`value`. 선택적 요소는 `description` 및 입니다`extrinsicProperties`.

설명 요소는와 동일한 형식을 따르며 최대 길이는 1설명,024자입니다.

`extrinsicId` 및 `extrinsicProperties` 재정의

`extrinsicId` 및는 이러한 추가 세부 정보와 함께 extrinsicId(필수) 및 extrinsicProperties와 동일한 형식을 `extrinsicProperties` 따릅니다.

- 요청 또는 응답에 `extrinsicId`이 제공된 경우이 값은 작업 수준에서 제공된 값보다 우선합니다. 시스템은 `extrinsicId` 먼저 요청/응답 수준을 사용해야 하며, 누락된 경우 작업 수준을 사용해야 합니다. `extrinsicId`
- 요청 또는 응답에 `extrinsicProperties`가 제공된 경우 이러한 속성은 작업 수준에서 제공된 va 값보다 우선합니다. 시스템은 작업 수준을 `extrinsicProperties` 취하고 요청/응답 수준에서 제공된 키-값 페어를 교체해야 합니다. `extrinsicProperties`

Example extrinsicId 및 extrinsicProperties 재정의 예제

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

   "extrinsicProperties": {
      "apiMaturity": "provisional",
```

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

위 예제에서 작업 요청의 유효 값은 다음과 같습니다.

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

기본 제공 작업

모든 기능에 대해 키워드 `ReadState` 및를 사용하여 사용자 지정 작업을 수행할 수 있습니다`UpdateState`. 이 두 작업 키워드는 데이터 모델에 정의된 기능의 속성에 따라 작동합니다.

ReadState

에 명령을 전송`managedThing`하여 상태 속성의 값을 읽습니다. 디바이스 상태를 강제`ReadState`로 업데이트하는 방법으로를 사용합니다.

UpdateState

일부 속성을 업데이트하는 명령을 보냅니다.

디바이스 상태 동기화를 강제 적용하면 다음 시나리오에서 유용할 수 있습니다.

1. 디바이스가 일정 기간 오프라인 상태였고 이벤트를 내보내지 않았습니다.
2. 디바이스가 방금 프로비저닝되었으며 아직 클라우드에서 유지 관리되는 상태가 없습니다.
3. 디바이스 상태가 디바이스의 실제 상태와 동기화되지 않았습니다.

ReadState 예제

SendManagedThingCommand API를 사용하여 조명이 켜져 있는지 또는 꺼져 있는지 확인합니다.

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

```
              "parameters": {
                "propertiesToRead": [ "OnOff" ]
              }
            }
          ]
        }
      ]
    }
  ]
}
```

matter.OnOff 기능에 대한 모든 상태 속성을 읽습니다.

```
{
  "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 예제

SendManagedThingCommand APIOnTime를 사용하여 조명의를 변경합니다.

```
{
```

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

이벤트

이벤트는 디바이스에서 구현한 스키마 관리형 단방향 신호입니다.

다음 제약 조건에 따라 이벤트를 구현합니다.

- 이벤트 배열에 고유한 이벤트만 포함
- 스키마 문서 크기 제한에 맞는 이벤트 수를 포함합니다.

관리형 통합 시스템의 이벤트

이벤트 작업

이벤트는 디바이스 또는 주변 환경의 변경 사항에 대해 사전에 알아볼 수 있는 표준화된 방법입니다. 이는 디바이스에서 수정되었거나 환경에서 감지된 항목에 대한 정보를 제공하기 위해 디바이스가 클라우드로 전송하는 모델링된 이벤트를 나타냅니다. 이러한 이벤트는 모델링되므로 고객은 제어 흐름에서 이를 사용하여 특정 이벤트와 그 안에 제공된 세부 정보에 대응할 수 있습니다.

이벤트의 경우 필수 요소는 `name` 및 입니다`extrinsicId`. 선택적 요소는 `description`, `extrinsicProperties`및 입니다`request`.

설명

설명은에 설명된 것과 동일한 형식을 따르설명며 최대 길이는 512자입니다.

요청

`request` 섹션은 선택 사항이며 요청 파라미터가 없는 경우 생략할 수 있습니다. 생략하면 시스템은 이벤트 이름만 사용하여 페이로드 없이 이벤트 요청을 보내는 디바이스를 지원합니다. 이는 펌프의 센서 장애와 같은 간단한 이벤트 또는 연기 또는 일산탄소 경보에서 경보가 음소거되는 경우에 사용됩니다.

복잡한 작업에는 추가 파라미터가 필요합니다. 예를 들어 카메라 영상 스트리밍 요청에는 사용할 스트리밍 프로토콜 또는 스트림을 특정 디스플레이 디바이스로 전송할지 여부에 대한 파라미터가 포함될 수 있습니다.

이벤트 요청의 경우 필수 요소는 입니다`parameters`. 선택적 요소는 없습니다.

응답

이벤트 응답은 현재 지원되지 않습니다.

유형 정의 스키마

다음 섹션에서는 유형 정의에 사용되는 스키마를 자세히 설명합니다.

$id

$id 요소는 스키마 정의를 식별합니다. 다음 구조를 따라야 합니다.

- `/schema-versions/` URI 접두사로 시작
- `definition` 스키마 유형 포함
- URI 경로 구분자로 슬래시(/) 사용
- 조각을 마침표(.)로 구분하여 스키마 자격 증명 포함
- @ 문자를 사용하여 스키마 ID와 버전을 구분합니다.
- 마침표(.)를 사용하여 버전 조각을 구분하는 셈버 버전으로 끝납니다.

스키마 자격 증명은 3~12자 길이의 루트 네임스페이스로 시작한 다음 선택적 하위 네임스페이스와 이름으로 시작해야 합니다.

셈버 버전에는 메이저 버전(최대 3자리), 마이너 버전(최대 3자리) 및 선택적 패치 버전(최대 4자리)이 포함됩니다.

> **Note**
>
> 예약된 네임스페이스 `aws` 또는를 사용할 수 없습니다. `matter`

Example$id 예제

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

$ref

$ref 요소는 시스템 내의 기존 유형 정의를 참조합니다. `$id` 요소와 동일한 제약 조건을 따릅니다.

> **Note**
>
> 형식 정의 또는 기능은 `$ref` 파일에 제공된 값과 함께 존재해야 합니다.

Example$ref 예제

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

name

이름 요소는 스키마 문서의 엔터티 이름을 나타내는 문자열입니다. 종종 약어가 포함되어 있으며 다음 규칙을 따라야 합니다.

- 영숫자, 마침표(.), 슬래시(/), 하이픈(-) 및 공백만 포함
- 문자로 시작
- 최대 192자

이름 요소는 Amazon Web Services 콘솔 UI 및 설명서에 사용됩니다.

Example예제 이름

```
Door Lock
```

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

제목

제목 요소는 스키마 문서로 표시되는 개체에 대한 설명 문자열입니다. 모든 문자를 포함할 수 있으며 설명서에 사용됩니다.

Example제목 예

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

설명

`description` 요소는 스키마 문서로 표시되는 개체에 대한 자세한 설명을 제공합니다. 모든 문자를 포함할 수 있으며 설명서에 사용됩니다.

Example예제 설명

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

extrinsicId

`extrinsicId` 요소는 Amazon Web Services IoT 시스템 외부에서 관리되는 식별자를 나타냅니다. Matter 기능의 경우 컨텍스트에 따라 `clusterId`, `fieldId`, `attributeIdcommandIdeventId`, 또는에 매핑됩니다.

는 문자열화된 10진수 정수(1~10자리) 또는 문자열화된 16진수 정수(0x 또는 0X 접두사 뒤에 1~8자리 16진수)일 `extrinsicId` 수 있습니다.

> **Note**
>
> AWS의 경우 공급업체 ID(VID)는 0x1577이고 Matter의 경우 0입니다. 시스템은 사용자 지정 스키마가 이러한 예약된 VIDs 기능에 사용하지 않도록 합니다.

Example extrinsicIds 예제

```
0018
0x001A
0x15771002
```

extrinsicProperties

`extrinsicProperties` 요소에는 외부 시스템에 정의되어 있지만 데이터 모델 내에서 유지되는 속성 세트가 포함되어 있습니다. Matter 기능의 경우 ZCL 클러스터, 속성, 명령 또는 이벤트 내에서 모델링되지 않았거나 부분적으로 모델링된 다양한 요소에 매핑됩니다.

외부 속성은 다음 제약 조건을 따라야 합니다.

- 속성 이름은 공백이나 특수 문자 없이 영숫자여야 합니다.
- 속성 값은 모든 JSON 스키마 값일 수 있습니다.
- 최대 20개의 속성

시스템은 `extrinsicProperties`, , `access`, 등 다양한 `apiMaturitycliciliFunctionName`를 지원합니다. 이러한 속성은 ACL에서 데이터 모델로의 변환 AWS (및 그 반대)을 용이하게 합니다.

> **Note**
>
> 외부 속성은 기능의 `action`, `eventproperty`, 및 `struct` 필드 요소에 대해 지원되지만 기능 또는 클러스터 자체에는 지원되지 않습니다.

시스템 지원 외부 속성

시스템은 ZCL로 또는 ZCL에서 변환하는 `extrinsicProperties` 동안 다음과 같이 모델링되지 않았거나 부분적으로 모델링된 클러스터, 속성, 명령 또는 이벤트 속성을 추적합니다.

`access`

각 액세스 객체에는 다음이 포함됩니다.

- `op` - 값이 `read`, `write`또는 `enum`인 로 모델링된 작업 `invoke`
- `privilege` - 값이 `view`, , `proxy_view`, `operate` `manage`또는 `enum`인 로 모델링된 권한 `administer`

- role - 연산자 역할을 나타내는 경계 없는 문자열

apiMaturity

성숙도를 나타내는 무한 일반 문자열입니다. ZCL에서 값이 stable, internal, 또는 enum인 로 모델링됩니다provisional. deprecated

side

값이 either, server및 인 열거형으로 모델링됨 client

부울 속성

다음 속성은 부울 플래그입니다.

- isFabricScoped
- isFabricSensitive
- mustUseAtomicWrite
- mustUseTimedInvoke

문자열 속성

다음 속성은 무한 문자열로 표시됩니다.

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

- `restriction`
- `storage`

변환 고려 사항

ZCL 변환의 경우 `extrinsicProperties`는 처리 없이 맵에 저장됩니다. 검색을 사용하는 사용자 지정 스키마는 ZCL 변환을 거치지 않습니다. 그러나 향후 사용자 지정 스키마에 대한 ZCL 변환을 구현하려는 경우 무제한 일반 문자열 유형을 모두 모델링`extrinsicProperties`하고 열거형, 패턴(정규식) 및 길이와 같은 제약 조건을 정의해야 합니다. 이렇게 준비하면 변환 중에 이러한 속성을 적절하게 처리할 수 있습니다.

반대로 커넥터 변환 AWS 의 경우 커넥터 형식에 이러한 세부 정보가 필요하지 않으므로가 전혀 `extrinsicProperties` 포함되지 않습니다.

기능 스키마 문서에서 유형 정의 빌드 및 사용

스키마의 모든 요소는 정의 유형으로 확인됩니다. 이러한 유형 정의는 기본 유형 정의(예: 부울, 문자열, 숫자) 또는 네임스페이스 유형 정의(편의를 위해 기본 유형 정의로 구축된 유형 정의)입니다.

사용자 지정 스키마를 정의할 때 기본 정의와 네임스페이스 유형 정의를 모두 사용할 수 있습니다.

목차

기본 유형 정의

기본 유형 정의는 관리형 통합에 정의된 모든 유형 정의의 구성 요소입니다. 사용자 지정 유형 정의를 포함한 모든 네임스페이스 정의는 `$ref` 키워드 또는 `type` 키워드를 통해 기본 유형 정의로 해석됩니다.

모든 프리미티브 유형은 `nullable` 키워드를 사용하여 null이 가능하며 키워드를 사용하여 모든 프리미티브 유형을 식별할 수 있습니다`type`.

부울

부울 유형은 기본값을 지원합니다.

샘플 정의:

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

정수 유형 지원

정수 유형은 다음을 지원합니다.

- `default` 값
- `maximum` 값
- `minimum` 값
- `exclusiveMaximum` 값
- `exclusiveMinimum` 값
- `multipleOf` 값

x가 검증 중인 값인 경우 다음이 true여야 합니다.

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

> **Note**
>
> 소수 부분이 0인 숫자는 정수로 간주되지만 부동 소수점 숫자는 거부됩니다.
>
> ```
> 1.0 // Schema-Compliant
> 3.1415926 // NOT Schema-Compliant
> ```

`minimum` 및를 모두 지정`exclusiveMinimum`하거나 `maximum` 및를 모두 지정할 수 있지만 두 가지를 동시에 사용하지 않는 `exclusiveMaximum`것이 좋습니다.

샘플 정의:

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

대체 정의:

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

숫자

정수 및 부동 소수점 숫자를 포함하여 모든 숫자 유형에 숫자 유형을 사용합니다.

숫자 유형은 다음을 지원합니다.

- `default` 값
- `maximum` 값

- `minimum` 값
- `exclusiveMaximum` 값
- `exclusiveMinimum` 값
- `multipleOf` 값입니다. 배수는 부동 소수점 숫자일 수 있습니다.

x가 검증 중인 값인 경우 다음이 true여야 합니다.

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

`minimum` 및를 모두 지정`exclusiveMinimum`하거나 `maximum` 및를 모두 지정할 수 있지만 두 가지를 동시에 사용하지 않는 `exclusiveMaximum`것이 좋습니다.

샘플 정의:

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

대체 정의:

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

문자열

문자열 유형은 다음을 지원합니다.

- `default` 값
- `maxLength` 및 `minLength` 값을 포함한 길이 제약 조건(음수가 아닌 숫자여야 함)
- `pattern` 정규식 값

정규식을 정의하면 표현식이 문자열 내 어느 곳에서든 일치하는 경우 문자열이 유효합니다. 예를 들어 정규식은 문자열 "p"뿐만 아니라 "apple"과 같은 p를 포함하는 모든 문자열과 p 일치합니다. 명확성을 위해 특정 이유가 없는 한 `^...$` (예: ^p$)를 사용하여 정규 표현식을 둘러싸는 것이 좋습니다.

샘플 정의:

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

NULL

Null 유형은 단일 값 만 허용합니다`null`.

샘플 정의:

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

배열

배열 유형은 다음을 지원합니다.

- `default` - 기본값으로 사용할 목록입니다.
- `items` - 모든 배열 요소에 적용되는 JSON 유형 정의입니다.
- 길이 제약 조건(음수가 아닌 숫자여야 함)
  - `minItems`
  - `maxItems`
- `pattern` 정규식 값
- `uniqueItems` - 배열의 요소가 고유해야 하는지 여부를 나타내는 부울

- `prefixItems` - 각 항목이 문서 배열의 각 인덱스에 해당하는 스키마인 배열입니다. 즉, 첫 번째 요소가 입력 배열의 첫 번째 요소를 검증하고 두 번째 요소가 입력 배열의 두 번째 요소를 검증하는 배열입니다.

샘플 정의:

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

배열 검증의 예:

```
//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.
```

튜플 검증을 사용한 대체 정의:

```
{
    "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
```

Objects

객체 유형은 다음을 지원합니다.

- 속성 제약 조건
  - properties - properties키워드를 사용하여 객체의 속성(키-값 페어)을 정의합니다. 의 값은 객체properties입니다. 여기서 각 키는 속성의 이름이고 각 값은 해당 속성을 검증하는 데 사용되는 스키마입니다. properties 키워드의 속성 이름과 일치하지 않는 속성은이 키워드에서 무시됩니다.
  - required - 기본적으로 properties 키워드로 정의된 속성은 필요하지 않습니다. 그러나 required 키워드를 사용하여 필요한 속성 목록을 제공할 수 있습니다. required 키워드는 0개 이상의 문자열 배열을 사용합니다. 이러한 각 문자열은 고유해야 합니다.
  - propertyNames -이 키워드를 사용하면 속성 이름에 대한 RegEx 패턴을 제어할 수 있습니다. 예를 들어 객체의 모든 속성이 특정 규칙에 따라 이름을 갖도록 적용할 수 있습니다.
  - patternProperties - 정규식을 스키마에 매핑합니다. 속성 이름이 지정된 정규식과 일치하는 경우 속성 값은 해당 스키마에 대해 검증되어야 합니다. 예를 들어, patternProperties를 사용하여 특정 종류의 속성 이름을 지정하면 값이 특정 스키마와 일치해야 합니다.
  - additionalProperties -이 키워드는 추가 속성 처리 방법을 제어합니다. 추가 속성은 속성 키워드에 이름이 나열되지 않거나의 정규식과 일치하는 속성입니다patternProperties. 기본적으로 추가 속성이 허용됩니다. 이 필드를 로 설정false하면 추가 속성이 허용되지 않습니다.
  - unevaluatedProperties -이 키워드는 하위 스키마에 선언된 속성을 인식할 수 있다는 점을 additionalProperties 제외하면와 유사합니다.는 스키마를 처리할 때 성공적으로 검증되고 허용된 속성 목록으로 사용하는 속성을 수집하여 unevaluatedProperties 작동합니다. 이렇게 하면 조건부로 속성을 추가하는 등 더 복잡한 작업을 수행할 수 있습니다. 자세한 내용은 아래 예제를 참조하세요.
- anyOf -이 키워드의 값은 비어 있지 않은 배열이어야 합니다. 배열의 각 항목은 유효한 JSON 스키마여야 합니다. 인스턴스는이 키워드의 값으로 정의된 하나 이상의 스키마에 대해 성공적으로 검증되는 경우이 키워드에 대해 성공적으로 검증됩니다.
- oneOf -이 키워드의 값은 비어 있지 않은 배열이어야 합니다. 배열의 각 항목은 유효한 JSON 스키마여야 합니다. 인스턴스는이 키워드의 값으로 정의된 정확히 하나의 스키마에 대해 성공적으로 검증되는 경우이 키워드에 대해 성공적으로 검증됩니다.

필수 예:

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

// Schema Compliant
{
  "test": 4
}

// NOT Schema Compliant
{}
```

PropertyNames 예제:

```
{
  "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 예제:

```
{
  "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 예제:

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

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

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

UnevaluatedProperties 예제:

```
{
  "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 예제:

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

// Schema Compliant
"short"
12

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

OneOf 예제:

```
{
  "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.
```

네임스페이스 형식 정의

네임스페이스 유형 정의는 기본 유형으로 빌드된 유형입니다. 이러한 유형은 *namespace*.*typename*. 관리형 통합이 aws 및 matter 네임스페이스 아래에 사전 정의된 유형을 제공하는 형식을 따라야 합니다. 예약 및 네임스페이스를 제외한 모든 aws matter 네임스페이스를 사용자 지정 유형에 사용할 수 있습니다.

사용 가능한 네임스페이스 유형 정의를 찾으려면 Type 필터가 로 설정된 ListSchemaVersions API를 사용합니다definition.

matter 유형

필터가 로 설정matter되고 Namespace 필터가 Type 로 설정된 ListSchemaVersions API를 사용하여 matter 네임스페이스에서 데이터 유형을 찾습니다definition.

aws 유형

필터가 로 설정aws되고 Namespace 필터가 Type 로 설정된 ListSchemaVersions API를 사용하여 aws 네임스페이스에서 데이터 유형을 찾습니다definition.

비트맵 유형 정의

비트맵에는 두 가지 필수 속성이 있습니다.

- type 객체여야 합니다.
- properties는 각 비트 정의를 포함하는 객체여야 합니다. 각 비트는 속성이 extrinsicId 및 인 객체입니다value. 각 비트의 값은 최소값이 0이고 최대값이 1 이상인 정수여야 합니다.

샘플 비트맵 정의:

```
{
    "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
}
```

열거형 유형 정의

열거형에는 세 가지 속성이 필요합니다.

- `type` 객체여야 합니다.

- enum는 최소 하나의 항목이 있는 고유한 문자열의 배열이어야 합니다.
- extrinsicIdMap는 열거형 값인 속성을 가진 객체입니다. 각 속성의 값은 열거형 값에 해당하는 외부 식별자여야 합니다.

샘플 열거형 정의:

```
{
    "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"
```

IoT 디바이스 명령 및 이벤트 관리

디바이스 명령은 물리적 디바이스를 원격으로 관리하여 중요한 보안, 소프트웨어 및 하드웨어 업데이트를 수행하는 것 외에도 디바이스를 완벽하게 제어할 수 있는 기능을 제공합니다. 대규모 디바이스 플릿을 사용하면 디바이스가 언제 명령을 수행하는지 알면 전체 디바이스 구현을 감독할 수 있습니다. 디바이스 명령 또는 자동 업데이트는 디바이스 상태 변경을 트리거하여 새 디바이스 이벤트를 생성합니다. 이 디바이스 이벤트는 고객 관리형 대상으로 자동으로 전송되는 알림을 트리거합니다.

주제

디바이스 명령

명령 요청은 디바이스로 전송되는 명령입니다. 명령 요청에는 전구 켜기와 같이 수행할 작업을 지정하는 페이로드가 포함됩니다. 디바이스 명령을 전송하기 위해 관리형 통합을 통해 최종 사용자를 대신하여 SendManagedThingCommand API가 호출되고 명령 요청이 디바이스로 전송됩니다.

에 대한 응답은 SendManagedThingCommand이며 가능한 경우 이를 사용하여 명령 전송 및 관련 명령 응답 워크플로를 traceId 추적할 traceId 수 있습니다.

SendManagedThingCommand API 작업에 대한 자세한 내용은 SendManagedThingCommand를 참조하세요.

UpdateState 작업

조명이 켜지는 시간과 같은 디바이스 상태를 업데이트하려면 SendManagedThingCommand API를 호출할 때 UpdateState 작업을 사용합니다. 에서 업데이트하려는 데이터 모델 속성과 새 값을 제공합니다parameters. 아래 예제는 전구OnTime의를 로 업데이트하는 SendManagedThingCommand API 요청을 보여줍니다5.

```
{
  "Endpoints": [
    {
      "endpointId": "1",
      "capabilities": [
        {
          "id": "matter.OnOff",
```

```
          "name": "On/Off",
          "version": "1",
          "actions": [
            {
              "name": "UpdateState",
              "parameters": {
                "OnTime": 5
              }
            }
          ]
        }
      ]
    }
  ]
}
```

ReadState 작업

모든 데이터 모델 속성의 현재 값을 포함하여 디바이스의 최신 상태를 가져오려면 SendManagedThingCommand API를 호출할 때 ReadState 작업을 사용합니다. 에서는 다음 옵션을 사용할 propertiesToRead수 있습니다.

- 조명이 켜져 있는지 또는 꺼져 있는지 OnOff 확인하는 등 최신 값을 얻기 위해 특정 데이터 모델 속성을 제공합니다.
- 와일드카드 연산자(*)를 사용하여 기능에 대한 모든 디바이스 상태 속성을 읽습니다.

아래 예제에서는 ReadState 작업을 사용하는 SendManagedThingCommand API 요청에 대한 두 시나리오를 모두 보여줍니다.

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

디바이스 이벤트

디바이스 이벤트에는 디바이스의 현재 상태가 포함됩니다. 이는 디바이스가 상태를 변경했거나 상태가 변경되지 않은 경우에도 상태를 보고하고 있음을 의미할 수 있습니다. 여기에는 데이터 모델에 정의된 속성 보고서 및 이벤트가 포함됩니다. 휴지통 주기가 완료되었거나 온도 조절기가 최종 사용자가 설정한 목표 온도에 도달한 이벤트일 수 있습니다.

디바이스 이벤트 알림

최종 사용자는 특정 디바이스 이벤트에 대한 업데이트를 위해 생성한 특정 고객 관리 대상을 구독할 수 있습니다. 고객 관리형 대상을 생성하려면 `CreateDestination` API를 호출합니다. 디바이스에서 디바이스 이벤트를 관리형 통합에 보고하면 고객 관리형 대상이 있는 경우 알림을 받습니다.

관리형 통합 리소스에 태그 지정

리소스를 관리하고 구성하는 데 도움이 되도록 필요에 따라 이러한 각 리소스에 고유한 메타데이터를 태그 형태로 할당할 수 있습니다. 이 단원에서는 태그를 설명하고 태그를 생성하는 방법을 보여 줍니다.

태그 기본 사항

태그를 사용하여 관리형 통합 리소스를 다양한 방법(예: 용도, 소유자 또는 환경)으로 분류할 수 있습니다. 이 기능은 지정한 태그에 따라 리소스를 빠르게 식별할 수 있으므로 동일한 유형의 리소스가 많을 때 유용합니다. 각 태그는 사용자가 정의하는 키와 선택적 값으로 구성됩니다. 예를 들어, 사물 유형에 대한 태그 세트를 정의하여 디바이스를 유형별로 추적할 수 있습니다. 각 리소스 유형에 대한 요건을 충족하는 태그 키 세트를 생성하는 것이 좋습니다. 일관된 태그 키 세트를 사용하면 리소스를 보다 쉽게 관리할 수 있습니다.

추가하거나 적용한 태그를 기준으로 리소스를 검색하고 필터링할 수 있습니다. IAM 정책에 태그 사용의 설명처럼 리소스에 대한 액세스 제어에 태그를 사용할 수도 있습니다.

사용 편의성을 위해 AWS Management Console의 Tag Editor는 태그를 생성하고 관리하는 중앙 통합 방법을 제공합니다. 자세한 내용은 관리 콘솔 작업의 태그 편집기 작업을 참조하세요. AWS

AWS CLI 및 관리형 통합 API를 사용하여 태그로 작업할 수도 있습니다. 다음 명령의 `Tags` 필드를 사용하여 태그를 생성할 때 관리형 사물, 프로비저닝 프로필, 자격 증명 로커 및 over-the-air(OTA) 작업과 태그를 연결할 수 있습니다.

- CreateManagedThing
- CreateProvisioningProfile
- CreateCredentialLocker
- CreateOtaTask
- CreateAccountAssociation

다음 명령을 사용하여 태깅을 지원하는 기존 리소스에 대해 태그를 추가, 수정, 삭제할 수 있습니다.

- TagResource
- ListTagsForResource

- UntagResource

태그 키와 값을 편집할 수 있으며 언제든지 리소스에서 태그를 제거할 수 있습니다. 태그의 값을 빈 문자열로 설정할 수 있지만 태그의 값을 Null로 설정할 수는 없습니다. 해당 리소스의 기존 태그와 키가 동일한 태그를 추가하면 새 값이 이전 값을 덮어씁니다. 리소스를 삭제하면, 리소스에 대한 연결이 완료된 태그 또한 삭제됩니다.

태그 규제 및 제한

태그에 적용되는 기본 제한 사항은 다음과 같습니다.

- 리소스 당 최대 태그 수 - 50개
- 최대 키 길이 - UTF-8의 유니코드 문자 127자
- 최대 값 길이 – UTF8의 유니코드 문자 255자
- 태그 키와 값은 대소문자를 구분합니다.
- 태그 이름이나 값에서 접두사 `aws:`는 사용하지 마세요. 이는 AWS 사용을 위해 예약되어 있습니다. 이 접두사가 지정된 태그 이름이나 값은 편집하거나 삭제할 수 없습니다. 이 접두사가 지정된 태그는 리소스당 태그 수 제한에 포함되지 않습니다.
- 태깅 스키마를 여러 서비스와 리소스에서 사용하는 경우 다른 서비스에서는 허용되는 문자에 제한이 있을 수 있다는 점에 주의하세요. 허용되는 문자는 UTF-8로 표현할 수 있는 문자, 공백 및 숫자와 + - = . _ : / @ 등의 특수 문자입니다.

IAM 정책에 태그 사용

관리형 통합 API 작업에 사용하는 IAM 정책에서 태그 기반 리소스 수준 권한을 적용할 수 있습니다. 이를 통해 사용자가 생성, 수정 또는 사용할 수 있는 리소스를 더욱 정확하게 제어할 수 있습니다. 리소스 태그를 기반으로 사용자 액세스(권한)를 제어하기 위해 IAM 정책에서 다음 조건 콘텍스트 키 및 값과 함께 `Condition` 요소(`Condition` 블록)를 사용합니다.

- `aws:ResourceTag/`*`tag-key`*`: `*`tag-value`*를 사용하여 특정 태그가 지정된 리소스에 대한 사용자 작업을 허용 또는 거부합니다.
- `aws:RequestTag/`*`tag-key`*`: `*`tag-value`*를 사용하여 태그를 허용하는 리소스를 생성하거나 수정하는 API 요청을 작성할 때 특정 태그를 사용하도록(또는 사용하지 않도록) 요구합니다.
- `aws:TagKeys: [`*`tag-key`*`, ...]`를 사용하여 태깅 가능한 리소스를 생성하거나 수정하는 API 요청을 작성할 때 특정 태그 키 집합을 사용하도록(또는 사용하지 않도록) 요구합니다.

> **Note**
>
> IAM 정책의 조건 컨텍스트 키 및 값은 태그를 지정할 수 있는 리소스의 식별자가 필수 파라미터인 관리형 통합 작업에만 적용됩니다. 예를 들어 GetCustomEndpoint 사용은 조건 컨텍스트 키 및 값을 기반으로 허용되거나 거부되지 않습니다.이 요청에서는 태그 지정 가능한 리소스(관리형 사물, 프로비저닝 프로필, 자격 증명 잠금 장치, over-the-air 작업)가 참조되지 않기 때문입니다. 태그 지정이 가능한 관리형 통합 리소스 및 지원하는 조건 키에 대한 자세한 내용은 의 AWS IoT 관리형 통합 기능을 위한 작업, 리소스 및 조건 키를 참조하세요 AWS IoT Device Management.

자세한 내용은 AWS Identity and Access Management 사용 설명서의 태그를 사용한 액세스 제어를 참조하세요. 이 설명서의 IAM JSON 정책 참조 단원에서는 IAM에서 JSON 정책의 자세한 구문과 설명, 요소의 예, 변수 및 평가 로직을 설명합니다.

다음 예제 정책은 `CreateManagedThing` 작업에 두 가지 태그 기반 제한을 적용합니다. 이 정책으로 제한되는 IAM 사용자는 다음과 같습니다.

- 태그가 "env=prod"인 관리형 사물을 생성할 수 없습니다(이 예에서는 행 참조`"aws:RequestTag/env" : "prod"`).
- 기존 태그가 "env=prod"인 관리형 사물을 수정하거나 액세스할 수 없습니다(이 예에서는 행 참조`"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": "*"
    }
  ]
}
```

또한 다음과 같이 목록에서 태그를 둘러싸 지정된 태그 키에 대해 여러 태그 값을 지정할 수도 있습니다.

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

> **Note**
>
> 태그를 기준으로 리소스에 대한 사용자 액세스를 허용 또는 거부하는 경우 동일한 리소스에서 태그를 추가 또는 제거할 수 있도록 사용자를 명시적으로 거부할 것을 고려해야 합니다. 그렇

지 않으면 사용자가 제한을 피해 태그를 수정하여 리소스에 대한 액세스 권한을 얻을 수 있습니다.

관리형 통합 알림

관리형 통합 알림은 디바이스에서 업데이트 및 주요 인사이트를 제공합니다. 알림에는 커넥터 이벤트, 디바이스 명령, 수명 주기 이벤트, OTA(Over-the-Air) 업데이트 및 오류 보고서가 포함됩니다. 이러한 인사이트는 자동화된 워크플로를 생성하거나, 즉각적인 조치를 취하거나, 문제 해결을 위해 이벤트 데이터를 저장할 수 있는 실행 가능한 정보를 제공합니다.

현재 Amazon Kinesis 데이터 스트림만 관리형 통합 알림의 대상으로 지원됩니다. 알림을 설정하기 전에 먼저 Amazon Kinesis 데이터 스트림을 설정하고 데이터 스트림에 대한 관리형 통합 액세스를 허용해야 합니다.

알림을 위한 Amazon Kinesis 설정

Amazon Kinesis 설정 단계

- 1단계: Amazon Kinesis 데이터 스트림 생성
- 2단계: 권한 정책 생성
- 3단계: IAM 대시보드로 이동하여 역할을 선택합니다.
- 4단계: 사용자 지정 신뢰 정책 사용
- 5단계: 권한 정책 적용
- 6단계: 역할 이름 입력

관리형 통합 알림을 위해 Amazon Kinesis를 설정하려면 다음 단계를 따르세요.

1단계: Amazon Kinesis 데이터 스트림 생성

Amazon Kinesis Data Stream은 대량의 데이터를 실시간으로 수집하고, 데이터를 내구성 있게 저장하고, 애플리케이션에서 데이터를 사용할 수 있도록 할 수 있습니다.

Amazon Kinesis 데이터 스트림을 생성하려면

- Kinesis 데이터 스트림을 생성하려면 Kinesis 데이터 스트림 생성 및 관리에 설명된 단계를 따릅니다.

2단계: 권한 정책 생성

관리형 통합이 Kinesis 데이터 스트림에 액세스할 수 있도록 허용하는 권한 정책을 생성합니다.

권한 정책 생성을 생성하려는 경우

- 권한 정책을 생성하려면 아래 정책을 복사하고 JSON 편집기를 사용하여 정책 생성에 설명된 단계를 따릅니다.

  JSON

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

3단계: IAM 대시보드로 이동하여 역할을 선택합니다.

IAM 대시보드를 열고 역할을 클릭합니다.

IAM 대시보드로 이동하려면

- IAM 대시보드를 열고 역할을 클릭합니다.

  자세한 내용은 AWS Identity and Access Management 사용 설명서의 IAM 역할 생성을 참조하세요.

4단계: 사용자 지정 신뢰 정책 사용

사용자 지정 신뢰 정책을 사용하여 관리형 통합에 Kinesis 데이터 스트림에 대한 액세스 권한을 부여할 수 있습니다.

사용자 지정 신뢰 정책을 사용하려면

- 새 역할을 생성하고 사용자 지정 신뢰 정책을 선택합니다. 다음을 클릭합니다.

다음 정책은 관리형 통합이 역할을 수임하도록 허용하며, Condition 문은 혼동된 대리자 문제를 방지하는 데 도움이 됩니다.

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:*"
                }
            }
        }
    ]
}
```

5단계: 권한 정책 적용

2단계에서 생성한 권한 정책을 역할에 추가합니다.

권한 정책을 추가하려면

- 권한 추가 페이지에서 2단계에서 생성한 권한 정책을 검색하고 추가합니다. 다음을 클릭합니다.

6단계: 역할 이름 입력

- 역할 이름을 입력하고 역할 생성을 클릭합니다.

관리형 통합 알림 설정

알림 설정 단계

관리형 통합 알림을 설정하려면 다음 단계를 따르세요.

1단계: 사용자에게 CreateDestination API를 호출할 수 있는 권한 부여

- 사용자에게 **`CreateDestination`** API를 호출할 수 있는 권한 부여

  다음 정책은 사용자가 CreateDestination API를 호출하기 위한 요구 사항을 정의합니다.

  관리형 통합에 대한 패스롤 권한을 얻으려면 사용 설명서의 AWS 서비스에 역할을 전달할 수 있는 AWS Identity and Access Management 사용자 권한 부여를 참조하세요.

  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":"*"
```

```
        }
    ]
}
```

2단계: CreateDestination API 호출

- **`CreateDestination`** API 호출

  Amazon Kinesis 데이터 스트림 및 스트림 액세스 역할을 생성한 후 CreateDestination API를 호출하여 알림이 라우팅될 알림 대상을 생성합니다. `DeliveryDestinationArn` 파라미터의 경우 새 Amazon Kinesis 데이터 스트림`arn`의를 사용합니다.

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

> **Note**
>
> `ClientToken`는 멱등성 토큰입니다. 처음에 동일한 클라이언트 토큰 및 파라미터를 사용하여 성공적으로 완료된 요청을 재시도하면 추가 작업을 수행하지 않고 재시도에 성공합니다.

3단계: CreateNotificationConfiguration API 호출

- **`CreateNotificationConfiguration`** API 호출

  마지막으로 CreateNotificationConfiguration API를 사용하여 선택한 이벤트 유형을 Kinesis 데이터 스트림으로 표시되는 대상으로 라우팅하는 알림 구성을 생성합니다. `DestinationName` 파라미터에서 `CreateDestination` API를 처음 호출할 때와 동일한 대상 이름을 사용합니다.

```
{
    "EventType": "DEVICE_EVENT"
```

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

관리형 통합으로 모니터링되는 이벤트 유형

다음은 관리형 통합 알림으로 모니터링되는 이벤트 유형입니다.

- DEVICE_COMMAND
  - SendManagedThingCommand API 명령의 상태입니다. 유효한 값은 succeeded 또는 failed입니다.

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

- DEVICE_COMMAND_REQUEST
  - Web Real-Time Communication(WebRTC)의 명령 요청입니다.

    WebRTC 표준은 두 피어 간의 통신을 허용합니다. 이러한 피어는 실시간 비디오, 오디오 및 임의 데이터를 전송할 수 있습니다. 관리형 통합은 WebRTC를 지원하여 고객 모바일 애플리케이션과 최종 사용자의 디바이스 간에 이러한 유형의 스트리밍을 활성화합니다. WebRTC 표준에 대한 자세한 내용은 WebRTC를 참조하세요.

```
{
              "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
  - 디바이스의 검색 상태입니다.

```
{
      "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
  - 디바이스 이벤트 발생에 대한 알림입니다.

```
{
      "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
  - 디바이스 수명 주기의 상태입니다.

```
{
      "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
  - 디바이스 OTA 알림입니다.
- DEVICE_STATE
  - 디바이스 상태가 업데이트되었을 때의 알림입니다.

```
{
      "messageType": "DEVICE_STATE",
      "source": "aws.iotmanagedintegrations",
      "customerAccountId": "123456789012",
      "timestamp": "1731623291671",
      "resources": [
        "arn:aws:iotmanagedintegrations:ca-central-1:123456789012:managed-
thing/61889008880012345678"
      ],
      "payload": {
        "addedStates": {
          "endpoints": [{
            "endpointId": "nonEndpointId",
            "capabilities": [{
              "id": "aws.OnOff",
              "name": "On/Off",
              "version": "1.0",
              "properties": [{
                "name": "OnOff",
                "value": {
```

```
                  "propertyValue": "\"onoff\"",
                  "lastChangedAt": "2024-06-11T01:38:09.000414Z"
                }
              }
            ]}
          ]}
        ]}
      }
}
```

Cloud-to-Cloud(C2C) 커넥터

cloud-to-cloud 커넥터를 사용하면 타사 디바이스와 간의 양방향 통신을 생성하고 용이하게 할 수 있습니다 AWS.

주제

cloud-to-cloud(C2C) 커넥터란 무엇입니까?

cloud-to-cloud 커넥터는를 타사 클라우드 공급자의 엔드포인트 AWS 클라우드 에 안전하게 연결하는 사전 구축된 소프트웨어 패키지입니다. 솔루션 공급자는 C2C 커넥터를 사용하여 AWS IoT Device Management의 관리형 통합을 활용하여 타사 클라우드에 연결된 디바이스를 제어할 수 있습니다.

관리형 통합에는 AWS 고객이 통합하려는 커넥터를 보고 선택할 수 있는 커넥터 카탈로그가 포함되어 있습니다. 자세한 내용은 C2C 커넥터 카탈로그란 무엇입니까? 섹션을 참조하세요.

관리형 통합을 사용하려면 모든 커넥터를 함수로 AWS Lambda 구현해야 합니다.

C2C 커넥터 카탈로그란 무엇입니까?

AWS IoT Device Management 커넥터 카탈로그의 관리형 통합은 AWS IoT Device Management의 관리형 통합과 타사 클라우드 공급자 간의 양방향 통신을 용이하게 하는 C2C 커넥터 모음입니다. AWS Management Console 또는에서 커넥터를 볼 수 있습니다 AWS CLI.

콘솔을 사용하여 관리형 통합 커넥터 카탈로그를 보려면

1. 관리형 통합 콘솔 열기
2. 왼쪽 탐색 창에서 관리형 통합을 선택합니다.
3. 관리형 통합 콘솔의 왼쪽 탐색 창에서 카탈로그를 선택합니다.

AWS Lambda C2C 커넥터로서의 함수

모든 C2C 커넥터 Lambda 함수는 관리형 통합과 타사 플랫폼의 해당 작업 간에 명령과 이벤트를 변환하고 전송합니다. Lambda에 대한 자세한 내용은 란 무엇입니까 AWS Lambda?를 참조하십시오.

예를 들어 최종 사용자가 타사 OEM에서 제조한 스마트 전구를 소유하고 있다고 가정해 보겠습니다. C2C 커넥터를 사용하면 최종 사용자가 관리형 통합 플랫폼을 통해이 조명을 켜거나 끄는 명령을 실행할 수 있습니다. 그러면이 명령이 커넥터에 호스팅된 Lambda 함수로 전달되어 요청을 타사 플랫폼에 대한 API 호출로 변환하여 디바이스를 켜거나 끕니다.

`CreateCloudConnector` API를 호출할 때 Lambda 함수가 필요합니다. Lambda 함수에 배포된 코드는에 언급된 모든 인터페이스와 기능을 구현해야 합니다C2C(Cloud-to-Cloud) 커넥터 빌드.

관리형 통합 커넥터 워크플로

개발자는 관리형 통합에 C2C 커넥터를 등록해야 합니다 AWS IoT Device Management. 이 등록 프로세스는 고객이 커넥터를 사용하기 위해 액세스할 수 있는 논리적 커넥터 리소스를 생성합니다.

> **Note**
>
> C2C 커넥터는 AWS IoT Device Management가 커넥터를 설명하기 위해 관리형 통합 내에서 생성된 메타데이터 세트입니다.

다음 다이어그램은 모바일 애플리케이션에서 클라우드 연결 디바이스로 명령을 전송할 때 C2C 커넥터의 역할을 보여줍니다. C2C 커넥터는 AWS IoT Device Management의 관리형 통합과 타사 클라우드 플랫폼 간의 변환 계층 역할을 합니다.

C2C(cloud-to-cloud) 커넥터 사용 지침

생성한 C2C 커넥터는 콘텐츠이고, 액세스한 다른 고객이 생성한 C2C 커넥터는 타사 콘텐츠입니다.는 관리형 통합의 일부로 C2C 커넥터를 생성하거나 관리하지 AWS 않습니다.

C2C 커넥터를 다른 관리형 통합 고객과 공유할 수 있습니다. 이렇게 하면 서비스 공급자 AWS 가 AWS 콘솔에 해당 C2C 커넥터 및 관련 연락처 정보를 나열할 수 있는 권한을 부여하고 다른 AWS 고객이 연락할 수 있음을 이해합니다. 고객에게 C2C 커넥터에 대한 액세스 권한을 부여하고 다른 AWS 고객의 C2C 커넥터 액세스 권한을 관리하는 모든 조건에 대한 책임은 전적으로 사용자에게 있습니다.

C2C(Cloud-to-Cloud) 커넥터 빌드

다음 섹션에서는 AWS IoT Device Management의 관리형 통합을 위한 C2C(Cloud-to-Cloud) 커넥터를 빌드하는 단계를 다룹니다.

주제

사전 조건

C2C(Cloud-to-Cloud) 커넥터를 생성하기 전에 다음이 필요합니다.

- C2C 커넥터를 호스팅하고 관리형 통합을 통해 등록 AWS 계정 하기 위한 입니다. 자세한 내용은 생성 단원을 AWS 계정참조하십시오.
- 커넥터를 빌드할 때는 특정 IAM 권한이 필요합니다. 를 사용하려면
- 커넥터가 사용되는 타사 클라우드 공급자가 OAuth 2.0 인증을 지원하는지 확인합니다. 자세한 내용은 계정 연결을 위한 OAuth 2.0 요구 사항 단원을 참조하십시오.

  또한 커넥터를 테스트하려면 커넥터 개발자에게 다음이 있어야 합니다.
  - C2C 커넥터와 연결할 타사 클라우드의 클라이언트 ID
  - C2C 커넥터와 연결할 타사 클라우드의 클라이언트 보안 암호
  - OAuth 2.0 권한 부여 URL
  - OAuth 2.0 토큰 URL
- 타사 API에 필요한 모든 API 키
- 에서 호스팅하는 OAuth 콜백 URL에 대한 타사 API 등록 또는 허용 목록에 필요한 모든 API 키입니다 AWS. 일부 타사는 OAuth 리디렉션 URL을 명시적으로 허용하지만, 다른 타사에는 사용자가 로그인하여 OAuth URL을 등록할 수 있는 워크플로가 있습니다. 관리형 통합 OAuth 리디렉션 엔드포인트를 허용 목록에 등록하는 데 필요한 사항을 이해하려면 특정 타사에 문의하세요.

필수 권한

커넥터를 빌드할 때는 특정 IAM 권한이 필요합니다. 작업에 대한 `iotmanagedintegrations:` 권한 외에도 다음 권한이 필요합니다.

- CreateAccountAssociation, CreateConnectorDestination, GetAccountAssociation 및 StartAccountAssociationRefresh, 필수 `secretsmanager:GetSecretValue`
- CreateCloudConnector에는 다음이 필요합니다. `lambda:Invoke`

`iotmanagedintegrations:` 권한 및 작업에 대한 자세한 내용은 AWS 관리형 통합에서 정의한 작업을 참조하세요.

C2C 커넥터 요구 사항

개발하는 C2C 커넥터는 AWS IoT Device Management의 관리형 통합과 타사 공급업체 클라우드 간의 양방향 통신을 용이하게 합니다. 커넥터는 AWS IoT Device Management가 최종 사용자를 대신하여 작업을 수행할 수 있도록 관리형 통합을 위한 인터페이스를 구현해야 합니다. 이러한 인터페이스는 최종 사용자 디바이스를 검색하고, AWS IoT Device Management 명령을 시작하고, 액세스 토큰을 기반으로 사용자를 식별하는 기능을 제공합니다. 디바이스 작업을 지원하려면 커넥터가 AWS IoT Device Management의 관리형 통합과 관련 타사 플랫폼 간의 요청 및 응답 메시지 번역을 관리해야 합니다.

C2C 커넥터에 대한 요구 사항은 다음과 같습니다.

- 타사 권한 부여 서버는 OAuth 2.0 표준과에 나열된 구성을 준수해야 합니다OAuth 구성 요구 사항.
- Matter Data Model AWS 구현의 식별자를 해석하려면 C2C 커넥터가 필요하며 Matter Data Model AWS 구현을 준수하는 응답과 이벤트를 내보내야 합니다. 자세한 내용은 AWS Matter 데이터 모델 구현 섹션을 참조하세요.
- C2C 커넥터는 AWS IoT Device Management APIs에 대한 관리형 통합을 `SigV4` 인증으로 호출할 수 있어야 합니다. SendConnectorEvent API로 전송되는 비동기 이벤트의 경우 커넥터를 등록하는 데 사용되는 것과 동일한 AWS 계정 자격 증명을 사용하여 관련 SendConnectorEvent 요청에 서명해야 합니다.
- 커넥터는 `AWS.ActivateUser`, `AWS.DiscoverDevicesAWS.SendCommand`, 및 `AWS.DeactivateUser` 작업을 구현해야 합니다.
- C2C 커넥터가 디바이스 명령 응답 또는 디바이스 검색과 관련된 타사 이벤트를 수신하면 `SendConnectorEvent` API와의 관리형 통합으로 전달해야 합니다. 이러한 이벤트 및 `SendConnectorEvent` API에 대한 자세한 내용은 SendConnectorEvent를 참조하세요.

> **Note**
>
> `SendConnectorEvent` API는 관리형 통합 SDK의 일부이며 요청의 수동 구축 및 서명 대신 사용됩니다.

계정 연결을 위한 OAuth 2.0 요구 사항

모든 C2C 커넥터는 OAuth 2.0 권한 부여 서버를 사용하여 최종 사용자를 인증합니다. 이 서버를 통해 최종 사용자는 타사 계정을 고객의 디바이스 플랫폼과 연결합니다. 계정 연결은 최종 사용자가 C2C 커넥터에서 지원하는 디바이스를 사용하는 데 필요한 첫 번째 단계입니다. 계정 연결 및 OAuth 2.0의 다양한 역할에 대한 자세한 내용은 섹션을 참조하세요계정 연결 역할.

C2C 커넥터는 권한 부여 흐름을 지원하기 위해 특정 비즈니스 로직을 구현할 필요는 없지만 C2C 커넥터와 연결된 OAuth2.0 권한 부여 서버는를 충족해야 합니다OAuth 구성 요구 사항.

> **Note**
>
> 용 관리형 통합은 권한 부여 코드 흐름이 있는 OAuth 2.0 AWS IoT Device Management 만 지원합니다. 자세한 내용은 RFC 6749를 참조하세요.

계정 연결은 관리형 통합 및 커넥터가 액세스 토큰을 사용하여 최종 사용자의 디바이스에 액세스할 수 있도록 허용하는 프로세스입니다. 이 토큰은 최종 사용자의 권한으로 AWS IoT Device Management에 대한 관리형 통합을 제공하므로 커넥터가 API 호출을 통해 최종 사용자의 데이터와 상호 작용할 수 있습니다. 자세한 내용은 계정 연결 워크플로 단원을 참조하십시오.

이러한 민감한 토큰을 로그에 기록하지 않는 것이 좋습니다. 그러나 로그에 저장되는 경우 CloudWatch Logs 데이터 보호 정책을 사용하여 로그의 토큰을 마스킹하는 것이 좋습니다. 자세한 내용은 Help protect sensitive log data with masking을 참조하세요.

에 대한 관리형 통합 AWS IoT Device Management 은 액세스 토큰을 직접 가져오지 않고 권한 부여 코드 부여 유형을 통해 가져옵니다. 먼저 AWS IoT Device Management의 관리형 통합은 권한 부여 코드를 얻어야 합니다. 그런 다음 코드를 액세스 토큰 및 새로 고침 토큰으로 교환합니다. 새로 고침 토큰은 이전 액세스 토큰이 만료될 때 새 액세스 토큰을 요청하는 데 사용됩니다. 액세스 토큰과 새로 고침 토큰이 모두 만료된 경우 계정 연결 흐름을 다시 수행해야 합니다. `StartAccountAssociationRefresh` API 작업을 사용하여이 작업을 수행할 수 있습니다.

Important

발급된 액세스 토큰의 범위는 OAuth 클라이언트가 아닌 사용자별로 지정해야 합니다. 토큰이 클라이언트에 속한 모든 사용자의 모든 디바이스에 대한 액세스를 제공해서는 안 됩니다. 권한 부여 서버는 다음 중 하나를 수행해야 합니다.

- JWT 토큰과 같이 추출 가능한 최종 사용자(리소스 소유자) ID가 포함된 액세스 토큰을 발급합니다.
- 발급된 각 액세스 토큰의 최종 사용자 ID를 반환합니다.

OAuth 구성 요구 사항

다음 표는 AWS IoT Device Management가 계정 연결을 수행하는 데 필요한 관리형 통합을 위한 OAuth 권한 부여 서버의 필수 파라미터를 보여줍니다.

OAuth 서버 파라미터

필드	필수	Comment
`clientId`	예	애플리케이션의 퍼블릭 식별자입니다. 인증 흐름을 시작하는 데 사용되며 공개적으로 공유할 수 있습니다.
`clientSecret`	예	특히 액세스 토큰에 대한 권한 부여 코드를 교환할 때 권한 부여 서버로 애플리케이션을 인증하는 데 사용되는 보안 키입니다. 기밀을 유지해야 하며 공개적으로 공유해서는 안 됩니다.
`authorizationType`	예	이 권한 부여 구성에서 지원하는 권한 부여 유형입니다. 현재 "OAuth 2.0"만 지원됩니다.
`authUrl`	예	타사 클라우드 공급자의 권한 부여 URL입니다.

`tokenUrl`	예	타사 클라우드 공급자의 토큰 URL입니다.
`tokenEndpointAuthenticationScheme`	예	“HTTP_BASIC” 또는 “REQUEST_BODY_CREDENTIALS”의 인증 체계입니다. HTTP_BASIC는 클라이언트 자격 증명이 권한 부여 헤더에 포함되어 있음을 나타내는 반면, 래더는 요청 본문에 포함되어 있음을 나타냅니다.

액세스 토큰 문자열 값이 UTF-8 문자 집합으로 인코딩된 Base64가 되도록 사용하는 OAuth 서버를 구성해야 합니다.

계정 연결 역할

C2C 커넥터를 생성하려면 OAuth 2.0 권한 부여 서버 및 계정 연결이 필요합니다. 자세한 내용은 계정 연결 워크플로 단원을 참조하십시오.

OAuth 2.0은 계정 연결을 구현할 때 다음 네 가지 역할을 정의합니다.

1. 권한 부여 서버
2. 리소스 소유자(최종 사용자)
3. 리소스 서버
4. 클라이언트

다음은 이러한 각 OAuth 역할을 정의합니다.

권한 부여 서버

> 권한 부여 서버는 타사 클라우드에서 최종 사용자의 자격 증명을 식별하고 인증하는 서버입니다. 이 서버에서 제공하는 액세스 토큰은 AWS 최종 사용자의 고객 플랫폼 계정과 타사 플랫폼 계정을 연결할 수 있습니다. 이 프로세스를 계정 연결이라고 합니다.
>
> 권한 부여 서버는 다음을 제공하여 계정 연결을 지원합니다.

- 최종 사용자가 시스템에 로그인할 수 있는 로그인 페이지를 표시합니다. 이를 일반적으로 권한 부여 엔드포인트라고 합니다.
- 시스템의 최종 사용자를 인증합니다.
- 최종 사용자를 식별하는 권한 부여 코드를 생성합니다.
- AWS IoT Device Management의 관리형 통합에 권한 부여 코드를 전달합니다.
- AWS IoT Device Management의 관리형 통합에서 권한 부여 코드를 수락하고 AWS IoT Device Management의 관리형 통합이 시스템의 최종 사용자 데이터에 액세스하는 데 사용할 수 있는 액세스 토큰을 반환합니다. 이는 일반적으로 토큰 URI 또는 엔드포인트라는 별도의 URI를 통해 완료됩니다.

> **Important**
>
> 권한 부여 서버는 AWS IoT Device Management Connector용 관리형 통합과 함께 사용할 OAuth 2.0 권한 부여 코드 흐름을 지원해야 합니다. AWS IoT Device Management용 관리형 통합은 코드 교환용 증명 키(PKCE)를 통한 권한 부여 코드 흐름도 지원합니다.
> 권한 부여 서버는 다음 중 하나를 수행해야 합니다.
>
> - JWT 토큰과 같이 추출 가능한 최종 사용자 또는 리소스 소유자 ID가 포함된 액세스 토큰 발급
> - 발급된 각 액세스 토큰에 대한 최종 사용자 ID를 반환할 수 있어야 합니다.
>
> 그렇지 않으면 커넥터가 필요한 `AWS.ActivateUser`작업을 지원할 수 없습니다. 이렇게 하면 관리형 통합에서 커넥터 사용을 방지할 수 있습니다.

커넥터 개발자 또는 소유자가 자체 권한 부여 서버를 유지 관리하지 않는 경우 사용되는 권한 부여 서버는 커넥터 개발자 타사 플랫폼에서 관리하는 리소스에 대한 권한을 제공해야 합니다. 즉, 권한 부여 서버에서 관리형 통합으로 수신한 모든 토큰은 디바이스(리소스)에 의미 있는 보안 경계를 제공해야 합니다. 예를 들어 최종 사용자 토큰은 다른 최종 사용자 디바이스에서 명령을 허용하지 않습니다. 토큰에서 제공하는 권한은 플랫폼 내의 리소스에 매핑됩니다. Lights™ 예제를 생각해 보세요. 최종 사용자가 커넥터와 계정 연결 흐름을 시작하면 권한 부여 서버 앞에 있는 Lights™ 로그인 페이지로 리디렉션됩니다. 로그인하고 클라이언트에 권한을 부여한 후에는 커넥터에 Lights™ 계정 내의 리소스에 대한 액세스 권한을 부여하는 토큰을 제공합니다.

리소스 소유자(최종 사용자)

리소스 소유자는 계정 연결을 수행하여 계정과 연결된 리소스에 대한 AWS IoT Device Management 고객 액세스를 위한 관리형 통합을 허용합니다. 예를 들어 최종 사용자가 Lights™ 모바일 애플리케이션에 온보딩한 스마트 전구를 가정해 보겠습니다. 리소스 소유자는 디바이

스를 구매하고 온보딩한 최종 사용자 계정을 말합니다. 이 예제에서 리소스 소유자는 Lights OAuthOAuth2.0 계정으로 모델링됩니다. 리소스 소유자로서이 계정은 명령을 실행하고 디바이스를 관리할 수 있는 권한을 제공합니다.

리소스 서버

이는 액세스 권한이 필요한 보호된 리소스를 호스팅하는 서버입니다(디바이스 데이터). AWS 고객은 최종 사용자를 대신하여 보호된 리소스에 액세스해야 하며 계정 연결 후 AWS IoT Device Management 커넥터에 대한 관리형 통합을 통해 액세스해야 합니다. 이전의 스마트 전구를 예로 들어보면 리소스 서버는 온보딩된 후 전구를 관리하는 Lights™가 소유한 클라우드 기반 서비스입니다. 리소스 서버를 통해 리소스 소유자는 스마트 전구를 켜고 끄는 등의 명령을 스마트 전구에 실행할 수 있습니다. 보호된 리소스는 최종 사용자의 계정 및 최종 사용자가 권한을 제공했을 수 있는 기타 계정/엔터티에만 권한을 제공합니다.

클라이언트

이 컨텍스트에서 클라이언트는 C2C 커넥터입니다. 클라이언트는 최종 사용자를 대신하여 리소스 서버 내의 리소스에 대한 액세스 권한이 부여된 애플리케이션으로 정의됩니다. 계정 연결 프로세스는 커넥터인 클라이언트를 나타내며, 타사 클라우드 내에서 최종 사용자의 리소스에 대한 액세스를 요청합니다.

커넥터는 OAuth 클라이언트이지만 AWS IoT Device Management의 관리형 통합은 커넥터를 대신하여 작업을 수행합니다. 예를 들어 AWS IoT Device Management의 관리형 통합은 권한 부여 서버에 액세스 토큰을 가져오도록 요청합니다. 커넥터는 리소스 서버의 보호된 리소스(디바이스 데이터)에 액세스하는 유일한 구성 요소이므로 여전히 클라이언트로 간주됩니다.

최종 사용자가 온보딩한 스마트 전구를 고려합니다. 고객 플랫폼과 Lights™ 권한 부여 서버 간에 계정 연결이 완료되면 커넥터 자체가 리소스 서버와 통신하여 최종 사용자의 스마트 전구에 대한 정보를 검색합니다. 그러면 커넥터가 최종 사용자로부터 명령을 수신할 수 있습니다. 여기에는 Lights™ 리소스 서버를 통해 조명을 자동으로 켜거나 끄는 작업이 포함됩니다. 따라서 커넥터를 클라이언트로 지정합니다.

계정 연결 워크플로

AWS IoT Device Management 플랫폼용 고객의 관리형 통합이 C2C 커넥터를 통해 타사 플랫폼에서 최종 사용자의 디바이스와 상호 작용할 수 있도록 다음 워크플로를 통해 액세스 토큰을 가져옵니다.

1. 사용자가 고객 애플리케이션을 통해 타사 디바이스의 온보딩을 시작하면 AWS IoT Device Management의 관리형 통합은 권한 부여 URI와 AssociationId를 반환합니다.

2. 애플리케이션 프런트엔드는 AssociationId를 저장하고 최종 사용자를 타사 플랫폼의 로그인 페이지로 리디렉션합니다.

   - 최종 사용자가 로그인합니다. 최종 사용자는 클라이언트에게 디바이스 데이터에 대한 액세스 권한을 부여합니다.

3. 타사 플랫폼은 권한 부여 코드를 생성합니다. 최종 사용자는 리디렉션 요청에 연결된 코드를 포함하여 AWS IoT Device Management 플랫폼 콜백 URI의 관리형 통합으로 리디렉션됩니다.
4. 관리형 통합은이 코드를 타사 플랫폼 토큰 URI와 교환합니다.
5. 토큰 URI는 권한 부여 코드를 검증하고 최종 사용자와 연결된 OAuth2.0 액세스 토큰 및 새로 고침 토큰을 반환합니다.
6. 관리형 통합은 `AWS.ActivateUser` 작업을 통해 C2C 커넥터를 호출하여 계정 연결 흐름을 완료하고 UserId를 가져옵니다.
7. 관리형 통합은 성공한 인증 페이지의 OAuthRedirectUrl(커넥터 정책 구성에서)을 고객 애플리케이션에 반환합니다.

   > **Note**
   >
   > 장애가 발생할 경우 AWS IoT Device Management의 관리형 통합은 오류 및 error_description 쿼리 파라미터를 URL에 추가하여 고객 애플리케이션에 오류 세부 정보를 제공합니다.

8. 고객 애플리케이션은 최종 사용자를 OAuthRedirectUrl로 리디렉션합니다. 이때 애플리케이션 프런트 엔드는 첫 번째 단계에서 연결의 AssociationId를 알고 있습니다.

   디바이스 검색 및 명령 전송 명령과 같이 C2C 커넥터를 통해 타사 클라우드 플랫폼에 대한 AWS IoT Device Management의 관리형 통합에서 이루어진 모든 후속 요청에는 OAuth2.0 액세스 토큰이 포함됩니다.

다음 다이어그램은 계정 연결의 주요 구성 요소 간의 관계를 보여줍니다.

C2C 커넥터 인터페이스 작업 구현

용 관리형 통합은 커넥터로 검증하기 위해 처리 AWS Lambda 해야 하는 4가지 작업을 AWS IoT Device Management 정의합니다. C2C 커넥터는 다음 각 작업을 구현해야 합니다.

1. AWS.ActivateUser - AWS IoT Device Management 서비스용 관리형 통합은이 API를 호출하여 제공된 OAuth2.0 토큰과 연결된 전역적으로 고유한 사용자 식별자를 검색합니다. 이 작업은 선택적으로 계정 연결 프로세스에 대한 추가 요구 사항을 수행하는 데 사용할 수 있습니다.
2. AWS.DiscoverDevices - AWS IoT Device Management 서비스의 관리형 통합은 사용자의 디바이스를 검색하기 위해이 API를 커넥터에 호출합니다.
3. AWS.SendCommand - AWS IoT Device Management 서비스의 관리형 통합은 사용자 디바이스에 대한 명령을 보내기 위해이 API를 커넥터에 호출합니다.
4. AWS.DeactivateUser - AWS IoT Device Management 서비스의 관리형 통합은 권한 부여 서버에서 연결을 해제하기 위해 사용자의 액세스 토큰을 비활성화하기 위해이 API를 커넥터로 호출합니다.

용 관리형 통합은 AWS IoT Device Management 작업을 통해 항상 JSON 문자열 페이로드를 사용하여 Lambda 함수를 호출합니다 AWS Lambda `invokeFunction`. 요청 작업은 모든 요청 페이로드에 `operationName` 필드를 포함해야 합니다. 자세한 내용은 AWS Lambda API 참조의 호출을 참조하세요.

각 호출 제한 시간은 2초로 설정되며, 호출이 실패하면 5회 재시도됩니다.

커넥터에 대해 구현하는 Lambda는 요청 페이로드`operationName`에서를 구문 분석하고 해당 기능을 구현하여 타사 클라우드에 매핑합니다.

```
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**
>
> 커넥터 개발자는 이전 예제에 나열된 `activateUserManager.activateUser(request)`, `deviceDiscoveryManager.listDevices(request)sendCommandManager.sendCommand(re` 및 `deactivateUser.deactivateUser` 작업을 구현해야 합니다.

다음 예제에서는 모든 필수 인터페이스에 대한 공통 필드가 있는 관리형 통합의 일반 커넥터 요청을 자세히 설명합니다. 예제에서 요청 헤더와 요청 페이로드가 모두 있는 것을 볼 수 있습니다. 요청 헤더는 모든 작업 인터페이스에서 공통적입니다.

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

기본 요청 헤더

기본 헤더 필드는 다음과 같습니다.

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

커넥터에서 호스팅하는 모든 API는 다음 헤더 파라미터를 처리해야 합니다.

기본 헤더 및 필드

필드	필수/선택 사항	설명
`header:auth`	예	커넥터 등록 중에 C2C 커넥터 빌더가 제공하는 권한 부여 정보입니다.
`header:auth:token`	예	타사 클라우드 공급자가 생성하고에 연결된 사용자의 권한 부여 토큰입니다`connectorAssociationID` .
`header:auth:type`	예	필요한 권한 부여 유형입니다.

> **Note**
>
> 커넥터에 대한 모든 요청에는 최종 사용자의 액세스 토큰이 연결됩니다. 최종 사용자와 관리형 통합 고객 간의 계정 연결이 이미 발생했다고 가정할 수 있습니다.

요청 페이로드

공통 헤더 외에도 모든 요청에는 페이로드가 있습니다. 이 페이로드에는 모든 작업 유형에 대한 고유한 필드가 있지만 각 페이로드에는 항상 존재하는 기본 필드 세트가 있습니다.

요청 페이로드 필드:

- `operationName`: 지정된 요청의 작업으로, , `AWS.ActivateUser`, `AWS.SendCommand``AWS.DiscoverDevices`, 값 중 하나와 같습니다`AWS.DeactivateUser`.
- `operationVersion`: 모든 작업은 시간이 지남에 따라 진화하고 타사 커넥터에 대한 안정적인 인터페이스 정의를 제공하도록 버전이 지정됩니다. 관리형 통합은 모든 요청의 페이로드에 버전 필드를 전달합니다.
- `connectorId`: 요청이 전송된 커넥터의 ID입니다.

기본 응답 헤더

모든 작업은 C2C 커넥터가 요청을 수신하고 처리를 시작했음을 확인하는 AWS IoT Device Management의 ACK 관리형 통합에 로 응답합니다. 다음은 해당 응답의 일반적인 예입니다.

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

모든 작업 응답에는 다음과 같은 공통 헤더가 있어야 합니다.

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

다음 표에는 기본 응답 헤더가 나열되어 있습니다.

기본 응답 헤더 및 필드

필드	필수/선택 사항	Comment
header:responseCode	예	요청의 실행 상태를 나타내는 값의 ENUM입니다.

이 문서에 설명된 다양한 커넥터 인터페이스 및 API 스키마 전체에 responseMessage 또는 Message 필드가 있습니다. 이 필드는 C2C 커넥터 Lambda가 요청 및 실행과 관련된 컨텍스트로 응답하는 데 사용되는 선택적 필드입니다. 가급적이면 이외의 상태 코드를 초래하는 모든 오류에는 오류를 설명하는 메시지 값이 포함되어야 200 합니다.

SendConnectorEvent API를 사용하여 C2C 커넥터 작업 요청에 응답

용 관리형 통합은 모든 `AWS.SendCommand` 및 `AWS.DiscoverDevices` 작업에 대해 커넥터가 비동기적으로 작동할 것으로 AWS IoT Device Management 예상합니다. 즉, 이러한 작업에 대한 초기 응답은 단순히 C2C 커넥터가 요청을 수신했음을 "확인"한다는 의미입니다.

`SendConnectorEvent` API를 사용하면 커넥터가 `AWS.DiscoverDevices` 및 `AWS.SendCommand` 작업과 사전 예방적 디바이스 이벤트(예: 수동으로 켜고 끄는 등)를 위해 아래 목록에서 로 이벤트 유형을 전송해야 합니다. 이러한 이벤트 유형 및 사용 사례에 대한 자세한 설명은 AWS.DiscoverDevices 작업 구현, AWS.SendCommand 작업 구현및 단원을 참조하십시오SendConnectorEvent API를 사용하여 디바이스 이벤트 전송.

예를 들어 C2C 커넥터가 `DiscoverDevices` 요청을 수신하면 AWS IoT Device Management 의 관리형 통합은 위에서 정의한 응답 형식과 동기적으로 응답할 것으로 예상합니다. 그런 다음 DEVICE_DISCOVERY 이벤트에 AWS.DiscoverDevices 작업 구현대해에 정의된 요청 구조로 `SendConnectorEvent`API를 호출해야 합니다. `SendConnectorEvent` 온 API 호출은 C2C 커넥터 Lambda AWS 계정 자격 증명에 액세스할 수 있는 모든 곳에서 발생할 수 있습니다. AWS IoT Device Management 검색 흐름이 성공하지 못합니다.

> **Note**
>
> 또는 필요한 경우 C2C 커넥터 Lambda 호출 응답 전에 `SendConnectorEvent` API 호출이 발생할 수 있습니다. 그러나이 흐름은 소프트웨어 개발을 위한 비동기 모델과 모순됩니다.

- SendConnectorEvent - 커넥터가 AWS IoT Device Management API에 대한이 관리형 통합을 호출하여 AWS IoT Device Management에 대한 관리형 통합으로 디바이스 이벤트를 전송합니다. 관리형 통합에서 허용되는 이벤트 유형은 3가지뿐입니다.
  - "DEVICE_DISCOVERY" -이 이벤트 작업은 특정 액세스 토큰에 대해 타사 클라우드 내에서 검색된 디바이스 목록을 전송하는 데 사용해야 합니다.
  - "DEVICE_COMMAND_RESPONSE" -이 이벤트 작업은 명령 실행의 결과로 특정 디바이스 이벤트를 전송하는 데 사용해야 합니다.
  - "DEVICE_EVENT" -이 이벤트 작업은 사용자 기반 명령의 직접적인 결과가 아닌 디바이스에서 시작된 모든 이벤트에 사용해야 합니다. 이는 디바이스 상태 변경 또는 알림을 사전에 보고하는 일반적인 이벤트 유형 역할을 할 수 있습니다.

AWS.ActivateUser 작업 구현

AWS IoT Device Management가 최종 사용자의 OAuth2.0 토큰에서 사용자 식별자를 검색하려면 관리형 통합에 `AWS.ActivateUser` 작업이 필요합니다. 에 대한 관리형 통합 AWS IoT Device Management 은 요청 헤더 내에 OAuth 토큰을 전달하고 커넥터가 응답 페이로드에 전역적으로 고유한 사용자 식별자를 포함할 것으로 예상합니다. 이 작업은 계정 연결 흐름이 성공한 후 발생합니다.

다음 목록은 성공적인 `AWS.Activate` 사용자 흐름을 촉진하기 위한 커넥터의 요구 사항을 간략하게 설명합니다.

- C2C 커넥터 Lambda는 AWS IoT Device Management에 대한 관리형 통합의 `AWS.ActivateUser` 작업 요청 메시지를 처리할 수 있습니다.
- C2C 커넥터 Lambda는 제공된 OAuth2.0 토큰에서 고유한 사용자 식별자를 확인할 수 있습니다. 일반적으로 JWT 토큰인 경우 토큰 자체에서 추출하거나 토큰에 의해 권한 부여 서버에서 요청할 수 있습니다.

`AWS.ActivateUser` 워크플로

1. 용 관리형 통합은 다음 AWS IoT Device Management 페이로드를 사용하여 C2C 커넥터 Lambda를 호출합니다.

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

2. C2C 커넥터는 토큰에서 또는 타사 리소스 서버를 쿼리하여 `AWS.ActivateUser` 응답에 포함할 사용자 ID를 결정합니다.
3. C2C 커넥터는 기본 페이로드와 `userId` 필드 내의 해당 사용자 식별자를 포함하여 `AWS.ActivateUser` 작업 Lambda 호출에 응답합니다.

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

AWS.DiscoverDevices 작업 구현

디바이스 검색은 최종 사용자가 소유한 물리적 디바이스 목록을 AWS IoT Device Management의 관리형 통합에서 유지 관리하는 최종 사용자 디바이스의 디지털 표현과 일치시킵니다. AWS IoT Device Management에서 AWS 고객이 수행합니다. 디바이스 검색은 AWS IoT Device Management의 관리형 통합이 커넥터를 호출하여 디바이스 검색 요청을 시작하는 비동기 프로세스입니다. C2C 커넥터는 검색된 최종 사용자 디바이스 목록을 관리형 통합에서 생성된 참조 식별자(라고 함`deviceDiscoveryId`)와 비동기적으로 반환합니다.

다음 다이어그램은 AWS IoT Device Management 검색 워크플로를 보여줍니다.

AWS.DiscoverDevices 워크플로

1. 고객은 최종 사용자를 대신하여 디바이스 검색 프로세스를 시작합니다.
2. 용 관리형 통합은 고객이 생성한 디바이스 검색 요청에 `deviceDiscoveryId` 대해 라는 참조 식별자를 AWS IoT Device Management AWS 생성합니다.
3. 용 관리형 통합은 최종 사용자의 유효한 OAuth와를 포함하여 `AWS.DiscoverDevices` 작업 인터페이스를 사용하여 디바이스 검색 요청을 C2C 커넥터`accessToken`로 AWS IoT Device Management 보냅니다`deviceDiscoveryId`.
4. 커넥터는 `DEVICE_DISCOVERY` 이벤트에 포함할 `deviceDiscoveryId`를 저장합니다. 이 이벤트에는 검색된 최종 사용자의 디바이스 목록도 포함되며, `SendConnectorEvent` API를 `DEVICE_DISCOVERY` 이벤트로 사용하여 AWS IoT Device Management의 관리형 통합으로 전송해야 합니다.
5. C2C 커넥터는 리소스 서버를 호출하여 최종 사용자가 소유한 모든 디바이스를 가져와야 합니다.

6. C2C 커넥터 Lambda는 AWS IoT Device Management의 관리형 통합에 대한 ACK 응답으로 Lambda 호출(invokeFunction)에 응답하며, 이는 AWS.DiscoverDevices 작업에 대한 초기 응답 역할을 합니다. 관리형 통합은 ACK를 통해 고객에게 시작된 디바이스 검색 프로세스에 대해 알립니다.
7. 리소스 서버는 최종 사용자가 소유하고 운영하는 디바이스 목록을 전송합니다.
8. 커넥터는 각 최종 사용자 디바이스를 각 디바이스에 대한 및 ConnectorDeviceId ConnectorDeviceName 기능 보고서를 포함하여 AWS IoT Device Management 필수 디바이스 형식의 관리형 통합으로 변환합니다.
9. C2C 커넥터는 검색된 디바이스 소유자UserId의 도 제공합니다. 리소스 서버 구현에 따라 디바이스 목록의 일부로 또는 별도의 호출로 리소스 서버에서 검색할 수 있습니다.
10. 다음으로 C2C 커넥터는 자격 AWS 계정 증명과 "DEVICE_DISCOVERY"로 설정된 작업 파라미터를 사용하여 SigV4를 통해 AWS IoT Device Management APISendConnectorEvent, 에 대한 관리형 통합을 호출합니다. AWS IoT Device Management connectorDeviceId, connectorDeviceName및와 같은 디바이스별 파라미터로 표시됩니다capabilityReport.

    - 리소스 서버 응답에 따라 그에 따라 AWS IoT Device Management의 관리형 통합에 알려야 합니다.

      예를 들어 리소스 서버에 최종 사용자의 검색된 디바이스 목록에 대한 페이지 매김 응답이 있는 경우 각 폴링에 대해 statusCode 파라미터를 사용하여 개별 DEVICE_DISCOVERY 작업 이벤트를 보낼 수 있습니다3xx. 디바이스 검색이 아직 진행 중인 경우 5, 6, 7단계를 반복합니다.
11. 용 관리형 통합은가 최종 사용자의 디바이스를 검색했다는 알림을 고객에게 AWS IoT Device Management 보냅니다.
12. C2C 커넥터가 statusCode 파라미터가 200으로 업데이트된 DEVICE_DISCOVERY 작업 이벤트를 전송하는 경우 관리형 통합은 고객에게 디바이스 검색 워크플로 완료를 알립니다.

> **Important**
>
> 원하는 경우 6단계 전에 7~11단계를 수행할 수 있습니다. 예를 들어 타사 플랫폼에 최종 사용자 디바이스를 나열하는 API가 있는 경우 C2C 커넥터 Lambda가 일반적인 ACK로 응답SendConnectorEvent하기 전에 DEVICE_DISCOVERY 이벤트를 로 전송할 수 있습니다.

디바이스 검색을 위한 C2C 커넥터 요구 사항

다음 목록은 성공적인 디바이스 검색을 용이하게 하기 위한 C2C 커넥터의 요구 사항을 간략하게 설명합니다.

- C2C 커넥터 Lambda는 AWS IoT Device Management 검색 요청 메시지를 처리하고 `AWS.DiscoverDevices` 작업을 처리할 수 있습니다.
- C2C 커넥터는 커넥터 등록에 AWS 계정 사용되는의 자격 증명을 사용하여 SigV4를 통해 AWS IoT Device Management APIs에 대한 관리형 통합을 호출할 수 있습니다.

디바이스 검색 프로세스

다음 단계에서는 AWS IoT Device Management를 위한 C2C 커넥터 및 관리형 통합을 사용한 디바이스 검색 프로세스를 간략하게 설명합니다. AWS IoT Device Management

디바이스 검색 프로세스

1. 관리형 통합은 디바이스 검색을 트리거합니다.
    - 다음 JSON 페이로드를 `DiscoverDevices` 사용하여에 POST 요청을 보냅니다.

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

2. 커넥터는 검색을 승인합니다.
    - 커넥터는 다음 JSON 응답과 함께 승인을 전송합니다.

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

3. 커넥터가 디바이스 검색 이벤트를 전송합니다.

   - 다음 JSON 페이로드를 /connector-event/*{your_connector_id}* 사용하여에 POST 요청을 보냅니다.

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

DISCOVER_DEVICES 이벤트에 대한 CapabilityReport 구성

위에 정의된 이벤트 구조에서 볼 수 있듯이 `AWS.DiscoverDevices` 작업에 대한 응답으로 사용되는 DISCOVER_DEVICES 이벤트에서 보고된 모든 디바이스에는 해당 디바이스의 기능을 설명하는 CapbilityReport가 필요합니다. 'CapabilityReport'는 AWS IoT Device Management 디바이스 기능에 대한 관리형 통합을 Matter 호환 형식으로 알려줍니다. ` CapabilityReport `에 다음 필드를 제공해야 합니다.

- `nodeId`, 문자열: 다음을 포함하는 디바이스 노드의 식별자입니다. `endpoints`
- `version`, 문자열: 커넥터 개발자가 설정한이 디바이스 노드의 버전
- `endpoints`, List<Cluster>:이 디바이스 엔드포인트에서 지원하는 Matter Data Model의 AWS 구현 목록입니다.
  - `id`, 문자열: 커넥터 개발자가 설정한 엔드포인트 식별자
  - `deviceTypes`, List<String>:이 엔드포인트가 캡처하는 디바이스 유형 목록, 즉 "Camera".
  - `clusters`, List<Cluster>:이 엔드포인트가 지원하는 Matter Data Model의 AWS 구현 목록입니다.
    - `id`, 문자열: Matter 표준에 정의된 클러스터 식별자입니다.
    - `revision`, 정수: Matter 표준에 정의된 클러스터 개정 번호입니다.
    - `attributes`, Map<String, Object>: 항목 표준에 정의된 식별자 및 유효한 값과 함께 속성 식별자 및 해당 현재 디바이스 상태 값의 맵입니다.
      - `id`, 문자열: Matter 데이터 모델의 AWS 구현에서 정의한 속성 ID입니다.
      - `value`, 객체: 속성 ID로 정의된 속성의 현재 값입니다. '값' 유형은 속성에 따라 변경될 수 있습니다. `value` 필드는 각 속성에 대해 선택 사항이며 커넥터 Lambda가 검색 중에 현재 상태를 확인할 수 있는 경우에만 포함되어야 합니다.
    - `commands`, List<String>: Matter 표준에 정의된 대로이 클러스터를 지원하는 명령 IDs.
    - `events`, List<String>: Matter 표준에 정의된 대로이 클러스터를 지원하는 이벤트 IDs.

지원되는 기능의 현재 목록과 AWS Matter Data Model의 해당 구현은 Data Model 설명서의 최신 릴리스를 참조하세요.

AWS.SendCommand 작업 구현

`AWS.SendCommand` 작업을 통해 AWS IoT Device Management의 관리형 통합은 최종 사용자가 AWS 고객을 통해 시작한 명령을 리소스 서버로 전송할 수 있습니다. 리소스 서버는 각 유형에 자체 응답 모델이 있는 여러 유형의 디바이스를 지원할 수 있습니다. 명령 실행은 AWS IoT Device Management의 관리형 통합이 'traceId'를 사용하여 명령 실행 요청을 보내는 비동기 프로세스로, 커넥터는 'SendConnectorEvent' API를 통해 관리형 통합으로 다시 전송되는 명령 응답에 포함됩니다. AWS IoT Device Management의 관리형 통합은 리소스 서버가 명령이 수신되었음을 승인하지만 명령이 실행되었음을 반드시 나타내는 것은 아닙니다.

다음 다이어그램은 최종 사용자가 집의 조명을 켜려고 하는 예제와 함께 명령 실행 흐름을 보여줍니다.

디바이스 명령 실행 워크플로

1. 최종 사용자는 AWS 고객의 애플리케이션을 사용하여 조명을 켜는 명령을 보냅니다.
2. 고객은 최종 사용자의 디바이스 정보와 함께 AWS IoT Device Management의 관리형 통합에 명령 정보를 전달합니다.
3. 관리형 통합은 커넥터가 명령 응답을 서비스로 다시 보내는 동안 사용할 "traceId"를 생성합니다.
4. AWS IoT Device Management의 관리형 통합은 `AWS.SendCommand` 작업 인터페이스를 사용하여 명령 요청을 커넥터로 보냅니다.
   - 이 인터페이스에서 정의한 페이로드는 디바이스 식별자, Matter endpoints/clusters/commands으로 공식화된 디바이스 명령, 최종 사용자의 액세스 토큰 및 기타 필수 파라미터로 구성됩니다.
5. 커넥터는 명령 응답에 `traceId` 포함할를 저장합니다.
   - 커넥터는 관리형 통합 명령 요청을 리소스 서버의 적절한 형식으로 변환합니다.
6. 커넥터는 제공된 최종 사용자의 액세스 토큰`UserId`에서 가져와 명령과 연결합니다.
   a. 는 별도의 호출을 사용하여 리소스 서버에서 검색하거나 JWT 및 유사한 토큰의 경우 액세스 토큰에서 추출할 `UserId` 수 있습니다.
   b. 구현은 리소스 서버 및 액세스 토큰 세부 정보에 따라 달라집니다.
7. 커넥터가 리소스 서버를 호출하여 최종 사용자의 조명을 "켜십시오".

8. 리소스 서버는 디바이스와 상호 작용합니다.

   a. 커넥터는 리소스 서버가 명령을 전달한 AWS IoT Device Management의 관리형 통합으로 릴레이되어 초기 동기식 명령 응답으로 ACK로 응답합니다.

   b. 그런 다음 관리형 통합은 이를 고객 애플리케이션으로 다시 릴레이합니다.

9. 디바이스가 켜지면 리소스 서버에서 해당 디바이스 이벤트를 캡처합니다.
10. 리소스 서버가 디바이스 이벤트를 커넥터로 보냅니다.
11. 커넥터는 리소스 서버에서 생성된 디바이스 이벤트를 관리형 통합 DEVICE_COMMAND_RESPONSE 이벤트 작업 유형으로 변환합니다.
12. 커넥터는 작업을 "DEVICE_COMMAND_RESPONSE"로 사용하여 `SendConnectorEvent` API를 호출합니다.

    - 초기 요청에서 AWS IoT Device Management에 대한 관리형 통합에서 `traceId` 제공하는를 연결합니다.

13. 관리형 통합은 고객에게 최종 사용자의 디바이스 상태 변경을 알립니다.
14. 고객은 최종 사용자에게 디바이스의 조명이 켜져 있음을 알립니다.

> **Note**
>
> 리소스 서버 구성에 따라 실패한 디바이스 명령 요청 및 응답 메시지를 처리하기 위한 로직이 결정됩니다. 여기에는 명령에 동일한 referenceId를 사용하는 메시지 재시도가 포함됩니다.

디바이스 명령 실행을 위한 C2C 커넥터 요구 사항

다음 목록은 성공적인 디바이스 명령 실행을 용이하게 하기 위한 C2C 커넥터의 요구 사항을 간략하게 설명합니다.

- C2C 커넥터 Lambda는 AWS IoT Device Management에 대한 관리형 통합의 `AWS.SendCommand` 작업 요청 메시지를 처리할 수 있습니다.
- C2C 커넥터는 리소스 서버로 전송된 명령을 추적하고 적절한 'traceId'로 매핑해야 합니다.
- C2C 커넥터를 등록하는 데 AWS 계정 사용되는의 AWS 자격 증명을 사용하여 SigV4를 통해 AWS IoT Device Management 서비스 API에 대한 관리형 통합을 호출할 수 있습니다.

1. 관리형 통합은 명령을 커넥터로 전송합니다(이전 다이어그램의 4단계 참조).

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

2. C2C 커넥터 ACK 명령(커넥터가 AWS IoT Device Management Service의 관리형 통합으로 ACK를 전송하는 이전 다이어그램의 7단계 참조).

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

3. 커넥터는 디바이스 명령 응답 이벤트를 전송합니다(이전 다이어그램의 11단계 참조).

- ```
  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**
>
> 명령 실행의 결과로 인한 디바이스 상태 변경은 SendConnectorEvent API를 통해 해당 DEVICE_COMMAND_RESPONSE 이벤트를 수신할 때까지 AWS IoT Device Management의 관리형 통합에 반영되지 않습니다. 즉, 관리형 통합이 이전 3단계에서 이 벤트를 수신할 때까지 커넥터 호출 응답이 성공을 나타내는지 여부에 관계없이 디바이스 상태는 업데이트되지 않습니다.
  ```

AWS.SendCommand 요청에 포함된 문제 '엔드포인트' 해석

관리형 통합은 디바이스 검색 중에 보고된 디바이스 기능을 사용하여 디바이스가 수락할 수 있는 명령을 결정합니다. 모든 디바이스 기능은 Matter Data Model의 AWS 구현을 통해 모델링되므로 모든 수신 명령은 지정된 클러스터 내의 '명령' 필드에서 파생됩니다. 커넥터는 'endpoints' 필드를 구문 분석하여 해당 Matter 명령을 결정하고 올바른 명령이 디바이스에 도달하도록 변환할 책임이 있습니다. 일반적으로 이는 Matter 데이터 모델을 관련 API 요청으로 변환하는 것을 의미합니다.

명령이 실행된 후 커넥터는 Matter Data Model의 AWS 구현에 의해 정의된 '속성'이 결과적으로 변경되었는지 확인합니다. 그런 다음 이러한 변경 사항은 API와 함께 전송된 API DEVICE_COMMAND_RESPONSE 이벤트를 통해 AWS IoT Device Management의 관리형 통합에 보고됩니다`SendConnectorEvent`.

다음 예제 `AWS.SendCommand`페이로드에 포함된 '엔드포인트' 필드를 고려합니다.

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

이 객체에서 커넥터는 다음을 확인할 수 있습니다.

1. 엔드포인트 및 클러스터 정보를 설정합니다.

   a. 엔드포인트를 "1"`id`로 설정합니다.

   > **Note**
   >
   > 디바이스가 단일 클러스터(예: 켜기/끄기)가 여러 기능을 제어할 수 있도록 여러 엔드포인트를 정의하는 경우(예: 조명 켜기/끄기 및 스트로브 켜기/끄기)이 ID는 명령을 올바른 기능으로 라우팅하는 데 사용됩니다.

   b. 클러스터를 "0x0202"(Fan Control 클러스터)`id`로 설정합니다.

2. 명령 정보를 설정합니다.

    a. 명령 식별자를 "0xff01"(에서 정의한 상태 업데이트 명령)로 설정합니다 AWS.

    b. 포함된 속성 식별자를 요청에 제공된 값으로 업데이트합니다.

3. 속성을 업데이트합니다.

    a. 속성 식별자를 "0x0000"(FanMode 속성)으로 설정합니다.

    b. 속성 값을 "3"(높은 팬 속도)으로 설정합니다.

관리형 통합은 Matter Data Model의 AWS 구현으로 엄격하게 정의되지 않은 두 가지 “사용자 지정” 명령 유형인 ReadState 및 UpdateState 명령을 정의했습니다. Matter 정의 클러스터 속성을 가져오고 설정하기 위해 관리형 통합은 UpdateState(id: 0xff01) 또는 ReadState(id: 0xff02)`AWS.SendCommand`와 관련된 명령 IDs와 함께 업데이트하거나 읽어야 하는 속성의 해당 파라미터를 사용하여 커넥터를 전송합니다. 이러한 명령은 Matter Data Model의 해당 AWS 구현에서 변경 가능(업데이트 가능) 또는 검색 가능(읽기 가능)으로 설정된 속성의 모든 디바이스 유형에 대해 호출할 수 있습니다.

SendConnectorEvent API를 사용하여 디바이스 이벤트 전송

디바이스 시작 이벤트 개요

`SendConnectorEvent` API는 `AWS.SendCommand` 및 `AWS.DiscoverDevices` 작업에 비동기적으로 응답하는 데 사용되지만 디바이스 시작 이벤트를 관리형 통합에 알리는 데도 사용됩니다. 디바이스 시작 이벤트는 사용자 시작 명령 없이 디바이스에서 생성된 모든 이벤트로 정의할 수 있습니다. 이러한 디바이스 이벤트에는 디바이스 상태 변경, 모션 감지, 배터리 잔량 등이 포함될 수 있지만 이에 국한되지 않습니다. DEVICE_EVENT 작업과 함께 `SendConnectorEvent` API를 사용하여 이러한 이벤트를 관리형 통합으로 다시 보낼 수 있습니다.

다음 섹션에서는 홈에 설치된 스마트 카메라를 예로 사용하여 이러한 이벤트의 작업 흐름을 자세히 설명합니다.

디바이스 이벤트 워크플로

1. 카메라는 리소스 서버로 전송되는 이벤트를 생성하는 모션을 감지합니다.
2. 리소스 서버는 이벤트를 처리하고 C2C 커넥터로 전송합니다.
3. 커넥터는이 이벤트를 AWS IoT Device Management `DEVICE_EVENT` 인터페이스의 관리형 통합으로 변환합니다.

4. C2C 커넥터는 작업이 "DEVICE_EVENT"로 설정된 SendConnectorEvent API를 사용하여이 디바이스 이벤트를 관리형 통합으로 전송합니다.
5. 관리형 통합은 관련 고객을 식별하고이 이벤트를 고객에게 다시 전달합니다.
6. 고객은이 이벤트를 수신하여 사용자 식별자를 통해 사용자에게 표시합니다.

SendConnectorEvent API 작업에 대한 자세한 내용은 AWS IoT Device Management API용 관리형 통합 참조 안내서SendConnectorEvent의 섹션을 참조하세요.

디바이스 시작 이벤트 요구 사항

다음은 디바이스 시작 이벤트에 대한 몇 가지 요구 사항입니다.

- C2C 커넥터 리소스는 리소스 서버에서 비동기 디바이스 이벤트를 수신할 수 있어야 합니다.
- C2C 커넥터 리소스는 C2C 커넥터를 등록하는 데 AWS 계정 사용되는의 AWS 자격 증명을 사용하여 SigV4를 통해 AWS IoT Device Management 서비스 API에 대한 관리형 통합을 호출할 수 있어야 합니다.

다음 예제에서는 SendConnectorEvent API를 통해 디바이스에서 시작된 이벤트를 전송하는 커넥터를 보여줍니다.

```
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"
                }
            ]
        }]
```

```
    }]
}
```

다음 예제에서 다음을 볼 수 있습니다.

- ID가 1인 디바이스 엔드포인트에서 가져옵니다.
- 이 이벤트와 관련된 디바이스 기능은 팬 제어 문제 클러스터와 관련하여 클러스터 ID가 0x0202입니다.
- 변경된 속성의 ID는 0x000이며 클러스터 내의 팬 모드 열거형과 관련이 있습니다. High 값과 관련하여 값 3으로 업데이트되었습니다.
- `connectorId`는 생성 시 클라우드 서비스에서 반환하는 파라미터이므로 커넥터는 GetCloudConnector를 사용하여 쿼리하고를 기준으로 필터링해야 합니다`lambdaARN`. Lambda 자체ARN는 `Lambda.get_function_url_config` API를 사용하여 쿼리됩니다. 이렇게 하면 lambda에서 `CloudConnectorId`에 동적으로 액세스할 수 있으며 이전과 같이 정적으로 구성되지 않습니다.

AWS.DeactivateUser 작업 구현

사용자 비활성화 개요

고객이 고객 AWS 계정을 삭제하거나 최종 사용자가 시스템의 계정을 AWS 고객의 시스템에서 연결 해제하려는 경우 제공된 사용자 액세스 토큰을 비활성화해야 합니다. 어느 사용 사례에서든 관리형 통합은 C2C 커넥터를 사용하여이 워크플로를 용이하게 해야 합니다.

아래 이미지는 시스템에서 최종 사용자 계정의 연결을 해제하는 방법을 보여줍니다.

사용자 비활성화 워크플로

1. 사용자는 AWS 고객의 계정과 C2C 커넥터와 연결된 타사 권한 부여 서버 간에 연결 해제 프로세스를 시작합니다.
2. 고객은 AWS IoT Device Management에 대한 관리형 통합을 통해 사용자 연결 삭제를 시작합니다.
3. 관리형 통합은 `AWS.DeactivateUser` 작업 인터페이스를 사용하여 커넥터에 대한 요청을 통해 비활성화 프로세스를 시작합니다.

    - /user의 액세스 토큰은 요청의 헤더에 포함됩니다.

4. C2C 커넥터는 요청을 수락하고 권한 부여 서버를 호출하여 토큰과 토큰이 제공하는 모든 액세스를 취소합니다.

   - 예를 들어 연결되지 않은 사용자 계정의 이벤트는를 수행한 후 더 이상 관리형 통합으로 전송되지 않아야 합니다`AWS.DeactivateUser`.

5. 권한 부여 서버가 액세스를 취소하고 응답을 C2C 커넥터로 다시 보냅니다.
6. C2C 커넥터는 AWS IoT Device Management에 대한 관리형 통합을 사용자의 액세스 토큰이 취소되었다는 ACK로 전송합니다.
7. 관리형 통합은 리소스 서버와 연결된 최종 사용자가 소유한 모든 리소스를 삭제합니다.
8. 관리형 통합은 고객에게 ACK를 전송하여 시스템과 관련된 모든 연결이 삭제되었음을 알립니다.
9. 고객은 최종 사용자에게 플랫폼에서 계정이 연결 해제되었음을 알립니다.

AWS.DeactivateUser 요구 사항

- C2C 커넥터 Lambda 함수는 `AWS.DeactivateUser` 작업을 처리하기 위해 관리형 통합으로부터 요청 메시지를 수신합니다.
- C2C 커넥터는 제공된 OAuth2.0 토큰과 권한 부여 서버 내 사용자의 해당 새로 고침 토큰을 취소해야 합니다.

다음은 커넥터가 수신할 `AWS.DeactivateUser`요청의 예입니다.

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

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

C2C 커넥터 호출

AWS Lambda 를 사용하면 리소스 기반 정책이 Lambda를 호출할 수 있는 사용자를 승인할 수 있습니다. AWS IoT Device Management의 관리형 통합은 이므로 관리형 통합이 리소스 정책을 통해 C2C 커넥터 Lambda를 호출하도록 허용 AWS 서비스해야 합니다.

최소한 다음과 같은 최소 권한이 있는 리소스 정책을 C2C 커넥터 Lambda에 연결합니다. 이렇게 하면 Lambda 함수 호출 권한과의 관리형 통합이 제공됩니다. 이 정책에는의 사용 가능성을 의도한 사용자로만 제한`connectorId`하는 데 도움이 되는 `Condition` 키가 포함되어 있습니다.

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"
        }
      }
    }
  ]
}
```

IAM 역할에 권한 추가

모든 관리형 통합 APIs 호출하려면 AWS sigV4 인증이 필요합니다. SigV4는 자격 AWS 계정 증명을 사용하여 AWS API 요청을 인증하기 위해 프로토콜에 서명합니다. 관리형 통합 APIs를 호출하는 데 사용하는 IAM 역할에는 API를 APIs

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

이러한 권한 추가에 대한 자세한 내용은에 문의하십시오 지원.

추가 리소스

C2C 커넥터를 등록하려면 다음이 필요합니다.

- 등록하려는 커넥터를 지정하는 Lambda ARN입니다.

C2C 커넥터 수동 테스트

C2C 커넥터 end-to-end를 수동으로 테스트하려면 고객과 최종 사용자를 모두 시뮬레이션해야 합니다.

다음 리소스가 필요합니다.

- 테스트하려는 커넥터를 지정하는 AWS Lambda ARN입니다.
- 클라우드 플랫폼에서 OAuth 2.0 사용자 계정을 테스트합니다.
- AWS IoT Device Management의 관리형 통합에 등록된 커넥터입니다. 자세한 내용은 C2C(Cloud-to-Cloud) 커넥터 사용 단원을 참조하십시오.

C2C(Cloud-to-Cloud) 커넥터 사용

C2C 커넥터는 요청 및 응답 메시지의 번역을 관리하고 관리형 통합과 타사 공급업체 클라우드 간의 통신을 활성화합니다. 다양한 디바이스 유형, 플랫폼 및 프로토콜에서 통합 제어를 용이하게 하여 타사 디바이스를 온보딩하고 관리할 수 있습니다.

다음 절차에서는 C2C 커넥터를 사용하는 단계를 나열합니다.

C2C 커넥터를 사용하는 단계:

1. CreateCloudConnector

   관리형 통합과 타사 공급업체 클라우드 간의 양방향 통신을 활성화하도록 커넥터를 구성합니다.

   커넥터를 설정할 때 다음 세부 정보를 제공합니다.

   - 이름: 커넥터에 대한 설명 이름을 선택합니다.
   - 설명: 커넥터의 용도와 기능에 대한 간략한 요약을 제공합니다.
   - AWS Lambda ARN: 커넥터에 전원을 공급하는 AWS Lambda 함수의 Amazon 리소스 이름(ARN)을 지정합니다.

   타사 공급업체 APIs와 통신하는 AWS Lambda 함수를 빌드하고 배포하여 커넥터를 생성합니다. 그런 다음 관리형 통합 내에서 CreateCloudConnector API를 호출하고 등록할 AWS Lambda 함수 ARN을 제공합니다. 관리형 통합에서 커넥터를 생성하는 동일한 AWS 계정에 AWS Lambda 함수가 배포되었는지 확인합니다. 통합을 식별하기 위해 고유한 커넥터 ID가 할당됩니다.

   샘플 CreateCloudConnector API 요청 및 응답:

```
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"
}
```

생성 흐름:

> **Note**
>
> 이 절차의 필요에 따라 GetCloudConnector, UpdateCloudConnector, DeleteCloudConnector 및 ListCloudConnectors APIs를 사용합니다.

2. CreateConnectorDestination

커넥터가 타사 공급업체 클라우드와 보안 연결을 설정하는 데 필요한 설정 및 인증 자격 증명을 제공하도록 대상을 구성합니다. 대상을 사용하여 권한 부여 URL, 인증 체계 및 자격 증명 위치를 포함한 OAuth 2.0 권한 부여 세부 정보와 같은 관리형 통합에 타사 인증 자격 증명을 등록합니다 AWS Secrets Manager.

사전 조건

ConnectorDestination을 생성하기 전에 다음을 수행해야 합니다.

- CreateCloudConnector API를 호출하여 커넥터를 생성합니다. 함수가 반환하는 ID는 CreateConnectorDestination API 호출에 사용됩니다.
- 커넥터의 3P 플랫폼에 `tokenUrl` 대한를 검색합니다. (authCode를 accessToken으로 교환할 수 있습니다.)
- 커넥터의 3P 플랫폼에 대한 authUrl을 검색합니다. (최종 사용자는 사용자 이름과 암호를 사용하여 인증할 수 있습니다.)
- 계정의 보안 암호 관리자에서 `clientId` 및 `clientSecret` (3P 플랫폼에서)를 사용합니다.

샘플 CreateConnectorDestination API 요청 및 응답:

```
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"
}
```

클라우드 대상 생성 흐름:

> **Note**
>
> 이 절차의 필요에 따라 GetCloudConnector, UpdateCloudConnector, DeleteCloudConnector 및 ListCloudConnectors APIs를 사용합니다.

3. CreateAccountAssociation

연결은 최종 사용자의 타사 클라우드 계정과 커넥터 대상 간의 관계를 나타냅니다. 연결을 생성하고 최종 사용자를 관리형 통합에 연결하면 고유한 연결 ID를 통해 디바이스에 액세스할 수 있습니다. 이 통합을 통해 디바이스 검색, 명령 전송, 이벤트 수신이라는 세 가지 주요 기능을 사용할 수 있습니다.

사전 조건

AccountAssociation을 생성하기 전에 다음을 완료해야 합니다.

- CreateConnectorDestination API를 호출하여 대상을 생성합니다. 함수가 반환하는 ID는 CreateAccountAssociation API 호출에 사용됩니다.
- CreateAccountAssociation API를 호출합니다.

샘플 CreateAccountAssociation API 요청 및 응답:

```
Request:

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

Response:

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

> **Note**
>
> 이 절차의 필요에 따라 GetCloudConnector, UpdateCloudConnector, DeleteCloudConnector 및 ListCloudConnectors APIs를 사용합니다.

AccountAssociation에는 GetAccountAssociation 및 ListAccountAssociations APIs. 이러한 APIs 연결의 상태를 보여줍니다. StartAccountAssociationRefresh API를 사용하면 새로 고침 토큰이 만료될 때 AccountAssociation 상태를 새로 고칠 수 있습니다.

4. 디바이스 검색

각 관리형 사물은 일련 번호 및 데이터 모델과 같은 디바이스별 세부 정보에 연결됩니다. 데이터 모델은 디바이스의 기능을 설명하여 전구, 스위치, 온도 조절기 또는 다른 유형의 디바이스인지 여

부를 나타냅니다. 3P 디바이스를 검색하고 3P 디바이스에 대한 managedThing을 생성하려면 아래 단계를 순서대로 따라야 합니다.

a. StartDeviceDiscovery API를 호출하여 디바이스 검색 프로세스를 시작합니다.

샘플 StartDeviceDiscovery API 요청 및 응답:

```
Request:

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

Response:

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

b. GetDeviceDiscovery API를 호출하여 검색 프로세스의 상태를 확인합니다.

c. ListDiscoveredDevices API를 호출하여 검색된 디바이스를 나열합니다.

샘플 ListDiscoveredDevices API 요청 및 응답:

```
Request:

//Empty body

Response:

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

```
      "Modification": "string"
    }
],
    "NextToken": "string"
}
```

d. CreateManagedThing API를 호출하여 검색 목록에서 관리형 통합으로 가져올 디바이스를 선택합니다.

샘플 CreateManagedThing API 요청 및 응답:

```
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. GetManagedThing API를 호출하여 새로 생성된를 확인합니다`managedThing`. 상태는 입니다`UNASSOCIATED`.

f. RegisterAccountAssociation API를 호출하여 이를 특정 `managedThing`에 연결합니다`accountAssociation`. 성공적인 RegisterAccountAssociation API가 끝나면 상태가 `managedThing` 변경됩니다`ASSOCIATED`.

샘플 RegisterAccountAssociation API 요청 및 응답:

```
Request:

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

```
    "ManagedThingId": "string"
}

Response:

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

5. 3P 디바이스로 명령 전송

   새로 온보딩된 디바이스를 제어하려면 이전에 생성한 연결 ID와 함께 SendManagedThingCommand API를 사용하고 디바이스에서 지원하는 기능을 기반으로 하는 제어 작업을 사용합니다. 커넥터는 계정 연결 프로세스의 저장된 자격 증명을 사용하여 타사 클라우드로 인증하고 작업에 대한 관련 API 호출을 호출합니다.

   샘플 SendManagedThingCommand API 요청 및 응답:

```
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"
}
```

명령을 3P 디바이스 흐름으로 전송합니다.

6. 커넥터가 관리형 통합으로 이벤트 전송

SendConnectorEvent API는 커넥터에서 관리형 통합에 이르는 네 가지 유형의 이벤트를 캡처하며, 작업 유형 파라미터에 대한 다음 열거형 값으로 표시됩니다.

- DEVICE_COMMAND_RESPONSE: 커넥터가 명령에 대한 응답으로 보내는 비동기 응답입니다.
- DEVICE_DISCOVERY: 커넥터는 디바이스 검색 프로세스에 대한 응답으로 검색된 디바이스 목록을 관리형 통합으로 전송하고 SendConnectorEvent API를 사용합니다.
- DEVICE_EVENT: 수신된 디바이스 이벤트를 전송합니다.
- DEVICE_COMMAND_REQUEST: 디바이스에서 시작된 명령 요청입니다. WebRTC 워크플로를 예로 들 수 있습니다.

커넥터는 선택적 `userId` 파라미터와 함께 SendConnectorEvent API를 사용하여 디바이스 이벤트를 전달할 수도 있습니다.

- 를 사용하는 디바이스 이벤트의 경우`userId`:

  샘플 SendConnectorEvent API 요청 및 응답:

```
Request:

{
    "UserId": "*****",
    "Operation": "DEVICE_EVENT",
    "OperationVersion": "1.0",
    "StatusCode": 200,
```

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

Response:

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

- 이 없는 디바이스 이벤트의 경우`userId`:

  샘플 SendConnectorEvent API 요청 및 응답:

```
Request:

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

Response:

{
    "ConnectorId": "string"
```

```
}
```

특정 연결managedThing과 계정 연결 간의 링크를 제거하려면 등록 취소 메커니즘을 사용합니다.

샘플 DeregisterAccountAssociation API 요청 및 응답:

```
Request:

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

Response:

HTTP/1.1 200 // Empty body
```

이벤트 흐름 전송:

7. 커넥터 상태를 "Listed"로 업데이트하여 다른 관리형 통합 고객에게 표시

   기본적으로 커넥터는 프라이빗이며 커넥터를 생성한 AWS 계정에만 표시됩니다. 커넥터를 다른 관리형 통합 고객에게 표시하도록 선택할 수 있습니다.

   커넥터를 다른 사용자와 공유하려면 커넥터 세부 정보 페이지의에서 표시 옵션을 사용하여 AWS Management Console 검토를 AWS 위해 커넥터 ID를에 제출합니다. 승인되면 동일한의 모든 관리형 통합 사용자가 커넥터를 사용할 수 있습니다 AWS 리전. 또한 커넥터의 연결된 AWS Lambda 함수에서 액세스 정책을 수정하여 특정 AWS 계정 IDs에 대한 액세스를 제한할 수 있습니다. 다른 고객이 커넥터를 사용할 수 있도록 하려면 다른 AWS 계정에서 표시되는 커넥터로 Lambda 함수에 대한 IAM 액세스 권한을 관리합니다.

   커넥터를 다른 관리형 통합 고객에게 표시하기 전에 커넥터 공유 및 액세스 권한을 관리하는 AWS 서비스 용어와 조직의 정책을 검토하세요.

관리형 통합 Hub SDK

이 섹션의 주제를 사용하여 관리형 통합 Hub SDK를 사용하여 IoT 허브 디바이스를 온보딩하고 제어하는 방법을 알아봅니다. 관리형 통합 엔드 디바이스 SDK에 대한 자세한 내용은 섹션을 참조하세요관리형 통합 엔드 디바이스 SDK.

Hub SDK 아키텍처

디바이스 온보딩

관리형 통합 작업을 시작하기 전에 Hub SDK 구성 요소가 디바이스 온보딩을 지원하는 방법을 검토하세요. 이 섹션에서는 코어 프로비저너와 프로토콜별 플러그인이 함께 작동하여 디바이스 인증, 통신 및 사용자 설정을 처리하는 방법을 포함하여 디바이스 온보딩에 필요한 필수 아키텍처 구성 요소를 다룹니다.

디바이스 온보딩을 위한 Hub SDK 구성 요소

SDK 구성 요소

- 코어 프로비저너
- 프로토콜별 프로비저너 플러그인
- 프로토콜별 미들웨어

코어 프로비저너

코어 프로비저너는 IoT 허브 배포에서 디바이스 온보딩을 오케스트레이션하는 중앙 구성 요소입니다. 관리형 통합과 프로토콜별 프로비저너 플러그인 간의 모든 통신을 조정하여 안전하고 안정적인 디바이스 온보딩을 보장합니다. 디바이스를 온보딩할 때 코어 프로비저너는 인증 흐름을 처리하고, MQTT 메시징을 관리하고, 다음 함수를 통해 디바이스 요청을 처리합니다.

MQTT 연결

클라우드 주제 게시 및 구독을 위해 MQTT 브로커와의 연결을 생성합니다.

메시지 대기열 및 핸들러

수신되는 디바이스 추가 및 제거 요청을 순서대로 처리합니다.

프로토콜 플러그인 인터페이스

인증 및 라디오 조인 모드를 관리하여 디바이스 온보딩을 위한 프로토콜별 프로비저너 플러그인과 함께 작동합니다.

Hub SDK 클라이언트 APIs

프로토콜별 CDMB 플러그인에서 관리형 통합으로 디바이스 기능 보고서를 수신하고 전달합니다.

프로토콜별 프로비저너 플러그인

프로토콜별 프로비저너 플러그인은 서로 다른 통신 프로토콜에 대한 디바이스 온보딩을 관리하는 라이브러리입니다. 각 플러그인은 코어 프로비저너의 명령을 IoT 디바이스에 대한 프로토콜별 작업으로 변환합니다. 이러한 플러그인은 다음을 수행합니다.

- 프로토콜별 미들웨어 초기화
- 코어 프로비저너 요청을 기반으로 한 무선 조인 모드 구성
- 미들웨어 API 호출을 통한 디바이스 제거

프로토콜별 미들웨어

프로토콜별 미들웨어는 디바이스 프로토콜과 관리형 통합 간의 변환 계층 역할을 합니다. 이 구성 요소는 프로비저닝 플러그인에서 명령을 수신하여 프로토콜 스택으로 전송하는 동시에 디바이스에서 응답을 수집하고 시스템을 통해 다시 라우팅하는 양방향 통신을 처리합니다.

디바이스 온보딩 흐름

Hub SDK를 사용하여 디바이스를 온보딩할 때 발생하는 작업 순서를 검토합니다. 이 섹션에서는 온보딩 프로세스 중에 구성 요소가 상호 작용하는 방식을 표시하고 지원되는 온보딩 방법을 간략하게 설명합니다.

온보딩 흐름

- 단순 설정(SS)
- 제로 터치 설정(ZTS)
- 사용자 안내 설정(UGS)

단순 설정(SS)

최종 사용자는 IoT 디바이스의 전원을 켜고 디바이스 제조업체 애플리케이션을 사용하여 QR 코드를 스캔합니다. 그런 다음 디바이스가 관리형 통합 클라우드에 등록되고 IoT 허브에 연결됩니다.

제로 터치 설정(ZTS)

제로 터치 설정(ZTS)은 공급망의 디바이스 업스트림을 사전 연결하여 디바이스 온보딩을 간소화합니다. 예를 들어 최종 사용자가 디바이스 QR 코드를 스캔하는 대신이 단계는 이전에 완료되어 디바이스를 고객 계정에 사전 연결합니다. 예를 들어이 단계는 이행 센터에서 완료할 수 있습니다.

최종 사용자가 디바이스를 수신하고 전원을 켜면 추가 설정 작업 없이 관리형 통합 클라우드에 자동으로 등록되고 IoT 허브에 연결됩니다.

사용자 안내 설정(UGS)

최종 사용자는 디바이스의 전원을 켜고 대화형 단계에 따라 관리형 통합에 온보딩합니다. 여기에는 IoT 허브의 버튼을 누르거나, 디바이스 제조업체 앱을 사용하거나, 허브와 디바이스 모두에서 버튼을 누를 수 있습니다. 단순 설정이 실패하면이 방법을 사용할 수 있습니다.

디바이스 제어

관리형 통합은 디바이스 등록, 명령 실행 및 제어를 처리합니다. 공급업체 및 프로토콜에 구애받지 않는 디바이스 관리를 사용하여 디바이스별 프로토콜에 대한 지식 없이 최종 사용자 경험을 구축할 수 있습니다.

디바이스 제어를 사용하면 전구 밝기 또는 도어 위치와 같은 디바이스 상태를 보고 수정할 수 있습니다. 이 기능은 상태 변경에 대한 이벤트를 내보내며, 이를 분석, 규칙 및 모니터링에 사용할 수 있습니다.

주요 기능

디바이스 상태 수정 또는 읽기

디바이스 유형에 따라 디바이스 속성을 보고 변경합니다. 에 액세스할 수 있습니다.

- 디바이스 상태: 현재 디바이스 속성 값

- 연결 상태: 디바이스 연결성 상태
- 상태: 배터리 잔량 및 신호 강도(RSSI)와 같은 시스템 값

상태 변경 알림

전구 밝기 조정 또는 도어 잠금 상태 변경과 같은 디바이스 속성 또는 연결 상태가 변경될 때 이벤트를 수신합니다.

오프라인 모드

디바이스는 인터넷 연결 없이도 동일한 IoT 허브의 다른 디바이스와 통신합니다. 연결이 재개되면 디바이스 상태가 클라우드와 동기화됩니다.

상태 동기화

여러 소스, 디바이스 제조업체 앱 및 수동 디바이스 조정의 상태 변경을 추적합니다.

관리형 통합을 통해 디바이스를 제어하는 데 필요한 Hub SDK 구성 요소 및 프로세스를 검토합니다. 이 주제에서는 Edge Agent, CDMB(Common Data Model Bridge) 및 프로토콜별 플러그인이 함께 작동하여 디바이스 명령을 처리하고, 디바이스 상태를 관리하고, 다양한 프로토콜에서 응답을 처리하는 방법을 설명합니다.

디바이스 제어 흐름

다음 다이어그램은 최종 사용자가 Zigbee 스마트 플러그를 켜는 방법을 설명하여 end-to-end 디바이스 제어 흐름을 보여줍니다.

디바이스 제어를 위한 Hub SDK 구성 요소

Hub SDK 아키텍처는 다음 구성 요소를 사용하여 IoT 구현에서 디바이스 제어 명령을 처리하고 라우팅합니다. 각 구성 요소는 클라우드 명령을 디바이스 작업으로 변환하고, 디바이스 상태를 관리하고, 응답을 처리하는 데 특정 역할을 합니다. 다음 섹션에서는 배포에서 이러한 구성 요소가 함께 작동하는 방법을 자세히 설명합니다.

Hub SDK는 다음 구성 요소로 구성되며 IoT 허브에서 디바이스 온보딩 및 제어를 용이하게 합니다.

기본 구성 요소:

엣지 에이전트

IoT 허브와 관리형 통합 간의 게이트웨이 역할을 합니다.

공통 데이터 모델 브리지(CDMB)

AWS 데이터 모델과 Z-Wave 및 Zigbee와 같은 로컬 프로토콜 데이터 모델 간에 변환됩니다. 여기에는 코어 CDMB 및 프로토콜별 CDMB 플러그인이 포함됩니다.

프로비저너

디바이스 검색 및 온보딩을 처리합니다. 여기에는 프로토콜별 온보딩 작업을 위한 코어 프로비저너와 프로토콜별 프로비저너 플러그인이 포함됩니다.

보조 구성 요소

허브 온보딩

보안 클라우드 통신을 위해 클라이언트 인증서 및 키로 허브를 프로비저닝합니다.

MQTT 프록시

관리형 통합 클라우드에 대한 MQTT 연결을 제공합니다.

Logger

로컬 또는 관리형 통합 클라우드에 로그를 씁니다.

관리형 통합 Hub SDK 설치 및 검증

자동 배포 또는 수동 스크립트 설치를AWS IoT Greengrass 위해 디바이스에 관리형 통합 Hub SDK를 설치하려면 다음 배포 방법 중에서 선택합니다. 이 섹션에서는 두 접근 방식에 대한 설정 및 검증 단계를 설명합니다.

배포 방법

- 를 사용하여 Hub SDK 설치 AWS IoT Greengrass
- 스크립트를 사용하여 Hub SDK 배포
- systemd를 사용하여 Hub SDK 배포

를 사용하여 Hub SDK 설치 AWS IoT Greengrass

AWS IoT Greengrass (Java 버전)를 사용하여 디바이스에 대한 관리형 통합 Hub SDK 구성 요소를 배포합니다.

> **Note**
>
> 이미를 설정하고 이해해야 합니다 AWS IoT Greengrass. 자세한 내용은 AWS IoT Greengrass 개발자 안내서 설명서의 정의 AWS IoT Greengrass 섹션을 참조하세요.

AWS IoT Greengrass 사용자는 다음 디렉터리를 수정할 수 있는 권한이 있어야 합니다.

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

주제

로컬에 구성 요소 배포

디바이스에서 CreateDeployment AWS IoT Greengrass API를 사용하여 Hub SDK 구성 요소를 배포합니다. 버전 번호는 정적이지 않으며 현재 사용하는 버전에 따라 다를 수 있습니다. 에는 **`version`**com.amazon.IoTManagedIntegrationsDevice.AceCommon=`0.2.0` 형식을 사용합니다.

```
/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"
```

클라우드 배포

AWS IoT Greengrass 개발자 안내서의 지침에 따라 다음 단계를 수행합니다.

1. Amazon S3에 아티팩트를 업로드합니다.
2. Amazon S3 아티팩트 위치를 포함하도록 레시피를 업데이트합니다.
3. 새 구성 요소에 대해 디바이스에 클라우드 배포를 생성합니다.

허브 프로비저닝 확인

구성 파일을 확인하여 프로비저닝이 성공했는지 확인합니다. `/data/aws/iotmi/config/iotmi_config.json` 파일을 열고 상태가 로 설정되어 있는지 확인합니다`PROVISIONED`.

CDMB 작업 확인

로그 파일에 CDMB 시작 메시지와 성공적인 초기화가 있는지 확인합니다. *## ##* 위치는가 설치된 위치에 따라 다를 수 AWS IoT Greengrass 있습니다.

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

예시

```
[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
```

LPW-Provisioner 작업 확인

로그 파일에서 LPW-Provisioner 시작 메시지 및 성공적인 초기화를 확인합니다. *## ##* 위치는가 설치된 위치에 따라 다를 수 AWS IoT Greengrass 있습니다.

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

예시

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

스크립트를 사용하여 Hub SDK 배포

설치 스크립트를 사용하여 관리형 통합 Hub SDK 구성 요소를 수동으로 배포한 다음 배포를 검증합니다. 이 섹션에서는 스크립트 실행 단계 및 확인 프로세스에 대해 설명합니다.

주제

환경 준비

SDK 설치 스크립트를 실행하기 전에 다음 단계를 완료합니다.

1. 폴더 `middleware` 내에 라는 `artifacts` 폴더를 생성합니다.
2. 허브 미들웨어 파일을 `middleware` 폴더에 복사합니다.
3. SDK를 시작하기 전에 초기화 명령을 실행합니다.

> **Important**
>
> 각 허브 재부팅 후 초기화 명령을 반복합니다.

```
#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
```

Hub SDK 스크립트 실행

아티팩트 디렉터리로 이동하여 `start_iotmi_sdk.sh` 스크립트를 실행합니다. 이 스크립트는 허브 SDK 구성 요소를 올바른 순서로 시작합니다. 다음 예제 로그를 검토하여 성공적인 시작을 확인합니다.

> **Note**
>
> 실행 중인 모든 구성 요소에 대한 로그는 `artifacts/logs` 폴더 내에서 찾을 수 있습니다.

```
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----
```

허브 프로비저닝 확인

의 `iot_provisioning_state` 필드가 로 `/data/aws/iotmi/config/iotmi_config.json` 설정되어 있는지 확인합니다`PROVISIONED`.

에이전트 작업 확인

로그 파일에 에이전트 시작 메시지와 성공적인 초기화가 있는지 확인합니다.

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

예시

```
[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**
>
> `artifacts` 디렉터리에 `iotmi.db` 데이터베이스가 있는지 확인합니다.

LPW-Provisioner 작업 확인

로그 파일에 `LPW-Provisioner` 시작 메시지와 성공적인 초기화가 있는지 확인합니다.

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

다음 코드에 예가 나와 있습니다.

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

systemd를 사용하여 Hub SDK 배포

> **Important**
>
> 최신 업데이트는 release.tgz 파일의 hubSystemdSetup 디렉터리readme.md에 있는를 따르세요.

이 섹션에서는 Linux 기반 허브 디바이스에 서비스를 배포하고 구성하기 위한 스크립트 및 프로세스에 대해 설명합니다.

개요

배포 프로세스는 두 가지 기본 스크립트로 구성됩니다.

- copy_to_hub.sh: 호스트 시스템에서 실행하여 필요한 파일을 허브에 복사합니다.
- setup_hub.sh: 허브에서 실행하여 환경을 구성하고 서비스를 배포합니다.

또한는 프로세스 부트스트랩 시퀀스 및 프로세스 권한 관리를 systemd/deploy_iotshd_services_on_hub.sh 처리하고에 의해 자동으로 트리거됩니다setup_hub.sh.

사전 조건

배포에 성공하려면 나열된 사전 요구 사항이 필요합니다.

- 허브에서 systemd 서비스 사용 가능
- 허브 디바이스에 대한 SSH 액세스
- 허브 디바이스에 대한 Sudo 권한
- scp 호스트 시스템에 설치된 유틸리티
- sed 호스트 시스템에 설치된 유틸리티

- 호스트 시스템에 설치된 유틸리티의 압축을 풉니다.

파일 구조

파일 구조는 다양한 구성 요소의 구성 및 관리를 용이하게 하여 콘텐츠에 효율적으로 액세스하고 탐색할 수 있도록 설계되었습니다.

```
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
```

SDK 릴리스 tgz 파일에서 전체 파일 구조는 다음과 같습니다.

```
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
```

초기 설정

SDK 패키지 추출

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

추출된 디렉터리로 이동하여 패키지를 준비합니다.

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

디바이스 구성 파일 추가

나열된 두 단계에 따라 디바이스 구성 파일을 생성하고 허브에 복사합니다.

1. 디바이스 구성 파일을 추가하여 필요한 디바이스 구성 파일을 생성합니다. SDK는이 파일을 함수에 사용합니다.
2. 구성 파일을 복사하여 생성된 구성 파일을 허브에 복사합니다.

허브에 파일 복사

호스트 시스템에서 배포 스크립트를 실행합니다.

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

Example예시

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

이 복사본은 다음을 복사합니다.

- 패키지 파일(허브에서 package.zip으로 이름이 변경됨)
- 구성 파일
- 인증서
- 시스템화된 서비스 파일

허브 설정

파일을 복사한 후 SSH를 허브에 넣고 설정 스크립트를 실행합니다.

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

```
sudo ./setup_hub.sh
```

사용자 및 그룹 구성

기본적으로 SDK 구성 요소에는 사용자 허브와 그룹 허브를 사용합니다. 여러 가지 방법으로 구성할 수 있습니다.

- 사용자 지정 사용자/그룹 사용:

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

- 설정 스크립트를 실행하기 전에 수동으로 생성합니다.

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

- 에서 명령을 추가합니다`setup_hub.sh`.

서비스 관리

모든 서비스를 다시 시작하려면 허브에서 다음 스크립트를 실행합니다.

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

설정 스크립트는 필요한 디렉터리를 생성하고, 적절한 권한을 설정하고, 서비스를 자동으로 배포합니다. SSH/SCP를 사용하지 않는 경우 특정 배포 방법에 `copy_to_hub.sh` 맞게를 수정해야 합니다. 배포하기 전에 모든 인증서 파일 및 구성이 올바르게 설정되었는지 확인합니다.

허브를 관리형 통합에 온보딩

필요한 디렉터리 구조, 인증서 및 디바이스 구성 파일을 구성하여 관리형 통합과 통신하도록 허브 디바이스를 설정합니다. 이 섹션에서는 허브 온보딩 하위 시스템 구성 요소가 함께 작동하는 방법, 인증서 및 구성 파일을 저장할 위치, 디바이스 구성 파일을 생성하고 수정하는 방법, 허브 프로비저닝 프로세스를 완료하는 단계를 설명합니다.

허브 온보딩 하위 시스템

허브 온보딩 하위 시스템은 다음과 같은 핵심 구성 요소를 사용하여 디바이스 프로비저닝 및 구성을 관리합니다.

허브 온보딩 구성 요소

허브 상태, 프로비저닝 접근 방식 및 인증 자료를 조정하여 허브 온보딩 프로세스를 관리합니다.

디바이스 구성 파일

다음을 포함한 필수 허브 구성 데이터를 디바이스에 저장합니다.

- 디바이스 프로비저닝 상태(프로비저닝됨 또는 프로비저닝되지 않음)
- 인증서 및 키 위치
- 인증 정보 MQTT 프록시와 같은 다른 SDK 프로세스는이 파일을 참조하여 허브 상태 및 연결 설정을 결정합니다.

인증서 핸들러 인터페이스

디바이스 인증서 및 키를 읽고 쓸 수 있는 유틸리티 인터페이스를 제공합니다. 이 인터페이스를 구현하여 다음 작업을 수행할 수 있습니다.

- 파일 시스템 스토리지
- 하드웨어 보안 모듈(HSM)
- 신뢰할 수 있는 플랫폼 모듈(TPM)
- 사용자 지정 보안 스토리지 솔루션

MQTT 프록시 구성 요소

다음을 사용하여 device-to-cloud 통신을 관리합니다.

- 프로비저닝된 클라이언트 인증서 및 키
- 구성 파일의 디바이스 상태 정보
- 관리형 통합에 대한 MQTT 연결

다음 다이어그램은 허브 온보딩 하위 시스템 아키텍처와 해당 구성 요소를 설명합니다. 를 사용하지 않는 경우 다이어그램의 해당 구성 요소를 무시할 AWS IoT Greengrass수 있습니다.

허브 온보딩 설정

플릿 프로비저닝 온보딩 프로세스를 시작하기 전에 각 허브 디바이스에 대해 다음 설정 단계를 완료합니다. 이 섹션에서는 관리형 사물을 생성하고, 디렉터리 구조를 설정하고, 필요한 인증서를 구성하는 방법을 설명합니다.

설정 단계

1단계: 사용자 지정 엔드포인트 등록

디바이스가 관리형 통합과 데이터를 교환하는 데 사용하는 전용 통신 엔드포인트를 생성합니다. 이 엔드포인트는 device-to-cloud 메시징에 대한 보안 연결 지점을 설정합니다.

엔드포인트를 등록하려면

- RegisterCustomEndpoint API를 사용하여 device-to-managed 통합 통신을 위한 엔드포인트를 생성합니다.

  RegisterCustomEndpoint 요청 예제

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

  응답:

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

  > **Note**
  >
  > 엔드포인트 주소를 저장합니다. 향후 디바이스 통신에 필요합니다.

  엔드포인트 정보를 반환하려면 `GetCustomEndpoint` API를 사용합니다.

  자세한 내용은 관리형 통합 API 참조 안내서의 RegisterCustomEndpoint API 및 GetCustomEndpoint API를 참조하세요.

2단계: 프로비저닝 프로필 생성

프로비저닝 프로필에는 디바이스가 관리형 통합에 연결하는 데 필요한 보안 자격 증명 및 구성 설정이 포함되어 있습니다.

플릿 프로비저닝 프로필을 생성하려면

- CreateProvisioningProfile API를 호출하여 다음을 생성합니다.
  - 디바이스 연결 설정을 정의하는 프로비저닝 템플릿
  - 디바이스 인증을 위한 클레임 인증서 및 프라이빗 키

> **Important**
>
> 클레임 인증서, 프라이빗 키 및 템플릿 ID를 안전하게 저장합니다. 디바이스를 관리형 통합에 온보딩하려면 이러한 자격 증명이 필요합니다. 이러한 자격 증명을 분실한 경우 새 프로비저닝 프로파일을 생성해야 합니다.

`CreateProvisioningProfile` 예제 요청

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

응답:

```
{
"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"
}
```

3단계: 관리형 사물 생성(플릿 프로비저닝)

CreateManagedThing API를 사용하여 허브 디바이스에 대한 관리형 사물을 생성합니다. 각 허브에는 고유한 인증 자료가 있는 자체 관리형 사물이 필요합니다. 자세한 내용은 관리형 통합 API 참조의 CreateManagedThing API를 참조하세요.

관리형 사물을 생성할 때 다음 파라미터를 지정합니다.

- Role: 명령 및 제어를 지원하지 않는 허브의 CONTROLLER 경우이 값을 로 설정하고, 그렇지 않으면 로 설정합니다DEVICE.
- AuthenticationMaterialType:이 값을 로 설정합니다WIFI_SETUP_QR_BAR_CODE.
- AuthenticationMaterial: 다음 필드를 포함합니다. UPC 또는를 사용할 수 EAN 있지만 둘 다 사용할 수는 없습니다.
  - SN:이 디바이스의 고유 일련 번호
  - UPC:이 디바이스의 범용 제품 코드
  - EAN:이 디바이스의 국제 문서 번호

> **Important**
>
> 각 디바이스의 인증 자료에는 고유한 일련 번호(SN)가 있어야 합니다.

CreateManagedThing 요청 예제:

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

자세한 내용은 관리형 통합 API 참조의 CreateManagedThing을 참조하세요.

(선택 사항) 관리형 사물 가져오기

PRE_ASSOCIATED 계속하려면 관리형 사물ProvisioningStatus의가 여야 합니다. ProvisioningStatus에 대한 자세한 내용은 디바이스 프로비저닝을 참조하세요. GetManagedThing API를 사용하여 관리형 사물이 존재하고 프로비저닝할 준비가 되었는지 확인합니다. 자세한 내용은 관리형 통합 API 참조의 GetManagedThing을 참조하세요.

4단계: 디렉터리 구조 생성

구성 파일 및 인증서의 디렉터리를 생성합니다. 기본적으로 허브 온보딩 프로세스는를 사용합니다/data/aws/iotmi/config/iotmi_config.json.

구성 파일에서 인증서 및 프라이빗 키에 대한 사용자 지정 경로를 지정할 수 있습니다. 이 가이드에서는 기본 경로를 사용합니다/data/aws/iotmi/certs.

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

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

5단계: 허브 디바이스에 인증 자료 추가

인증서와 키를 허브 디바이스에 복사한 다음 디바이스별 구성 파일을 생성합니다. 이러한 파일은 프로비저닝 프로세스 중에 허브와 관리형 통합 간의 보안 통신을 설정합니다.

클레임 인증서 및 키를 복사하려면

- CreateProvisioningProfile API 응답에서 허브 디바이스로 이러한 인증 파일을 복사합니다.
  - claim_cert.pem: 클레임 인증서(모든 디바이스에 공통)
  - claim_pk.key: 클레임 인증서의 프라이빗 키

  /data/aws/iotmi/certs 디렉터리에 두 파일을 모두 배치합니다.

> **Important**
>
> 인증서와 프라이빗 키를 PEM 형식으로 저장할 때는 줄 바꿈 문자를 올바르게 처리하여 적절한 형식을 지정해야 합니다. PEM 인코딩 파일의 경우 줄 바꿈 문자를 실제 줄 구분자로 바꿔(`\n`)야 합니다. 이스케이프된 줄 바꿈을 저장하는 것만으로는 나중에 올바르게 검색되지 않기 때문입니다.

> **Note**
>
> 보안 스토리지를 사용하는 경우 이러한 자격 증명을 파일 시스템 대신 보안 스토리지 위치에 저장합니다. 자세한 내용은 보안 스토리지를 위한 사용자 지정 인증서 핸들러 생성 단원을 참조하십시오.

6단계: 디바이스 구성 파일 생성

고유한 디바이스 식별자, 인증서 위치 및 프로비저닝 설정이 포함된 구성 파일을 생성합니다. SDK는 허브 온보딩 중에이 파일을 사용하여 디바이스를 인증하고, 프로비저닝 상태를 관리하고, 연결 설정을 저장합니다.

> **Note**
>
> 각 허브 디바이스에는 고유한 디바이스별 값이 있는 자체 구성 파일이 필요합니다.

다음 절차에 따라 구성 파일을 생성하거나 수정하고 허브에 복사합니다.

- 구성 파일(플릿 프로비저닝)을 생성하거나 수정합니다.

  디바이스 구성 파일에서 다음 필수 필드를 구성합니다.

  - 인증서 경로
    1. `iot_claim_cert_path`: 클레임 인증서의 위치(`claim_cert.pem`)
    2. `iot_claim_pk_path`: 프라이빗 키의 위치(`claim_pk.key`)
    3. 보안 스토리지 인증서 핸들러를 구현할 때 두 필드 모두에 `SECURE_STORAGE` 사용
  - 연결 설정

1. fp_template_name: 이전의 ProvisioningProfile 이름입니다.
2. endpoint_url: RegisterCustomEndpoint API 응답의 관리형 통합 엔드포인트 URL입니다(리전의 모든 디바이스에 대해 동일).

- 디바이스 식별자
  1. SN: CreateManagedThing API 호출과 일치하는 디바이스 일련 번호(디바이스당 고유)
  2. UPCCreateManagedThing API 직접 호출의 범용 제품 코드(이 제품의 모든 디바이스에서 동일)

```
{
    "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"
    }
}
```

구성 파일의 내용

iotmi_config.json 파일의 내용을 검토합니다.

내용

Key(키)	값	고객이 추가했나요?	Notes
iot_provisioning_method	FLEET_PROVISIONING	예	사용할 프로비저닝 방법을 지정합니다.
iot_claim_cert_path	또는를 지정하는 파일 경로입니다SECURE_ST	예	또는를 사용할 파일 경로를 지정합니다SECURE_STORAGE .

Key(키)	값	고객이 추가했나요?	Notes
	ORAGE . 예: /data/aws/iotmi/certs/claim_cert.pem		
iot_claim_pk_path	또는를 지정하는 파일 경로입니다SECURE_STORAGE . 예: /data/aws/iotmi/certs/claim_pk.pem	예	또는를 사용할 파일 경로를 지정합니다SECURE_STORAGE .
fp_template_name	플릿 프로비저닝 템플릿 이름은 이전에 ProvisioningProfile 사용된의 이름과 같아야 합니다.	예	이전에 ProvisioningProfile 사용된의 이름과 동일
endpoint_url	관리형 통합을 위한 엔드포인트 URL입니다.	예	디바이스는이 URL을 사용하여 관리형 통합 클라우드에 연결합니다. 이 정보를 얻으려면 RegisterCustomEndpoint API를 사용합니다.
SN	디바이스 일련 번호입니다. 예를 들어 AIDACKCEVSQ6C2EXAMPLE 입니다.	예	각 디바이스에 대해이 고유한 정보를 제공해야 합니다.
UPC	디바이스 범용 제품 코드. 예를 들어 841667145075 입니다.	예	디바이스에 대해이 정보를 제공해야 합니다.
managed_thing_id	관리형 사물의 ID입니다.	아니요	이 정보는 나중에 허브 프로비저닝 후 온보딩 프로세스에 의해 추가됩니다.

Key(키)	값	고객이 추가했나요?	Notes
iot_provisioning_state	프로비저닝 상태입니다.	예	프로비저닝 상태는 로 설정해야 합니다NOT_PROVISIONED .
iot_permanent_cert_path	IoT 인증서 경로입니다. 예를 들어 /data/aws/iotmi/iot_cert.pem 입니다.	아니요	이 정보는 나중에 허브 프로비저닝 후 온보딩 프로세스에 의해 추가됩니다.
iot_permanent_pk_path	IoT 프라이빗 키 파일 경로입니다. 예를 들어 /data/aws/iotmi/iot_pk.pem 입니다.	아니요	이 정보는 나중에 허브 프로비저닝 후 온보딩 프로세스에 의해 추가됩니다.
client_id	MQTT 연결에 사용할 클라이언트 ID입니다.	아니요	이 정보는 나중에 다른 구성 요소가 사용할 수 있도록 허브 프로비저닝 후 온보딩 프로세스에 의해 추가됩니다.
mqtt_keep_alive_interval	범위는 30~1200이고 단위는 초 단위입니다. 기본값은 300입니다.	예	이를 사용하여 MQTT 연결에 대한 연결 유지 간격을 설정합니다.
event_manager_upper_bound	기본값은 500입니다.	아니요	이 정보는 나중에 다른 구성 요소가 사용할 수 있도록 허브 프로비저닝 후 온보딩 프로세스에 의해 추가됩니다.

7단계: 구성 파일을 허브에 복사

구성 파일을 /data/aws/iotmi/config 또는 사용자 지정 디렉터리 경로에 복사합니다. 온보딩 프로세스 중에 바이HubOnboarding너리에 대한이 경로를 제공합니다.

플릿 프로비저닝의 경우

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

디바이스 온보딩 및 허브에서 작동

관리형 사물을 생성하고 허브에 연결하여 관리형 통합 허브에 온보딩되도록 디바이스를 설정합니다. 간단한 설정 또는 사용자 안내 설정을 통해 디바이스를 허브에 온보딩할 수 있습니다.

주제

- 디바이스 온보딩 및 운영을 위한 간단한 설정
- 디바이스 온보딩 및 운영을 위한 사용자 안내 설정

디바이스 온보딩 및 운영을 위한 간단한 설정

관리형 사물을 생성하고 허브에 연결하여 관리형 통합 허브에 온보딩되도록 디바이스를 설정합니다. 이 섹션에서는 간단한 설정을 사용하여 디바이스 온보딩 프로세스를 완료하는 단계를 설명합니다.

사전 조건

디바이스 온보딩을 시도하기 전에 다음 단계를 완료합니다.

- 허브 디바이스를 관리형 통합 허브에 온보딩합니다.
- 관리형 통합 AWS CLI 명령 참조 AWS CLI 에서의 최신 버전 설치
- DEVICE_LIFE_CYCLE 이벤트 알림을 구독합니다.

설정 단계

- 1단계: 자격 증명 로커 생성
- 2단계: 허브에 자격 증명 로커 추가
- 3단계: 자격 증명으로 관리형 사물을 생성합니다.

- 4단계: 디바이스를 연결하고 상태를 확인합니다.
- 5단계: 디바이스 기능 가져오기
- 6단계: 관리형 사물에 명령 전송
- 7단계: 허브에서 관리형 사물 제거

1단계: 자격 증명 로커 생성

디바이스에 대한 자격 증명 로커를 생성합니다.

자격 증명 로커를 생성하려면

- `create-credential-locker` 명령을 사용합니다. 이 명령을 실행하면 Wi-Fi 설정 키 페어 및 디바이스 인증서를 포함한 모든 제조 리소스가 생성됩니다.

  create-credential-locker 예

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

  응답:

  ```
  {
    "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"
  }
  ```

  자세한 내용은 관리형 통합 명령 참조의 create-credential-locker 명령을 참조하세요 AWS CLI .

2단계: 허브에 자격 증명 로커 추가

허브에 자격 증명 로커를 추가합니다.

허브에 자격 증명 로커를 추가하려면

- 다음 명령을 사용하여 허브에 자격 증명 로커를 추가합니다.

  ```
  aws iotmi --region AWS_REGION --endpoint AWS_ENDPOINT update-managed-thing \
  ```

```
--identifier "HUB_MANAGED_THING_ID" --credential-locker-id "LOCKER_ID"
```

3단계: 자격 증명으로 관리형 사물을 생성합니다.

디바이스의 자격 증명을 사용하여 관리형 사물을 생성합니다. 각 디바이스에는 자체 관리형 사물이 필요합니다.

관리형 사물을 생성하려면

- `create-managed-thing` 명령을 사용하여 디바이스에 대한 관리형 사물을 생성합니다.

  create-managed-thing 예

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

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

> **Note**
>
> Z-wave 및 Zigbee 디바이스에는 별도의 명령이 있습니다.

  응답:

```
{
  "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"
}
```

  자세한 내용은 관리형 통합 명령 참조의 create-managed-thing 명령을 참조하세요 AWS CLI .

4단계: 디바이스를 연결하고 상태를 확인합니다.

디바이스를 연결하고 상태를 확인합니다.

- `get-managed-thing` 명령을 사용하여 디바이스의 상태를 확인합니다. 관리형 사물의 ProvisioningStatus가 활성화되어 있어야 합니다. ProvisioningStatus에 대한 자세한 내용은 디바이스 프로비저닝을 참조하세요.

  get-managed-thing 예

```
#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"
```

  응답:

```
{
  "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"
}
```

자세한 내용은 관리형 통합 명령 참조의 get-managed-thing 명령을 참조하세요 AWS CLI .

5단계: 디바이스 기능 가져오기

`get-managed-thing-capabilities` 명령을 사용하여 엔드포인트 ID를 가져오고 디바이스에 대해 가능한 작업 목록을 봅니다.

디바이스의 기능을 가져오려면

- `get-managed-thing-capabilities` 명령을 사용하고 엔드포인트 ID를 기록해 둡니다.

  get-managed-thing-capabilties 예

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

  응답:

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

자세한 내용은 관리형 통합 명령 참조의 get-managed-thing-capabilities 명령을 참조하세요 AWS CLI .

6단계: 관리형 사물에 명령 전송

send-managed-thing-command 명령을 사용하여 토글 작업 명령을 관리형 사물로 보냅니다.

관리형 사물에 명령을 보내려면

- send-managed-thing-command 명령을 사용하여 관리형 사물에 명령을 보냅니다.

  send-managed-thing-command 예

```
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**
>
> 이 예제에서는 jq cli를에 사용하지만 전체 `endpointId` 문자열을 전달할 수도 있습니다.

응답:

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

자세한 내용은 관리형 통합 명령 참조의 send-managed-thing-command 명령을 참조하세요 AWS CLI .

7단계: 허브에서 관리형 사물 제거

관리형 사물을 제거하여 허브를 정리합니다.

관리형 사물을 삭제하려면

- `delete-managed-thing` 명령을 사용하여 디바이스 허브에서 관리형 사물을 제거합니다.

  delete-managed-thing 예

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

  자세한 내용은 관리형 통합 명령 참조의 delete-managed-thing 명령을 참조하세요 AWS CLI .

  > **Note**
  >
  > 디바이스가 `DELETE_IN_PROGRESS` 상태로 멈춘 경우 `--force` 플래그를에 추가합니다`delete-managed-thing command`.

> **Note**
>
> Z파 디바이스의 경우 명령을 실행한 후 디바이스를 페어링 모드로 전환해야 합니다.

디바이스 온보딩 및 운영을 위한 사용자 안내 설정

허브에 연결하는 관리형 사물을 생성하여 관리형 통합 허브에 온보딩되도록 디바이스를 설정합니다. 이 섹션에서는 사용자 안내 설정을 사용하여 디바이스 온보딩 프로세스를 완료하는 단계를 설명합니다.

사전 조건

디바이스 온보딩을 시도하기 전에 다음 단계를 완료합니다.

- 허브 디바이스를 관리형 통합 허브에 온보딩합니다.
- 관리형 통합 AWS CLI 명령 참조 AWS CLI 에서의 최신 버전 설치
- DEVICE_DISCOVERY-STATUS 이벤트 알림을 구독합니다.

사용자 안내 설정 단계

- 사전 조건: Z Wave 디바이스에서 페어링 모드 활성화
- 1단계: 디바이스 검색 시작
- 2단계: 검색 작업 ID 쿼리
- 3단계: 디바이스에 대한 관리형 사물 생성
- 4단계: 관리형 사물 쿼리
- 5단계: 관리형 사물 기능 가져오기
- 6단계: 관리형 사물에 명령 전송
- 7단계: 관리형 사물 상태 확인
- 8단계: 허브에서 관리형 사물 제거

사전 조건: Z Wave 디바이스에서 페어링 모드 활성화

Z파 디바이스에서 페어링 모드를 활성화합니다. 페어링 모드는 Z-Wave 디바이스마다 다를 수 있으므로 페어링 모드를 올바르게 설정하려면 디바이스의 지침을 참조하세요. 일반적으로 사용자가 눌러야 하는 버튼입니다.

1단계: 디바이스 검색 시작

허브에 대한 디바이스 검색을 시작하여 디바이스를 온보딩하는 데 사용되는 검색 작업 ID를 가져옵니다.

디바이스 검색을 시작하려면

- start-device-discovery 명령을 사용하여 검색 작업 ID를 가져옵니다.

  start-device-discovery 예

```
#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 \
```

  응답:

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

> **Note**
>
> Z-wave 및 Zigbee 디바이스에는 별도의 명령이 있습니다.

자세한 내용은 관리형 통합 명령 참조의 start-device-discovery API를 AWS CLI 참조하세요.

2단계: 검색 작업 ID 쿼리

`list-discovered-devices` 명령을 사용하여 디바이스의 인증 구성 요소를 가져옵니다.

검색 작업 ID를 쿼리하려면

- list-discovered-devices 명령과 함께 검색 작업 ID를 사용하여 디바이스의 인증 구성 요소를 가져옵니다.

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

응답:

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

3단계: 디바이스에 대한 관리형 사물 생성

`create-managed-thing` 명령을 사용하여 디바이스에 대한 관리형 사물을 생성합니다. 각 디바이스에는 자체 관리형 사물이 필요합니다.

관리형 사물을 생성하려면

- `create-managed-thing` 명령을 사용하여 디바이스에 대한 관리형 사물을 생성합니다.

  create-managed-thing 예

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

응답:

```
{
    "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"
  }
```

자세한 내용은 관리형 통합 명령 참조의 create-managed-thing 명령을 참조하세요 AWS CLI .

4단계: 관리형 사물 쿼리

get-managed-thing 명령을 사용하여 관리형 사물이 활성화되었는지 확인할 수 있습니다.

관리형 사물을 쿼리하려면

- get-managed-thing 명령을 사용하여 관리형 사물의 프로비저닝 상태가 로 설정되어 있는지 확인합니다ACTIVATED. 프로비저닝 상태에 대한 자세한 내용은 디바이스 프로비저닝을 참조하세요.

  get-managed-thing 예

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

  응답:

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

자세한 내용은 관리형 통합 명령 참조의 get-managed-thing 명령을 참조하세요 AWS CLI .

5단계: 관리형 사물 기능 가져오기

를 사용하여 관리형 사물의 사용 가능한 작업 목록을 볼 수 있습니다`get-managed-thing-capabiltiies`.

디바이스의 기능을 가져오려면

- `get-managed-thing-capabilities` 명령을 사용하여 엔드포인트 ID를 가져옵니다. 또한 가능한 작업 목록을 기록해 둡니다.

  get-managed-thing-capabilities 예

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

  응답:

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

자세한 내용은 관리형 통합 명령 참조의 get-managed-thing-capabilities AWS CLI명령을 참조하세요.

6단계: 관리형 사물에 명령 전송

send-managed-thing-command 명령을 사용하여 토글 작업 명령을 관리형 사물로 보낼 수 있습니다.

토글 작업을 사용하여 관리형 사물에 명령을 보냅니다.

- `send-managed-thing-command` 명령을 사용하여 토글 작업 명령을 보냅니다.

  send-managed-thing-command 예

```
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**
>
> 이 예제에서는 jq cli를에 사용하지만 전체 endpointId 문자열을 전달할 수도 있습니다.

응답:

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

자세한 내용은 관리형 통합 명령 참조의 send-managed-thing-command 명령을 참조하세요 AWS CLI .

7단계: 관리형 사물 상태 확인

관리형 사물의 상태를 확인하여 토글 작업이 성공했는지 확인합니다.

관리형 사물의 디바이스 상태를 확인하려면

- get-managed-thing-state 명령을 사용하여 토글 작업이 성공했는지 확인합니다.

  get-managed-thing-state 예

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

응답:

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

자세한 내용은 관리형 통합 명령 참조의 get-managed-thing-state AWS CLI 명령을 참조하세요.

8단계: 허브에서 관리형 사물 제거

관리형 사물을 제거하여 허브를 정리합니다.

관리형 사물을 삭제하려면

- delete-managed-thing 명령을 사용하여 관리형 사물을 제거합니다.

  delete-managed-thing 예

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

자세한 내용은 관리형 통합 명령 참조의 delete-managed-thing 명령을 참조하세요 AWS CLI .

> **Note**
>
> 디바이스가 `DELETE_IN_PROGRESS` 상태로 멈춘 경우 `delete-managed-thing` 명령에 `--force` 플래그를 추가합니다.

> **Note**
>
> Z파 디바이스의 경우 명령을 실행한 후 디바이스를 페어링 모드로 전환해야 합니다.

보안 스토리지를 위한 사용자 지정 인증서 핸들러 생성

디바이스 인증서 관리는 관리형 통합 허브를 온보딩할 때 매우 중요합니다. 인증서는 기본적으로 파일 시스템에 저장되지만 보안 강화 및 유연한 자격 증명 관리를 위해 사용자 지정 인증서 핸들러를 생성할 수 있습니다.

관리형 통합 엔드 디바이스 SDK는 공유 객체(.so) 라이브러리로 구현할 수 있는 보안 스토리지 인터페이스에 대한 인증서 핸들러를 제공합니다. 인증서를 읽고 쓰도록 보안 스토리지 구현을 구축한 다음 런타임에 라이브러리 파일을 HubOnboarding 프로세스에 연결합니다.

API 정의 및 구성 요소

다음 `secure_storage_cert_handler_interface.hpp` 파일을 검토하여 구현을 위한 API 구성 요소 및 요구 사항을 이해합니다.

주제

- API 정의
- 핵심 구성 요소

API 정의

`secure_storage_cert_hander_interface.hpp`의 콘텐츠

```
/*
```

```
    * 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
```

핵심 구성 요소

- CERT_TYPE_T - 허브에 있는 다양한 유형의 인증서입니다.
  - 클레임 - 원래 허브에 있는 클레임 인증서는 영구 인증서로 교환됩니다.
  - DHA - 지금은 사용되지 않습니다.
  - 영구 - 관리형 통합 엔드포인트와 연결하기 위한 영구 인증서입니다.
- read_cert_and_private_key - (FUNCTION TO BE IMPLEMENTED) 참조 입력에 대한의 인증서 및 키 값을 읽습니다. 이 함수는 CLAIM 및 PERMANENT 인증서를 모두 읽을 수 있어야 하며 위에서 언급한 인증서 유형으로 구분됩니다.
- write_permanent_cert_and_private_key - (FUNCTION TO BE IMPLEMENTED) 영구 인증서와 키 값을 원하는 위치에 씁니다.

예제 빌드

내부 구현 헤더를 퍼블릭 인터페이스(`secure_storage_cert_handler_interface.hpp`)와 분리하여 깨끗한 프로젝트 구조를 유지합니다. 이러한 분리를 통해 인증서 핸들러를 구축하는 동안 퍼블릭 및 프라이빗 구성 요소를 관리할 수 있습니다.

> **Note**
>
> 공개`secure_storage_cert_handler_interface.hpp`로 선언합니다.

주제

- 프로젝트 구조
- 인터페이스 상속
- 구현
- CMakeList.txt

프로젝트 구조

인터페이스 상속

인터페이스를 상속하는 구체적인 클래스를 생성합니다. 빌드 시 프라이빗 헤더와 퍼블릭 헤더를 쉽게 구분할 수 있도록 별도의 디렉터리에서이 헤더 파일과 기타 파일을 숨깁니다.

```
#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
```

구현

위에 정의된 스토리지 클래스를 구현합니다`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;
 }
```

인터페이스에 정의된 팩토리 함수를 구현합니다src/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
)
```

사용법

컴파일 후 `libSecureStorageCertHandler.so` 공유 객체 라이브러리 파일과 관련 심볼 링크가 있습니다. 라이브러리 파일과 심볼 링크를 모두 HubOnboarding 바이너리에서 예상되는 라이브러리 위치에 복사합니다.

주제

- 주요 고려 사항
- 보안 스토리지 사용

주요 고려 사항

- 사용자 계정에 HubOnboarding 바이너리와 `libSecureStorageCertHandler.so` 라이브러리 모두에 대한 읽기 및 쓰기 권한이 있는지 확인합니다.
- `secure_storage_cert_handler_interface.hpp`를 유일한 퍼블릭 헤더 파일로 유지합니다. 다른 모든 헤더 파일은 프라이빗 구현에 남아 있어야 합니다.
- 공유 객체 라이브러리 이름을 확인합니다. 를 빌드하는 동안 `libSecureStorageCertHandler.so` HubOnboarding은 파일 이름에와 같은 특정 버전이 필요할 수 있습니다`libSecureStorageCertHandler.so.1.0`. `ldd` 명령을 사용하여 라이브러리 종속성을 확인하고 필요에 따라 심볼 링크를 생성합니다.
- 공유 라이브러리 구현에 외부 종속성이 있는 경우 디렉터리와 같이 HubOnboarding이 액세스할 수 있는 `/usr/lib or the iotmi_common` 디렉터리에 저장합니다.

보안 스토리지 사용

`iot_claim_cert_path` 및를 모두 `iot_claim_pk_path`로 설정하여 `iotmi_config.json` 파일을 업데이트합니다**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"
  }
}
```

사용자 지정 프로토콜 플러그인

사용자 지정 프로토콜 플러그인을 사용하여 독점 IoT 프로토콜을 AWS IoT Device Management 에코시스템의 관리형 통합에 통합할 수 있습니다. 잘 정의된 SDK 인터페이스를 통해 디바이스를 온보딩하고, 기능을 정의하고, 실시간 제어 흐름을 처리하는 동시에 관리형 통합 및 허브 SDK 구성 요소와의 완전한 호환성을 유지할 수 있습니다.

다음 목록에서는 사용자 지정 프로토콜 플러그인의 주요 기능에 대해 설명합니다.

데이터 모델 사용자 지정

프로비저닝 흐름 중에 자체 AWS 데이터 모델 스키마를 정의하고 관리형 통합에 업로드합니다. 워크플로에서 나중에 이러한 스키마를 사용할 수 있습니다.

유연한 플러그인 구현

- 를 사용하여 자체 플러그인 구성 요소를 빌드합니다Hub SDK 클라이언트.
- 프로비저닝 및 제어와 같은 다양한 기능에 대해 별도의 플러그인을 구현하거나 둘 다에 대해 통합 클라이언트를 생성합니다.
- 분리되고 개발 친화적인 코드 로직 구현을 위해 관리형 통합 자산과 미들웨어 스택과 같은 자체 자산 간에 명확한 경계를 유지합니다.

이전 버전과의 호환성

기존 고객의 경우 기존 라디오 유형을 그대로 유지하면서 새 사용자 지정 프로토콜을 원활하게 온보딩합니다.

다음 다이어그램은 사용자 지정 프로토콜 플러그인 아키텍처를 보여줍니다.

Hub SDK 클라이언트

Hub SDK 클라이언트 라이브러리는 관리형 통합 Hub SDK와 동일한 허브에서 실행되는 자체 프로토콜 스택 간의 인터페이스 역할을 합니다. 프로토콜 스택과 Device Hub SDK 구성 요소의 상호 작용을 용이하게 하기 위해 퍼블릭 APIs 세트를 노출합니다. 사용 사례에는 사용자 지정 플러그인 제어, 사용자 지정 플러그인 프로비저너 및 로컬 컨트롤러가 포함됩니다.

주제

- 관리형 통합 Hub SDK 가져오기
- Hub SDK 툴킷 정보
- Hub SDK 클라이언트를 사용하여 사용자 지정 애플리케이션 생성
- 사용자 지정 애플리케이션 실행
- Hub SDK 클라이언트 API 참조
- 데이터 타입

관리형 통합 Hub SDK 가져오기

Hub SDK 클라이언트는 관리형 통합 SDK와 함께 제공됩니다. 허브 SDK에 액세스하려면 관리형 통합 콘솔에서 문의하세요.

Hub SDK 툴킷 정보

다운로드 후 애플리케이션에서 사용할 수 있는 모든 퍼블릭 헤더 파일과 `.so` 파일이 포함된 `IotMI-DeviceSDK-Toolkit` 폴더가 표시됩니다. 또한 관리형 통합 팀은 데모`main.cpp`용 예제와 직접 실행할 수 `bin/` 있는 데모 애플리케이션 바이너리를 제공합니다. 필요에 따라 이를 애플리케이션의 시작점으로 사용할 수 있습니다.

Hub SDK 클라이언트를 사용하여 사용자 지정 애플리케이션 생성

다음 단계에 따라 사용자 지정 애플리케이션을 생성합니다.

1. 애플리케이션에 헤더 파일(.h)과 공유 객체 파일(.so)을 포함합니다.

   애플리케이션에 퍼블릭 헤더 파일(.h)과 공유 객체 파일(.so)을 포함해야 합니다. .so 파일의 경우 lib 폴더에 배치할 수 있습니다. 최종 레이아웃은 다음과 유사합니다.

   ```
   ### 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. 기본 애플리케이션에서 Hub SDK 클라이언트를 생성합니다.

   a. 기본 애플리케이션에서 요청을 처리하는 데 사용하기 전에 먼저 Hub SDK 클라이언트를 초기화해야 합니다. 를 사용하여 클라이언트를 구성할 수 있습니다`clientId`.

   b. 클라이언트가 있으면 관리형 통합 디바이스 SDK에 연결할 수 있습니다.

      다음은 Hub SDK 클라이언트를 생성하고 연결하는 방법의 예입니다.

      ```
      #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*는 사용자 가이드 설정의 start-device-discovery 또는 간단한 설정 프로비저닝 흐름의 create-managed-thing에 지정한 것과 동일해야 합니다.

3. 다음 단계를 수행하여 게시하고 구독합니다.

a. 연결이 설정되면 이제 관리형 통합 Hub SDK에서 들어오는 태스크를 구독할 수 있습니다. 수신 작업은 제어 작업 또는 프로비저닝 작업일 수 있습니다. 또한 수신된 작업에 대한 자체 콜백 함수와 자체 추적을 위한 사용자 지정 컨텍스트를 정의해야 합니다.

```
    // 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. 연결이 설정되면 이제 애플리케이션의 요청을 관리형 통합 Hub SDK에 게시할 수 있습니다. 비즈니스 목적에 따라 페이로드가 다른 고유한 작업 메시지 유형을 정의할 수 있습니다. 요청에는 구독 흐름과 마찬가지로 제어 요청과 프로비저닝 요청이 모두 포함될 수 있습니다. 마지막으로에 주소를 할당`rspPayload`하여 관리형 통합 Hub SDK에서 동기화된 방식으로 응답을 받을 수 있습니다.

```
// 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. 자체를 빌드`CMakeLists.txt`하고 거기에서 애플리케이션을 빌드합니다. 최종 출력은 다음과 같은 실행 가능한 바이너리일 수 있습니다. `MyFirstApplication`

사용자 지정 애플리케이션 실행

사용자 지정 애플리케이션을 실행하기 전에 다음 단계를 완료하여 허브를 설정하고 관리형 통합 Hub SDK를 시작합니다.

- 의 온보딩 지침을 따릅니다허브를 관리형 통합에 온보딩.
- 에 설명된 설치 프로세스를 완료합니다관리형 통합 Hub SDK 설치 및 검증.

사전 조건이 충족되면 사용자 지정 애플리케이션을 실행할 수 있습니다. 예:

```
./MyFirstApplication
```

Important

자체 애플리케이션을 시작하려면에 나열된 시작 스크립트를 스크립트스크립트를 사용하여 Hub SDK 배포로 수동으로 업데이트해야 합니다. 주문이 중요하므로 주문을 변경하지 마세요. 다음을 업데이트합니다. 변경 사항

```
./IotMI-DeviceSDK-Toolkit/bin/DeviceSDKClientDemo >> $LOGS_DIR/
logDeviceSDKClientDemo_logs.txt &
```

아래로 변경합니다.

```
./MyFirstApplication >> $LOGS_DIR/MyFirstApplication_logs.txtt &
```

Hub SDK 클라이언트 API 참조

Hub SDK 클라이언트(DeviceSDKClient 클래스)는 사용자 지정 애플리케이션이 관리형 통합 Device SDK와 상호 작용할 수 있는 인터페이스를 제공합니다. 이 클라이언트를 사용하여 다음을 수행할 수 있습니다.

- 관리형 통합 구성 요소에서 프로비저닝 관련 및 제어 관련 작업을 구독합니다.
- 관리형 통합 구성 요소에 프로비저닝 관련 및 제어 관련 요청을 게시합니다.

AWS IoT Device Management APIs의 관리형 통합에 대한 자세한 내용은 란 AWS Lambda 무엇입니까?를 참조하십시오.

주제

클라이언트 초기화

사용을 시작하려면 클라이언트 ID로 DeviceSDKClient초기화합니다.

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

이렇게 하면가 지정된 새 DeviceSDKClient 인스턴스가 생성됩니다clientId. 는 관리형 통합에 등록한 것과 일치해야 clientId 합니다.

파라미터

clientId (문자열) -이 인스턴스의 클라이언트 ID입니다.

```
connect()
```

DeviceSDKClient 인스턴스를 관리형 통합에 연결합니다.

반환

- IOTMI_STATUS_OK - 연결에 성공했습니다.
- IOTMI_STATUS_CUSTOM_PLUGIN_CONNECTION_ERROR - 관리형 통합에 연결하는 동안 오류가 발생했습니다.

프로비저닝 작업 구독

이러한 방법을 사용하여 관리형 통합 구성 요소에서 프로비저닝 관련 작업을 구독할 수 있습니다.

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

관리형 통합 구성 요소에서 디바이스 온보딩 및 프로비저닝 해제와 같은 프로비저닝 관련 작업을 구독합니다.

파라미터

- `callback` (DeviceSDKClient_SubscriberCallback) - 작업이 수신될 때 실행되는 콜백 함수입니다.
- `context` (char*) - 콜백 함수에 전달되는 사용자 지정 컨텍스트입니다.

반환

- `IOTMI_STATUS_OK` - 구독이 성공했습니다.
- `IOTMI_STATUS_CUSTOM_PLUGIN_CLIENT_NOT_CONNECTED` - DeviceSDKClient 인스턴스가 관리형 통합에 연결되지 않았습니다.
- `IOTMI_STATUS_CUSTOM_PLUGIN_SUBSCRIBE_ERROR` - 작업을 구독하는 동안 오류가 발생했습니다.

프로비저닝 작업 게시

이러한 방법을 사용하여 관리형 통합 구성 요소에 프로비저닝 관련 요청을 게시합니다.

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

관리형 통합 구성 요소에 프로비저닝 관련 요청을 게시합니다. 예: 디바이스 온보딩 이벤트 또는 프로비저닝 해제 상태

파라미터

`request` (DataModel::iotmi_client_request_t) - 세부 정보가 포함된 요청 구조에 대한 포인터입니다.

반환

- `IOTMI_STATUS_OK` - 요청이 성공적으로 게시되었습니다.
- `IOTMI_STATUS_CUSTOM_PLUGIN_CLIENT_NOT_CONNECTED` - `DeviceSDKClient` 인스턴스가 관리형 통합에 연결되어 있지 않습니다.

- IOTMI_STATUS_INVALID_PARAMETER - 요청의 하나 이상의 파라미터가 잘못되었습니다.
- IOTMI_STATUS_INVALID_JSON_OBJECT - 요청 페이로드가 유효한 JSON 객체가 아닙니다.
- IOTMI_STATUS_NO_MEMORY - 메모리 할당 오류가 발생했습니다.

제어 작업 구독

이러한 방법을 사용하여 관리형 통합 구성 요소에서 제어 관련 작업을 구독할 수 있습니다.

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

관리형 통합 구성 요소에서 제어 관련 작업(예: 디바이스 제어 요청)을 구독합니다.

파라미터

- callback (DeviceSDKClient_SubscriberCallback) - 작업이 수신될 때 실행되는 콜백 함수입니다.
- context (char) - 콜백 함수에 전달되는 사용자 지정 컨텍스트입니다.

반환

- IOTMI_STATUS_OK - 구독이 성공했습니다.
- IOTMI_STATUS_CUSTOM_PLUGIN_CLIENT_NOT_CONNECTED - DeviceSDKClient 인스턴스가 관리형 통합에 연결되어 있지 않습니다.
- IOTMI_STATUS_CUSTOM_PLUGIN_SUBSCRIBE_ERROR - 작업을 구독하는 동안 오류가 발생했습니다.

제어 작업 게시

이러한 방법을 사용하여 관리형 통합 구성 요소에 제어 관련 요청을 게시합니다.

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

관리형 통합 구성 요소에 제어 관련 요청을 게시합니다. 요청되지 않은 이벤트, 명령 요청 또는 디바이스 상태 쿼리를 예로 들 수 있습니다.

파라미터

`request` (DataModel::iotmi_client_request_t) - 세부 정보가 포함된 요청 구조에 대한 포인터입니다.

반환

- `IOTMI_STATUS_OK` - 요청이 성공적으로 게시되었습니다.
- `IOTMI_STATUS_CUSTOM_PLUGIN_CLIENT_NOT_CONNECTED` - DeviceSDKClient 인스턴스가 관리형 통합에 연결되지 않았습니다.
- `IOTMI_STATUS_INVALID_PARAMETER` - 요청의 하나 이상의 파라미터가 잘못되었습니다.
- `IOTMI_STATUS_INVALID_JSON_OBJECT` - 요청 페이로드가 유효한 JSON 객체가 아닙니다.
- `IOTMI_STATUS_NO_MEMORY` - 메모리 할당 오류가 발생했습니다.

로깅 기능

이러한 방법을 사용하여 관리형 통합이 제공하는 로깅 기능을 구현합니다.

로거 초기화

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

로깅 기능을 사용하기 전에 로거를 초기화해야 합니다.

파라미터

`logger_name` - 지정한 로거 이름입니다. 기본값은 다음과 같습니다. `MyApplication`

매크로 로깅

`LOGGER_LOGD(...)`

애플리케이션에서 DEBUG 수준 로깅에이 매크로를 사용합니다.

`LOGGER_LOGI(...)`

애플리케이션에서 INFO 수준 로깅에이 매크로를 사용합니다.

`LOGGER_LOGW(...)`

애플리케이션에서 WARN 수준 로깅에이 매크로를 사용합니다.

`LOGGER_LOGE(...)`

애플리케이션에서 오류 수준 로깅에이 매크로를 사용합니다.

> **Note**
>
> 로깅 기능에 대한 자세한 내용은 Hub 로깅 설명서를 참조하세요. 사용자 지정 프로토콜 플러그인은 관리형 통합이 제공하는 모든 로깅 기능을 완벽하게 지원합니다.

기타 API

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

DeviceSDKClient 인스턴스와 연결된 클라이언트 ID를 반환합니다.

반환

클라이언트 ID입니다.

데이터 타입

이 섹션에서는 사용자 지정 프로토콜 플러그인에 사용되는 데이터 유형을 정의합니다.

iotmi_client_request_t

관리형 통합 구성 요소에 게시할 요청을 나타냅니다.

messageType

메시지 유형(CommonTypes::C2MIMessageType). 다음 목록은 유효한 값을 보여줍니다.

- `C2MI_DEVICE_ONBOARDED`: 관련 페이로드가 있는 디바이스 온보딩 메시지를 나타냅니다.
- `C2MI_DE_PROVISIONING_PRE_ASSOCIATED_COMPLETE`: 사전 연결된 디바이스에 대한 프로비저닝 해제 작업 완료 알림을 나타냅니다.
- `C2MI_DE_PROVISIONING_ACTIVATED_COMPLETE`: 활성화된 디바이스에 대한 프로비저닝 해제 작업 완료 알림을 나타냅니다.
- `C2MI_DE_PROVISIONING_COMPLETE_RESPONSE`: 프로비저닝 해제 작업 완료 응답을 나타냅니다.

- C2MI_CONTROL_EVENT: 잠재적인 디바이스 상태 변경이 있는 제어 이벤트를 나타냅니다.
- C2MI_CONTROL_SEND_COMMAND: 로컬 컨트롤러의 제어 명령을 나타냅니다.
- C2MI_CONTROL_SEND_DEVICE_STATE_QUERY: 로컬 컨트롤러의 제어 디바이스 상태 쿼리를 나타냅니다.

reqPayload

요청 페이로드, 일반적으로 JSON 형식 문자열입니다.

rspPayload

관리형 통합 구성 요소로 채워지는 응답 페이로드입니다.

iotmi_클라이언트_이벤트_t

관리형 통합 구성 요소에서 수신된 이벤트를 나타냅니다.

event_id

이벤트의 고유 식별자입니다.

length

이벤트 데이터의 길이입니다.

data

를 포함하여 이벤트 데이터에 대한 포인터입니다messageType. 다음 목록은 가능한 값을 보여줍니다.

- C2MI_PROVISION_UGS_TASK: UGS 흐름에 대한 프로비저닝 작업을 나타냅니다.
- C2MI_PROVISION_SS_TASK: SimpleSetup 흐름에 대한 프로비저닝 작업을 나타냅니다.
- C2MI_DE_PROVISION_PRE_ASSOCIATED_TASK: 사전 연결된 디바이스에 대한 프로비저닝 해제 작업을 나타냅니다.
- C2MI_DE_PROVISION_ACTIVATED_TASK: 활성화된 디바이스에 대한 프로비저닝 해제 작업을 나타냅니다.
- C2MI_DEVICE_ONBOARDED_RESPONSE: 디바이스 온보딩 응답을 나타냅니다.
- C2MI_CONTROL_TASK: 제어 작업을 나타냅니다.
- C2MI_CONTROL_EVENT_NOTIFICATION: 로컬 컨트롤러에 대한 제어 이벤트 알림을 나타냅니다.

ctx

이벤트와 연결된 사용자 지정 컨텍스트입니다.

허브 제어

허브 제어는 관리형 통합 엔드 디바이스 SDK의 확장으로, 이를 통해 Hub SDK의 `MQTTProxy` 구성 요소와 인터페이스할 수 있습니다. 허브 제어를 사용하면 엔드 디바이스 SDK를 사용하여 코드를 구현하고 관리형 통합 클라우드를 통해 허브를 별도의 디바이스로 제어할 수 있습니다. 허브 제어 SDK는 Hub SDK에 로 레이블이 지정된 별도의 패키지로 제공됩니다`iot-managed-integrations-hub-control-x.x.x`.

주제

사전 조건

허브 제어를 설정하려면 다음이 필요합니다.

- Hub SDK 버전 0.4.0 이상에 온보딩된 허브입니다.
- 에서 최신 버전의 End Device SDK를 다운로드합니다 AWS Management Console.
- 허브에서 실행되는 MQTT 프록시 구성 요소, 버전 0.5.0 이상.

엔드 디바이스 SDK 구성 요소

End 디바이스 SDK에서 다음 구성 요소를 사용합니다.

- 데이터 모델의 코드 생성기
- 데이터 모델 핸들러

Hub SDK에는 이미 온보딩 프로세스와 클라우드에 대한 연결이 있으므로 다음 구성 요소가 필요하지 않습니다.

- 프로비저닝 담당자

- PKCS 인터페이스
- 작업 핸들러
- MQTT 에이전트

엔드 디바이스 SDK와 통합

1. 데이터 모델용 코드 생성기의 지침에 따라 하위 수준 C 코드를 생성합니다.
2. 엔드 디바이스 SDK 통합의 지침에 따라 다음을 수행합니다.

   a. 빌드 환경 설정

   Amazon Linux 2023/x86_64에서 개발 호스트로 코드를 빌드합니다. 필요한 빌드 종속성을 설치합니다.

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

   b. 하드웨어 콜백 함수 개발

   하드웨어 콜백 함수를 구현하기 전에 API의 작동 방식을 이해합니다. 이 예제에서는 On/Off 클러스터 및 OnOff 속성을 사용하여 디바이스 함수를 제어합니다. API 세부 정보는 섹션을 참조하세요하위 수준 C-Function APIs.

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

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

c. 엔드포인트 및 후크 하드웨어 콜백 함수 설정

함수를 구현한 후 엔드포인트를 생성하고 콜백을 등록합니다. 다음 작업을 완료합니다.

i. 디바이스 에이전트 생성

ii. 지원하려는 각 클러스터 구조체의 콜백 함수 포인트 채우기

iii. 엔드포인트 설정 및 지원되는 클러스터 등록

```
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);
}
```

예: 허브 제어 구축

허브 제어는 Hub SDK 패키지의 일부로 제공됩니다. 허브 제어 하위 패키지에는 레이블이 지정되어 `iot-managed-integrations-hub-control-x.x.x` 있으며 수정되지 않은 디바이스 SDK와 다른 라이브러리가 포함되어 있습니다.

1. 코드 생성 파일을 `example` 폴더로 이동합니다.

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

2. 허브 제어를 빌드하려면 다음 명령을 실행합니다.

```
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. `HubOnboarding` 및 `MQTTProxy` 구성 요소가 실행 중인 상태에서 허브의 `MQTTProxy` 구성 요소를 사용하여 예제를 실행합니다.

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

데이터 모델은 섹션을 참조관리형 통합 데이터 모델하세요. 의 5단계End Device SDK 시작하기에 따라 엔드포인트를 설정하고 최종 사용자와 iot-managed-integrations 통신을 관리합니다.

지원되는 예제

다음 예제가 빌드 및 테스트되었습니다.

- iotmi_device_dm_air_purifier_demo
- iotmi_device_basic_diagnostics

- iotmi_device_dm_camera_demo

지원하는 플랫폼

다음 표에는 허브 제어에 지원되는 플랫폼이 나와 있습니다.

아키텍처	운영 체제	GCC 버전	Binutils 버전
X86_64	Linux	10.5.0	2.37
aarch64	Linux	10.5.0	2.37

CloudWatch Logs 활성화

Hub SDK는 포괄적인 로깅 기능을 제공합니다. 기본적으로 Hub SDK는 로컬 파일 시스템에 로그를 기록합니다. 그러나 클라우드 API를 활용하여 CloudWatch Logs로 로그 스트리밍을 구성할 수 있습니다. 이 로그 스트리밍은 다음을 제공합니다.

- 디바이스 성능 모니터링: 사전 예방적 디바이스 관리를 위해 세부 런타임 로그를 캡처합니다. 디바이스 플릿에서 고급 로그 분석 및 모니터링 활성화
- 문제 해결: 신속한 진단 분석을 위해 세분화된 로그 항목을 생성합니다. 심층 조사를 위해 시스템 및 애플리케이션 수준 이벤트를 기록합니다.
- 유연한 중앙 집중식 로깅: 직접 디바이스 액세스가 없는 원격 로그 관리. 검색 가능한 단일 리포지토리에서 여러 디바이스의 로그를 집계합니다.

사전 조건

- 관리형 디바이스를 클라우드에 온보딩합니다. 세부 정보는 허브 온보딩 설정 섹션을 참조하세요.
- Hub 에이전트 시작 및 성공적인 초기화를 확인합니다. 세부 정보는 관리형 통합 Hub SDK 설치 및 검증 섹션을 참조하세요.

> **Note**
>
> 로깅 구성을 생성하려면 PutRuntimeLogConfiguration API에서 세부 정보를 참조하세요.

> **Warning**
>
> 로그를 활성화하면 계층형 할당량 측정에 포함됩니다. 로그 수준을 높이면 메시지 볼륨이 증가하고 추가 비용이 발생합니다.

Setup Hub SDK 로그 구성

API를 호출하여 런타임 로그 구성을 설정하여 허브 SDK 로그 설정을 구성합니다.

Example샘플 API 요청

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

RuntimeLogConfigurations 속성

다음 속성은 선택 사항이며 `RuntimeLogConfigurations` API에서 구성할 수 있습니다.

LogLevel

런타임 추적의 최소 심각도 수준을 설정합니다. 값: `DEBUG, ERROR, INFO, WARN`

기본값: `WARN` (릴리스된 빌드)

LogFlushLevel

로컬 스토리지로 즉시 데이터를 플러싱하기 위한 심각도 수준을 결정합니다. 값: `DEBUG, ERROR, INFO, WARN`

기본값: `DISABLED`

LocalStoreLocation

런타임 트레이스의 스토리지 위치를 지정합니다. 기본값: `/var/log/awsiotmi`

- 활성 로그: `/var/log/awsiotmi/ManagedIntegrationsDeviceSdkHub.log`
- 교체된 로그: `/var/log/awsiotmi/ManagedIntegrationsDeviceSdkHub.N.log` (N은 교체 순서를 나타냄)

LocalStoreFileRotationMaxBytes

현재 파일이 지정된 크기를 초과하면 파일 교체를 트리거합니다.

> **Important**
>
> 최적의 효율성을 위해 파일 크기를 125KB 미만으로 유지합니다. 125KB를 초과하는 값은 자동으로 제한됩니다.

LocalStoreFileRotationMaxFiles,

로그 데몬에서 허용되는 최대 교체 파일 수를 설정합니다.

UploadLog

클라우드로의 런타임 추적 전송을 제어합니다. 로그는 `/aws/iotmanagedintegration` CloudWatch Logs 그룹에 저장됩니다.

기본값: `false`.

UploadPeriodMinutes

런타임 추적 업로드 빈도를 정의합니다. 기본값: `5`

DeleteLocalStoreAfterUpload

업로드 후 파일 삭제를 제어합니다. 기본값: `true`

> **Note**
>
> false로 설정하면 업로드된 파일의 이름이 다음과 같이 변경됩니다. `/var/log/awsiotmi/ManagedIntegrationsDeviceSdkHub.uploaded.{uploaded_timestamp}`

예제 로그 파일

아래 CloudWatch Logs 파일의 예를 참조하세요.

지원되는 Zigbee 및 Z-Wave 디바이스 유형

이 페이지에는 관리형 통합으로 테스트되고 지원되는 허브 연결 디바이스 유형이 나열되어 있습니다. 관리형 통합은 이러한 디바이스에 사용자 안내 설정(UGS) 대해 단순 설정(SS) 및를 모두 지원합니다.

이 표에는 지원되는 Zigbee 디바이스가 나열되어 있습니다.

Zigbee 디바이스 유형	지원되는 기능
스마트 전구 / 밝기 조절 가능 조명 / RGB 조명	OnOff, LevelControl, ColorControl
스마트 플러그	OnOff
스마트 스위치	OnOff
LED 스트립	OnOff, LevelControl, ColorControl
워터 밸브	OnOff
Radiator 밸브	온도 조절기, OnOff, 타이머
온도 조절기	온도 조절기, FanControl, OnOff, 타이머
차고문 열림 버튼	WindowCovering, OnOff, LevelControl
연기 경보	BooleanState, OnOff, TemperatureMeasurement, Timer, SmokeCOAlarm
모션 센서	BooleanState
점유/인간 존재 센서	BooleanState, OccupancySensing
도어 및 윈도우 센서	BooleanState
누수 센서	BooleanState
진동 센서	BooleanState
온도 및 습도 센서	TemperatureMeasurement, RelativeHumidityMeasurement

이 표에는 지원되는 Z-Wave 디바이스가 나열되어 있습니다.

Z-Wave 디바이스 유형	지원되는 기능
스마트 전구/조명등	OnOff, LevelControl

Z-Wave 디바이스 유형	지원되는 기능
스마트 플러그	OnOff
차고문 컨트롤러	OnOff, LevelControl
에너지 측정기	ElectricalEnergyMeasurement, ElectricalPowerMeasurement
배터리	LevelControl
사이렌	LevelControl
모션 센서	BooleanState
도어 및 윈도우 센서	BooleanState
누수 센서	BooleanState
온도 센서	TemperatureMeasurement
CO 센서	SmokeCOAlarm
연기 센서	SmokeCOAlarm

Raspberry Pi에서 관리형 통합 실행

> **Note**
>
> Raspberry Pi에서 AWS IoT Hub SDK를 구현하는 것은 학습 및 테스트 목적으로만 사용되는 데모 프로젝트이며 프로덕션 환경에서는 사용할 수 없습니다. 이 데모에서는 쉽게 개발할 수 있도록 다음 구성을 설정합니다.
> AWS 자격 증명 스토리지: 데모용으로만 자격 증명과 인증서는 더 쉬운 테스트 및 개발을 위해 액세스 가능한 위치에 저장됩니다. 프로덕션 환경은 AWS Secrets Manager또는 Systems Manager Parameter Store와 같은 보안 스토리지 솔루션을 사용해야 합니다. 저장 데이터 암호화를 구현하고 AWS IoT 보안 지침을 따라야 합니다.
> 컨테이너 권한: 데모는 승격된 권한으로 실행되어 호스트 리소스에 대한 무제한 액세스를 허용하고 개발 워크플로를 간소화합니다. 프로덕션 환경에서 컨테이너는 최소한의 필수 권한으로 작동해야 합니다.

네트워크 브리지 구성: 데모는 더 쉬운 디버깅 및 모니터링을 위해 내부 네트워크 트래픽을 노출하는 네트워크 브리지 구성을 사용합니다. 프로덕션 환경에서는 적절한 네트워크 격리 및 세분화를 구현하여 내부 네트워크 트래픽에 대한 무단 액세스를 방지합니다.
USB 디바이스 권한: 개발 주변 장치 및 테스트 디바이스를 쉽게 연결할 수 있도록 무제한 USB 디바이스 액세스가 활성화됩니다. 프로덕션의 경우 엄격한 USB 디바이스 제어 및 검증을 구현하여 디바이스 스푸핑 공격을 방지합니다.
이러한 구성은 간단한 테스트를 가능하게 하며 프로덕션 환경에서 사용해서는 안 됩니다. 프로덕션에 배포할 때 호스트 시스템 손상 및 자격 증명에 대한 무단 액세스를 방지하려면 보안 모범 사례를 따르십시오.

사전 조건으로 Raspberry Pi를 설정하기 전에 Sonoff Zigbee USB 동글을 설정해야 합니다.

Sonoff Zigbee USB 동글에 대한 플래시 펌웨어

사전 조건

- Sonoff Zigbee USB 동글
- Windows: CP210x 범용 Windows 드라이버 설치

펌웨어 플래시

1. Zigbee Dongle 펌웨어 빌드 7.4.1.0을 다운로드합니다.
2. Silabs 펌웨어 플래셔를 엽니다.
3. Sonoff Zigbee USB 동글을 컴퓨터에 연결합니다.
4. 스크롤하여 ZBDongle-E를 찾습니다.
5. 연결을 선택합니다.
6. 디바이스가 연결될 때까지 기다립니다.
7. 펌웨어 변경을 선택합니다.
8. 자체 펌웨어 업로드를 선택합니다.
9. Zigbee Dongle Firmware Build 7.4.1.0 다운로드 위치를 찾아 선택합니다.
10. 설치를 클릭합니다.
11. 펌웨어가 설치될 때까지 기다립니다.

12. 설치가 완료되면 계속을 선택합니다.

이제 동글을 사용할 준비가 되었습니다.

아래 나열된 옵션 중에서 선택하여 Raspberry Pi에서 관리형 통합 Hub SDK를 실행합니다. 두 접근 방식에 대한 설정 및 검증 단계는 아래에 나열되어 있습니다.

주제

- Raspberry Pi의 관리형 통합 Hub SDK 이미지
- Raspberry Pi의 관리형 통합 Hub SDK Docker 컨테이너
- 관리형 통합 데모 애플리케이션

Raspberry Pi의 관리형 통합 Hub SDK 이미지

> **Note**
>
> Raspberry Pi에서 AWS IoT Hub SDK의이 구현은 학습 및 테스트 목적으로만 사용되는 데모 프로젝트이며 프로덕션 환경에서는 사용할 수 없습니다. 이 데모에서는 쉽게 개발할 수 있도록 다음 구성을 설정합니다.
> AWS 자격 증명 스토리지: 데모용으로만 자격 증명과 인증서는 더 쉬운 테스트 및 개발을 위해 액세스 가능한 위치에 저장됩니다. 프로덕션 환경은 AWS Secrets Manager또는 Systems Manager Parameter Store와 같은 보안 스토리지 솔루션을 사용해야 합니다. 유휴 시 암호화를 구현하고 AWS IoT 보안 지침을 따라야 합니다.
> 컨테이너 권한: 데모는 승격된 권한으로 실행되어 호스트 리소스에 대한 무제한 액세스를 허용하고 개발 워크플로를 간소화합니다. 프로덕션 환경에서 컨테이너는 최소한의 필수 권한으로 작동해야 합니다.
> 네트워크 브리지 구성: 데모는 디버깅 및 모니터링을 더 쉽게 하기 위해 내부 네트워크 트래픽을 노출하는 네트워크 브리지 구성을 사용합니다. 프로덕션 환경에서는 적절한 네트워크 격리 및 세분화를 구현하여 내부 네트워크 트래픽에 대한 무단 액세스를 방지합니다.
> USB 디바이스 권한: 개발 주변 장치 및 테스트 디바이스를 쉽게 연결할 수 있도록 무제한 USB 디바이스 액세스가 활성화됩니다. 프로덕션의 경우 엄격한 USB 디바이스 제어 및 검증을 구현하여 디바이스 스푸핑 공격을 방지합니다.

이러한 구성은 간단한 테스트를 가능하게 하며 프로덕션 환경에서 사용해서는 안 됩니다. 프로덕션에 배포할 때 호스트 시스템 손상 및 자격 증명에 대한 무단 액세스를 방지하려면 보안 모범 사례를 따르십시오.

사전 조건

Raspberry Pi 이미지를 배포하기 전에 다음 요구 사항을 완료합니다.

- Raspberry Pi 이미저를 다운로드하여 설치합니다.
- SD 카드를 받습니다.
- 2.4Ghz 64비트 쿼드 코어 CPU(8GB RAM)를 사용하여 Raspberry Pi 5를 설정합니다.
- Sonoff Zigbee USB 동글을 연결합니다.
- Sonoff Zigbee USB 동글에 대한 플래시 펌웨어.
- Silicon Labs SLUSB001A 동글을 연결합니다.
- AWS 계정에 가입합니다.
- AWS CLI 관리형 통합 AWS CLI 명령 참조에서의 최신 버전을 설치합니다.

새 SD 카드에서 Raspberry Pi 이미지 플래시

다음 단계를 사용하여 관리형 통합 이미지를 SD 카드에 플래시합니다.

1. 관리형 통합 Raspberry Pi Hub SDK 이미지를 다운로드합니다.
2. 데스크톱에서 Raspberry Pi Imager를 시작합니다.
3. 컴퓨터의 내장 SD 카드 리더 또는 외부 USB 카드 리더에 SD 카드를 삽입합니다.
4. 디바이스 선택 → Raspberry Pi 5를 선택합니다.
5. OS 선택 → 사용자 지정 사용 → IotMI-HubSDK-RPi-Image-v1.0.0.img.gz 파일 찾기 → 열기를 선택합니다.
6. 스토리지 선택 → SD 카드 리더 선택을 선택합니다.
7. 구성이 다음 화면과 일치하는지 확인합니다.

8. 다음을 클릭합니다.
9. OS 사용자 지정 설정을 구성합니다.
   - 호스트 이름: raspberrypi를 선택합니다.

- 사용자 이름 및 암호:
  - 사용자 이름 및 암호 설정 활성화:
  - 사용자 이름:에를 입력합니다hub123456.
  - 암호:에를 입력합니다sh123456.
- 무선 LAN:
  - 무선 LAN 구성을 활성화합니다.
  - 라우터의 SSID와 암호를 입력합니다.

    예제 설정:

    - SSID:iotmi-tplink
    - 암호:******** (최소 8자)
  - 국가:를 로 설정합니다US.
- 로캘 설정:
  - 시간대를 로 설정합니다America/Los Angeles.
  - 키보드 레이아웃을 로 설정합니다US.
- SSH:
  - 서비스 탭을 선택합니다.
  - SSH 활성화를 선택합니다.
  - 암호 인증 사용을 선택합니다.

10.OS 사용자 지정 및 데이터 삭제를 위한 모든 팝업을 확인합니다.

11.쓰기 프로세스가 완료될 때까지 기다립니다.

12.다음 화면을 사용하여 성공적으로 완료되었는지 확인합니다.

13.계속을 클릭합니다.

14.SD 카드를 제거하고 Raspberry Pi에 삽입합니다.

Raspberry Pi에서 Hub SDK 실행

구성된 Raspberry Pi에서 Hub SDK 서비스를 시작합니다.

1. 준비된 SD 카드를 Raspberry Pi 5 디바이스에 삽입합니다.
2. Sonoff Zigbee USB 동글과 Silicon Labs SLUSB001A 동글을 Raspberry Pi에 연결합니다.

3. Raspberry Pi의 전원을 켭니다.
4. Raspberry Pi와 컴퓨터(SSH)가 동일한 네트워크에 있는지 확인합니다.
5. 이미지 배포 중에 설정한 자격 증명을 사용하여 Raspberry Pi로 SSH합니다.

```
ssh username@hostname
```

6. 허브 SDK 디렉터리로 이동합니다.

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

7. Hub 온보딩 설정을 완료하여 인증 및 구성 자료를 추가합니다.

> **Note**
>
> 이 단계를 수행하려면 YUL 또는 DUB 리전에 있어야 합니다.

8. Hub SDK를 실행합니다.

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

시스템에는 성공적인 Hub SDK 시작에 대한 다음 응답이 표시됩니다.

```
-------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----
```

다음 단계

Hub SDK를 성공적으로 시작한 후에서 디바이스 온보딩 및 관리를 진행합니다디바이스 온보딩 및 운영을 위한 사용자 안내 설정.

Raspberry Pi의 관리형 통합 Hub SDK Docker 컨테이너

> **Note**
>
> Raspberry Pi에서 AWS IoT Hub SDK를 구현하는 것은 학습 및 테스트 목적으로만 사용되는 데모 프로젝트이며 프로덕션 환경에서는 사용할 수 없습니다. 이 데모에서는 쉽게 개발할 수 있도록 다음 구성을 설정합니다.
> AWS 자격 증명 스토리지: 데모용으로만 자격 증명과 인증서는 더 쉬운 테스트 및 개발을 위해 액세스 가능한 위치에 저장됩니다. 프로덕션 환경은 AWS Secrets Manager또는 Systems Manager Parameter Store와 같은 보안 스토리지 솔루션을 사용해야 합니다. 저장 데이터 암호화를 구현하고 AWS IoT 보안 지침을 따라야 합니다.

컨테이너 권한: 데모는 승격된 권한으로 실행되어 호스트 리소스에 대한 무제한 액세스를 허용하고 개발 워크플로를 간소화합니다. 프로덕션 환경에서 컨테이너는 최소한의 필수 권한으로 작동해야 합니다.
네트워크 브리지 구성: 데모는 더 쉬운 디버깅 및 모니터링을 위해 내부 네트워크 트래픽을 노출하는 네트워크 브리지 구성을 사용합니다. 프로덕션 환경에서는 적절한 네트워크 격리 및 세분화를 구현하여 내부 네트워크 트래픽에 대한 무단 액세스를 방지합니다.
USB 디바이스 권한: 개발 주변 장치 및 테스트 디바이스를 쉽게 연결할 수 있도록 무제한 USB 디바이스 액세스가 활성화됩니다. 프로덕션의 경우 엄격한 USB 디바이스 제어 및 검증을 구현하여 디바이스 스푸핑 공격을 방지합니다.
이러한 구성은 간단한 테스트를 가능하게 하며 프로덕션 환경에서 사용해서는 안 됩니다. 프로덕션에 배포할 때 호스트 시스템 손상 및 자격 증명에 대한 무단 액세스를 방지하려면 보안 모범 사례를 따르십시오.

사전 조건

Docker 컨테이너의 경우 다음 사전 조건이 필요합니다.

- Raspberry Pi 이미저를 다운로드하여 설치합니다.
- SD 카드를 받습니다.
- 2.4Ghz 64비트 쿼드 코어 CPU(8GB RAM)로 Raspberry Pi 5를 설정합니다.
- Sonoff Zigbee USB 동글을 연결합니다.
- Sonoff Zigbee USB 동글에 대한 플래시 펌웨어.
- Silicon Labs SLUSB001A 동글을 연결합니다.
- AWS 계정에 가입합니다.
- AWS CLI 관리형 통합 AWS CLI 명령 참조에서 최신 버전의를 설치합니다.
- IP 주소 또는 호스트 이름을 사용하여 Raspberry Pi에 대한 SSH 액세스.

Raspberry Pi에서 Managed integrations Hub SDK Docker 컨테이너 사용

1. 관리형 통합 Raspberry Pi Hub SDK Docker를 다운로드합니다.
2. SCP를 사용하여 파일을 Raspberry Pi에 복사합니다.

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

3. SSH를 통해 Raspberry Pi에 연결합니다.

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

4. 없는 경우 Docker를 설치합니다.

```
# 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. 없는 경우 Docker Compose를 설치합니다.

```
# Install Docker Compose
sudo apt-get update
sudo apt-get install -y docker-compose-plugin
```

6. Hub SDK 파일을 추출합니다.

```
# Navigate to the home directory
cd

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

7. hub-docker 디렉터리로 이동합니다.

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

8. Hub 온보딩 설정을 완료하여 인증 및 설정을 구성합니다.

> **Note**
>
> 이 단계를 수행하려면 YUL 또는 DUB 리전에 있어야 합니다.

9. Docker 컨테이너를 시작합니다.

```
# The first time it's called, it will build the container
```

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

예상 출력:

```
[+] 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-\-\-\-
```

Hub SDK를 성공적으로 시작한 후에서 디바이스 온보딩 및 관리를 진행합니다디바이스 온보딩 및 운영을 위한 사용자 안내 설정.

Note

- Docker 컨테이너 bash 쉘에 액세스하려면 다음 명령을 실행합니다.

  ```
  docker compose exec hubsdk bash
  ```

- 재부팅 후 컨테이너를 다시 시작하려면 다음 명령을 실행합니다.

  ```
  docker compose up -d
  ```

- Hub SDK를 업데이트하려면 다음 폴더의 바이너리를 바꿉니다.

  ```
  hub-docker/iotmi
  ```

- 데이터를 보존하면서 컨테이너를 안전하게 다시 시작하려면 다음을 수행합니다.

  ```
  docker compose down
  docker compose up -d
  ```

```
docker compose logs -f
```

관리형 통합 데모 애플리케이션

> **Note**
>
> Raspberry Pi에서 AWS IoT Hub SDK를 구현하는 것은 학습 및 테스트 목적으로만 사용되는 데모 프로젝트이며 프로덕션 환경에서는 사용할 수 없습니다. 이 데모에서는 쉽게 개발할 수 있도록 다음 구성을 설정합니다.
> AWS 자격 증명 스토리지: 데모용으로만 자격 증명과 인증서는 더 쉬운 테스트 및 개발을 위해 액세스 가능한 위치에 저장됩니다. 프로덕션 환경은 AWS Secrets Manager또는 Systems Manager Parameter Store와 같은 보안 스토리지 솔루션을 사용해야 합니다. 저장 데이터 암호화를 구현하고 AWS IoT 보안 지침을 따라야 합니다.
> 컨테이너 권한: 데모는 승격된 권한으로 실행되어 호스트 리소스에 대한 무제한 액세스를 허용하고 개발 워크플로를 간소화합니다. 프로덕션 환경에서 컨테이너는 최소한의 필수 권한으로 작동해야 합니다.
> 네트워크 브리지 구성: 데모는 더 쉬운 디버깅 및 모니터링을 위해 내부 네트워크 트래픽을 노출하는 네트워크 브리지 구성을 사용합니다. 프로덕션 환경에서는 적절한 네트워크 격리 및 세분화를 구현하여 내부 네트워크 트래픽에 대한 무단 액세스를 방지합니다.
> USB 디바이스 권한: 개발 주변 장치 및 테스트 디바이스를 쉽게 연결할 수 있도록 무제한 USB 디바이스 액세스가 활성화됩니다. 프로덕션의 경우 엄격한 USB 디바이스 제어 및 검증을 구현하여 디바이스 스푸핑 공격을 방지합니다.
> 이러한 구성은 간단한 테스트를 가능하게 하며 프로덕션 환경에서 사용해서는 안 됩니다. 프로덕션에 배포할 때 호스트 시스템 손상 및 자격 증명에 대한 무단 액세스를 방지하려면 보안 모범 사례를 따르십시오.

데모 애플리케이션은 스마트 홈 디바이스 관리를 위한 관리형 통합 기능을 보여주는 React 기반 데모 애플리케이션입니다. 이 애플리케이션은 최신 웹 인터페이스를 통해 Z-Wave 및 Zigbee 디바이스에 대한 디바이스 온보딩, 제어 및 모니터링을 보여줍니다.

사전 조건

- AWS 계정에 가입합니다.
- 자격 증명 로커를 생성하고 자격 증명 로커를 허브에 추가합니다.
- Hub 온보딩 설정을 완료합니다.

- Node.js 18+ 및 npm.
- AWS CLI 관리형 통합 AWS CLI 명령 참조에서의 최신 버전을 설치합니다.
- 최신 웹 브라우저(Chrome, Firefox, Safari, Edge)

애플리케이션 설치 및 구성

1. 관리형 통합 데모 애플리케이션을 다운로드합니다.
2. 패키지 추출:

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

3. 종속 항목을 설치합니다:

```
npm install
```

4. 루트 디렉터리에 `.env` 파일을 생성합니다.

```
# 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. 애플리케이션을 빌드하고 시작합니다.

```
npm start
```

6. 다음에서 애플리케이션에 액세스합니다.

```
http://localhost:3000
```

요금 정보는 AWS IoT 디바이스 관리 요금 페이지의 관리형 통합 섹션을 참조하세요.

관리형 통합 허브 오프보드

Hub SDK 오프보드 프로세스 개요

허브 오프보딩 프로세스는 AWS 클라우드 관리 시스템에서 허브를 제거합니다. 클라우드가 DeleteManagedThing 요청을 보내면 프로세스는 두 가지 기본 목표를 달성합니다.

디바이스 측 작업:

- 허브의 내부 상태 재설정
- 로컬로 저장된 모든 데이터 삭제
- 향후 재온보딩 가능성을 위해 디바이스 준비

클라우드 측 작업:

- 허브와 연결된 모든 클라우드 리소스 제거
- 이전 계정과의 연결 해제 완료

고객은 일반적으로 다음과 같은 경우 허브 오프보딩을 시작합니다.

- 허브의 연결된 계정 변경
- 기존 허브를 새 디바이스로 교체

이 프로세스는 허브 구성 간에 깔끔하고 안전한 전환을 보장하여 원활한 디바이스 관리와 계정 유연성을 제공합니다.

사전 조건

- 온보딩된 허브가 있어야 합니다. 지침은 허브 온보딩 설정을 참조하세요.
- /data/aws/iotmi/config/에 있는 `iotmi_config.json` 파일에서가 `iot_provisioning_state`를 표시하는지 확인합니다`PROVISIONED`.
- 에서 참조되는 영구 인증서와 키가 지정된 경로에 `iotmi_config.json` 있는지 확인합니다.

- HubOnboarding, Agent, Provisioner 및 MQTT 프록시가 올바르게 구성되고 실행 중인지 확인합니다.
- 허브에 하위 디바이스가 없는지 확인합니다. 계속하기 전에 DeleteManagedThing API를 사용하여 모든 하위 디바이스를 제거합니다.

Hub SDK 오프보드 프로세스

다음 단계에 따라 허브를 오프보드합니다.

hub_managed_thing ID 검색

`iotmi_config.json` 파일은 관리형 통합 허브의 관리형 사물 ID를 저장하는 데 사용됩니다. 이 식별자는 허브가 AWS IoT 관리형 통합 서비스와 통신할 수 있도록 하는 중요한 정보입니다. 관리형 사물 ID는 JSON 파일의 rw(읽기-쓰기) 섹션 내 `managed_thing_id` 필드 아래에 저장됩니다. 이는 다음 샘플 구성에서 볼 수 있습니다.

```
{
      "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
          }
      }
}
```

오프보드 허브로 명령 전송

계정 자격 증명을 사용하고 이전 섹션에서 `managed_thing_id` 검색된 로 명령을 실행합니다.

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

허브가 오프보딩되었는지 확인

계정 자격 증명을 사용하고 이전 섹션에서 managed_thing_id 검색된 로 명령을 실행합니다.

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

성공 및 실패 시나리오

성공 시나리오

허브 오프보드 명령이 성공하면 다음 샘플 응답이 예상됩니다.

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

또한 허브 오프보딩 명령이 성공하면 다음 샘플이 관찰iotmi_config.json됩니다. rw 섹션에 iot_provisioning_state 및 선택적으로 메타데이터만 포함되어 있는지 확인합니다. 메타데이터의 부재는 허용됩니다.는 NOT_PROVISIONED여야 iot_provisioning_state 합니다.

```
{
      "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
          }
      }
}
```

실패 시나리오

허브 오프보드 명령이 실패한 경우 다음 샘플 응답이 예상됩니다.

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

- ProvisioningStatus가 인 경우 Hub 복구의 지침을 DELETE_IN_PROGRESS따릅니다.
- ProvisioningStatus가가 아닌 경우 허브 오프보드 DELETE_IN_PROGRESS명령이 관리형 통합 클라우드에서 실패했거나 관리형 통합 클라우드에서 수신되지 않았습니다. Hub 복구의 지침을 따릅니다.
- 오프보딩에 실패한 경우 iotmi_config.json 파일은 아래 샘플 파일과 같습니다.

```
{
      "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
          }
```

```
        }
}
```

(선택 사항) 오프보딩 후 Hub SDK

> **Important**
>
> 다음 시나리오에서는 오프보딩 Hub SDK가 실패한 후 또는 오프보딩 후 허브를 다시 온보딩하려는 경우 수행할 선택적 작업을 나열합니다.

재온보딩

오프보딩에 성공한 경우 3단계: 관리형 사물 생성(플릿 프로비저닝) 및 나머지 온보딩 프로세스에 따라 Hub SDK를 온보딩합니다.

허브 복구

디바이스 허브 오프보딩 성공 및 클라우드 오프보딩 실패

GetManagedThing API 호출`iotmi_config.json`이 `Managed Thing resource not found` 메시지를 반환하지 않지만 파일이 오프보딩된 경우. 샘플 json 파일은 성공 시나리오를 참조하세요.

이 시나리오에서 복구하려면 강제 삭제를 참조하세요.

디바이스 허브 오프보딩 실패

이 시나리오는 파일이 올바르게 오프보드되지 않는 경우`iotmi_config.json`입니다. 샘플 json 파일의 실패 시나리오를 참조하세요.

이 시나리오에서 복구하려면 강제 삭제를 참조하세요. `iotmi_config.json`가 아직 오프보딩되지 않은 경우 허브를 공장 초기화해야 합니다.

디바이스 허브 오프보딩 및 클라우드 오프보딩 실패

이 시나리오에서 `iotmi_config.json`는 여전히 오프보딩되지 않으며 허브 상태는 `ACTIVATED`또는 입니다`DISCOVERED`.

이 시나리오에서 복구하려면 강제 삭제를 참조하세요. 강제 삭제`iotmi_config.json`에 실패하거나 오프보딩되지 않은 경우 허브를 초기 기본값으로 재설정해야 합니다.

허브가 오프라인 상태이고 허브 상태가 DELETE_IN_PROGRESS임

이 시나리오에서는 허브가 오프라인 상태이고 클라우드가 오프보딩 명령을 수신합니다.

이 시나리오에서 복구하려면 강제 삭제를 참조하세요.

강제 삭제

성공적인 디바이스 허브 오프보딩 없이 클라우드 리소스를 삭제하려면 다음 단계를 따르세요. 이 작업을 수행하면 클라우드와 디바이스 상태 간에 불일치가 발생하여 향후 운영에 문제가 발생할 수 있습니다.

허브의 `managed_thing_id` 및 force 파라미터를 사용하여 DeleteManagedThing API를 호출합니다.

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

그런 다음 GetManagedThing API를 호출하고를 반환하는지 확인합니다`Managed Thing resource not found`. 이렇게 하면 클라우드 리소스가 삭제됩니다.

> **Note**
>
> 클라우드 상태와 디바이스 상태 간에 불일치가 발생할 수 있으므로이 접근 방식은 권장되지 않습니다. 일반적으로 클라우드 리소스를 삭제하기 전에 성공적인 디바이스 허브 오프보딩을 보장하는 것이 좋습니다.

프로토콜별 미들웨어

> **Important**
>
> 여기에 제공된 설명서와 코드는 미들웨어의 참조 구현을 설명합니다. SDK의 일부로 제공되지 않습니다.

프로토콜별 미들웨어는 기본 프로토콜 스택과 상호 작용하는 중요한 역할을 합니다. 관리형 통합 Hub SDK의 디바이스 온보딩 및 디바이스 제어 구성 요소 모두 이를 사용하여 최종 디바이스와 상호 작용합니다.

미들웨어는 다음 함수를 수행합니다.

- 공통 APIs 세트를 제공하여 여러 공급업체의 디바이스 프로토콜 스택에서 APIs.

- 스레드 스케줄러, 이벤트 대기열 관리 및 데이터 캐시와 같은 소프트웨어 실행 관리를 제공합니다.

미들웨어 아키텍처

아래 블록 다이어그램은 Zigbee 미들웨어의 아키텍처를 나타냅니다. Z-Wave와 같은 다른 프로토콜의 미들웨어 아키텍처도 비슷합니다.

프로토콜별 미들웨어에는 세 가지 주요 구성 요소가 있습니다.

- ACS Zigbee DPK: Zigbee Device Porting Kit(DPK)는 기본 하드웨어 및 운영 체제에서 추상화를 제공하여 이식성을 가능하게 하는 데 사용됩니다. 기본적으로 이는 여러 공급업체의 Zigbee 라디오를 제어하고 통신하기 위한 공통 세트 APIs를 제공하는 하드웨어 추상화 계층(HAL)으로 간주될 수 있습니다. Zigbee 미들웨어에는 Silicon Labs Zigbee 애플리케이션 프레임워크를 위한 DPK API 구현이 포함되어 있습니다.
- ACS Zigbee 서비스: Zigbee 서비스는 전용 데몬으로 실행됩니다. 여기에는 IPC 채널을 통해 클라이언트 애플리케이션의 API 호출을 처리하는 API 핸들러가 포함됩니다. AIPC는 Zigbee 어댑터와 Zigbee 서비스 간의 IPC 채널로 사용됩니다. 비동기/동기 명령 처리, HAL의 이벤트 처리, 이벤트 등록/게시를 위한 ACS Event Manager 사용과 같은 다른 기능을 제공합니다.
- ACS Zigbee 어댑터: Zigbee 어댑터는 애플리케이션 프로세스 내에서 실행되는 라이브러리입니다(이 경우 애플리케이션은 CDMB 플러그인임). Zigbee 어댑터는 CDMB/프로비저너 프로토콜 플러그인과 같은 클라이언트 애플리케이션에서 엔드 디바이스를 제어하고 통신하기 위해 사용하는 APIs 세트를 제공합니다.

End-to-end 미들웨어 명령 흐름 예제

다음은 Zigbee 미들웨어를 통한 명령 흐름의 예입니다.

다음은 Z-Wave 미들웨어를 통한 명령 흐름의 예입니다.

프로토콜별 미들웨어 코드 조직

이 섹션에는 `IotManagedIntegrationsDeviceSDK-Middleware`리포지토리 내 각 구성 요소의 코드 위치에 대한 정보가 포함되어 있습니다. 다음은이 리포지토리의 폴더 구조의 예입니다.

```
./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
```

주제

- Zigbee 미들웨어 코드 구성
- Z-Wave 미들웨어 코드 구성

Zigbee 미들웨어 코드 구성

다음은 Zigbee 참조 미들웨어 코드 조직을 보여줍니다.

주제

- ACS Zigbee DPK
- Silicon Labs Zigbee SDK
- ACS Zigbee 서비스
- ACS Zigbee 어댑터

ACS Zigbee DPK

Zigbee DPK의 코드는 아래 예제에 나열된 디렉터리 내에 있습니다.

```
./IotManagedIntegrationsDeviceSDK-Middleware/example-iot-ace-dpk/example/dpk/ace_hal/
|– common
|–   |– fxnDbusClient
|–   |– include
|– kvs
|– log
|– wifi
|–   |– include
|–   |– src
|–   |– wifid
|–       |– fxnWifiClient
|–       |– include
```

```
|– zibgee
|–   |– include
|–   |– src
|–   |– zigbeed
|–       |– ember
|–       |– include
|– zwave
|–   |– include
|–   |– src
|–   |– zwaved
|–       |– fxnZwaveClient
|–       |– include
|–       |– zware
```

Silicon Labs Zigbee SDK

Silicon Labs SDK는 `IotManagedIntegrationsDeviceSDK-Middleware/example-iot-ace-z3-gateway` 폴더 내에 표시됩니다. 이 ACS Zigbee DPK 계층은이 Silicon Labs SDK에 구현됩니다.

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

ACS Zigbee 서비스

Zigbee Service의 코드는 `IotManagedIntegrationsDeviceSDK-Middleware/example-iot-ace-general/middleware/zigbee/` 폴더 내에 있습니다. 이 위치의 `src` 및 `include` 하위 폴더에는 ACS Zigbee 서비스와 관련된 모든 파일이 포함됩니다.

```
IotManagedIntegrationsDeviceSDK-Middleware/example-iot-ace-general/middleware/zigbee/
src/
|– zb_alloc.c
|– zb_callbacks.c
|– zb_database.c
|– zb_discovery.c
|– zb_log.c
|– zb_main.c
|– zb_region_info.c
|– zb_server.c
```

```
|– zb_svc.c
|– zb_svc_pwr.c
|– zb_timer.c
|– zb_util.c
|– zb_zdo.c
|– zb_zts.c
IotManagedIntegrationsDeviceSDK-Middleware/example-iot-ace-general/middleware/zigbee/
include/
|– init.zigbeeservice.rc
|– zb_ace_log_uml.h
|– zb_alloc.h
|– zb_callbacks.h
|– zb_client_aipc.h
|– zb_client_event_handler.h
|– zb_database.h
|– zb_discovery.h
|– zb_log.h
|– zb_region_info.h
|– zb_server.h
|– zb_svc.h
|– zb_svc_pwr.h
|– zb_timer.h
|– zb_util.h
|– zb_zdo.h
|– zb_zts.h
```

ACS Zigbee 어댑터

ACS Zigbee 어댑터의 코드는 `IotManagedIntegrationsDeviceSDK-Middleware/example-iot-ace-general/middleware/zigbee/api` 폴더 내부에 있습니다. 이 위치의 `src` 및 `include` 하위 폴더에는 ACS Zigbee 어댑터 라이브러리와 관련된 모든 파일이 포함되어 있습니다.

```
IotManagedIntegrationsDeviceSDK-Middleware/example-iot-ace-general/middleware/zigbee/
api/src/
|– zb_client_aipc.c
|– zb_client_api.c
|– zb_client_event_handler.c
|– zb_client_zcl.c
IotManagedIntegrationsDeviceSDK-Middleware/example-iot-ace-general/middleware/zigbee/
api/include/
|– ace
|–   |– zb_adapter.h
|–   |– zb_command.h
```

```
|—   |— zb_network.h
|—   |— zb_types.h
|—   |— zb_zcl.h
|—   |— zb_zcl_cmd.h
|—   |— zb_zcl_color_control.h
|—   |— zb_zcl_hvac.h
|—   |— zb_zcl_id.h
|—   |— zb_zcl_identify.h
|—   |— zb_zcl_level.h
|—   |— zb_zcl_measure_and_sensing.h
|—   |— zb_zcl_onoff.h
|—   |— zb_zcl_power.h
```

Z-Wave 미들웨어 코드 구성

다음은 Z파 참조 미들웨어 코드 구성을 보여줍니다.

주제

- ACS Z파 DPK
- Silicon Labs ZWare 및 Zip Gateway
- ACS Z-Wave 서비스
- ACS Z-Wave 어댑터

ACS Z파 DPK

Z-Wave DPK의 코드는 `IotManagedIntegrationsDeviceSDK-Middleware/example-iot-ace-dpk/example/dpk/ace_hal/zwave` 폴더 내에 있습니다.

```
./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 및 Zip Gateway

Silicon labs ZWare Zip Gateway의 코드는 `IotManagedIntegrationsDeviceSDK-Middleware/example-iot-ace-z3-gateway` 폴더 내에 있습니다. 이 ACS Z-Wave DPK 계층은 Z-Wave C-APIs 및 Zip 게이트웨이에 구현됩니다.

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

ACS Z-Wave 서비스

Z-Wave Service의 코드는 폴더에 나열된 `IoTmanagedintegrationsMiddlewares/exampleiot-ace-zwave-mw/` 폴더 내에 있습니다. 이 위치의 `src` 및 `include` 폴더에는 ACS Z-Wave 서비스와 관련된 모든 파일이 포함되어 있습니다.

```
IotManagedIntegrationsDeviceSDK-Middleware/example-iot-ace-zwave-mw/src/
|— zwave_mgr.c
|— zwave_mgr_cc.c
|— zwave_mgr_ipc_aipc.c
|— zwave_svc.c
|— zwave_svc_dispatcher.c
|— zwave_svc_hsm.c
|— zwave_svc_ipc_aipc.c
|— zwave_svc_main.c
```

```
|– zwave_svc_publish.c
IotManagedIntegrationsDeviceSDK-Middleware/example-iot-ace-zwave-mw/include/
|– ace
|–   |– zwave_common_cc.h
|–   |– zwave_common_cc_battery.h
|–   |– zwave_common_cc_doorlock.h
|–   |– zwave_common_cc_firmware.h
|–   |– zwave_common_cc_meter.h
|–   |– zwave_common_cc_notification.h
|–   |– zwave_common_cc_sensor.h
|–   |– zwave_common_cc_switch.h
|–   |– zwave_common_cc_thermostat.h
|–   |– zwave_common_cc_version.h
|–   |– zwave_common_types.h
|–   |– zwave_mgr.h
|–   |– zwave_mgr_cc.h
|– zwave_log.h
|– zwave_mgr_internal.h
|– zwave_mgr_ipc.h
|– zwave_svc_hsm.h
|– zwave_svc_internal.h
|– zwave_utils.h
```

ACS Z-Wave 어댑터

ACS Zigbee 어댑터의 코드는 `IotManagedIntegrationsDeviceSDK-Middleware/example-iot-ace-zwave-mw/cli/` 폴더 내부에 있습니다. 이 위치의 `src` 및 `include` 폴더에는 ACS Z-Wave 어댑터 라이브러리와 관련된 모든 파일이 포함되어 있습니다.

```
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
```

미들웨어를 SDK와 통합

새 허브의 미들웨어 통합은 다음 섹션에서 설명합니다.

주제

디바이스 이식 키트(DPK) API 통합

칩셋 공급업체 SDK를 미들웨어와 통합하기 위해 중앙의 DPK(디바이스 이식 키트) 계층에서 표준 API 인터페이스를 제공합니다. 관리형 통합 서비스 공급자 또는 ODMs은 IoT Hub에서 사용되는 Zigbee/Z-wave/Wi-Fi 칩셋에서 지원하는 공급업체 SDK를 기반으로 이러한 APIs를 구현해야 합니다.

참조 구현 및 코드 구성

미들웨어를 제외하고 관리형 통합 Device Agent 및 CDMB(Common Data Model Bridge)와 같은 다른 모든 Device SDK 구성 요소는 수정 없이 사용할 수 있으며 교차 컴파일만 하면 됩니다.

미들웨어 구현은 Silicon Labs SDK for Zigbee 및 Z-Wave를 기반으로 합니다. 새 허브에 사용되는 Z-Wave 및 Zigbee 칩셋이 미들웨어에 있는 Silicon Labs SDK에서 지원되는 경우 수정 없이 참조 미들웨어를 사용할 수 있습니다. 미들웨어를 교차 컴파일하기만 하면 새 허브에서 실행할 수 있습니다.

Zigbee용 DPK(디바이스 이식 키트) APIs는에서 찾을 수 있으며 `acehal_zigbee.c`DPK APIs의 참조 구현은 `zigbee` 폴더 내에 있습니다.

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

```
|-   |-   |- zbd_common.h
|-   |-   |- zbd_network_creator.h
|-   |-   |- zbd_power_settings.h
|-   |-   |- zbd_zts.h
```

Z-Wave용 DPK APIs는에서 찾을 수 `acehal_zwave.c` 있으며 DPK APIs의 참조 구현은 `zwaved` 폴더 내에 있습니다.

```
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
```

다른 공급업체 SDK에 대한 DPK 계층을 구현하기 위한 출발점으로 참조 구현을 사용하고 수정할 수 있습니다. 다른 공급업체 SDK를 지원하려면 다음 두 가지 수정이 필요합니다.

1. 현재 공급업체 SDK를 리포지토리의 새 공급업체 SDK로 바꿉니다.
2. 새 공급업체 SDK에 따라 미들웨어 DPK(디바이스 이식 키트) APIs를 구현합니다.

관리형 통합 엔드 디바이스 SDK

스마트 디바이스를 관리형 통합에 연결하고 통합 제어 인터페이스를 통해 명령을 처리하는 IoT 플랫폼을 구축합니다. End 디바이스 SDK는 디바이스 펌웨어와 통합되며 SDK 엣지 구성 요소와 함께 간소화된 설정과 AWS IoT Core 및 AWS IoT 디바이스 관리에 대한 보안 연결을 제공합니다. 에서 최신 버전의 End Device SDK 다운로드 AWS Management Console

이 가이드에서는 펌웨어에서 End Device SDK를 구현하는 방법을 설명합니다. 아키텍처, 구성 요소 및 통합 단계를 검토하여 구현 구축을 시작합니다.

주제

End 디바이스 SDK란 무엇입니까?

End 디바이스 SDK란 무엇입니까?

End 디바이스 SDK는에서 제공하는 소스 코드, 라이브러리 및 도구의 모음입니다 AWS IoT. 리소스가 제한된 환경을 위해 구축된 SDK는 임베디드 Linux 및 실시간 운영 체제(RTOS)에서 실행되는 카메라 및 공기청정기와 같이 최소 512KB RAM 및 4MB 플래시 메모리가 있는 디바이스를 지원합니다. AWS IoT 관리 콘솔에서 End device SDK의 최신 버전을 다운로드합니다.

핵심 구성 요소

SDK는 클라우드 통신을 위한 MQTT 에이전트, 작업 관리를 위한 작업 핸들러, 관리형 통합인 데이터 모델 핸들러를 결합합니다. 이러한 구성 요소는 함께 작동하여 디바이스와 관리형 통합 간의 안전한 연결과 자동화된 데이터 변환을 제공합니다.

자세한 기술 요구 사항은 섹션을 참조하세요기술 참조.

엔드 디바이스 SDK 아키텍처 및 구성 요소

이 섹션에서는 End device SDK 아키텍처와 해당 구성 요소가 하위 수준 C-Functions와 상호 작용하는 방법을 설명합니다. 다음 다이어그램은 SDK 프레임워크의 핵심 구성 요소와 그 관계를 보여줍니다.

엔드 디바이스 SDK 구성 요소

엔드 디바이스 SDK 아키텍처에는 관리형 통합 기능 통합을 위한 다음 구성 요소가 포함되어 있습니다.

프로비저닝 담당자

보안 MQTT 통신을 위한 디바이스 인증서 및 프라이빗 키를 포함하여 관리형 통합 클라우드에서 디바이스 리소스를 생성합니다. 이러한 자격 증명은 디바이스와 관리형 통합 간에 신뢰할 수 있는 연결을 설정합니다.

MQTT 에이전트

스레드 세이프 C 클라이언트 라이브러리를 통해 MQTT 연결을 관리합니다. 이 백그라운드 프로세스는 메모리가 제한된 디바이스에 대해 구성 가능한 대기열 크기로 다중 스레드 환경에서 명령 대기열을 처리합니다. 메시지는 처리를 위해 관리형 통합을 통해 라우팅됩니다.

작업 핸들러

디바이스 펌웨어, 보안 패치 및 파일 전송over-the-air(OTA) 업데이트를 처리합니다. 이 기본 제공 서비스는 등록된 모든 디바이스의 소프트웨어 업데이트를 관리합니다.

데이터 모델 핸들러

Matter Data Model의 AWS구현을 사용하여 관리형 통합과 하위 수준 C-함수 간의 작업을 변환합니다. 자세한 내용은 GitHub의 Matter 설명서를 참조하세요.

키 및 인증서

PKCS #11 API를 통해 암호화 작업을 관리하여 하드웨어 보안 모듈과 corePKCS11과 같은 소프트웨어 구현을 모두 지원합니다. 이 API는 TLS 연결 중에 Provisionee 및 MQTT Agent와 같은 구성 요소에 대한 인증서 작업을 처리합니다.

프로비저닝 담당자

프로비저닝 담당자는 클레임을 통한 플릿 프로비저닝을 지원하는 관리형 통합의 구성 요소입니다. 프로비저닝 담당자를 사용하면 디바이스를 안전하게 프로비저닝할 수 있습니다. SDK는 디바이스 프로

비저닝에 필요한 리소스를 생성합니다. 여기에는 관리형 통합 클라우드에서 가져온 디바이스 인증서와 프라이빗 키가 포함됩니다. 디바이스를 프로비저닝하려는 경우 또는 디바이스를 다시 프로비저닝해야 할 수 있는 변경 사항이 있는 경우 프로비저닝 담당자를 사용할 수 있습니다.

주제

프로비저닝 담당자 워크플로

이 프로세스를 수행하려면 클라우드 측과 디바이스 측 모두에서 설정해야 합니다. 고객은 사용자 지정 엔드포인트, 프로비저닝 프로필 및 관리형 사물과 같은 클라우드 요구 사항을 구성합니다. 첫 번째 디바이스 전원을 켤 때 권한 부여자는 다음을 수행합니다.

1. 클레임 인증서를 사용하여 관리형 통합 엔드포인트에 연결
2. 플릿 프로비저닝 후크를 통해 디바이스 파라미터 검증
3. 디바이스에 영구 인증서와 프라이빗 키를 가져와 저장합니다.
4. 디바이스는 영구 인증서를 사용하여 다시 연결합니다.
5. 디바이스 기능을 검색하여 관리형 통합에 업로드합니다.

프로비저닝에 성공하면 디바이스가 관리형 통합과 직접 통신합니다. 프로비저닝 담당자는 재프로비저닝 작업에 대해서만를 활성화합니다.

환경 변수 설정

클라우드 환경에서 다음 AWS 자격 증명을 설정합니다.

```
$ 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
```

사용자 지정 엔드포인트 등록

클라우드 환경에서 RegisterCustomEndpoint API 명령을 사용하여 device-to-cloud 통신을 위한 사용자 지정 엔드포인트를 생성합니다.

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

응답의 예

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

> **Note**
>
> 프로비저닝 파라미터를 구성하기 위한 엔드포인트 주소를 저장합니다. GetCustomEndpoint API를 사용하여 엔드포인트 정보를 반환합니다. 자세한 내용은 관리형 통합 API 참조 안내서의 GetCustomEndpoint API 및 RegisterCustomEndpoint API를 참조하세요.

프로비저닝 프로필 생성

플릿 프로비저닝 방법을 정의하는 프로비저닝 프로파일을 생성합니다. 클라우드 환경에서 CreateProvisioningProfile API를 실행하여 디바이스 인증을 위한 클레임 인증서와 프라이빗 키를 반환합니다.

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

응답의 예

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

```
    "ProvisioningType":"FLEET_PROVISIONING"}
```

corePKCS11 플랫폼 추상화 라이브러리(PAL)를 구현하여 corePKCS11 라이브러리가 디바이스와 함께 작동하도록 할 수 있습니다. corePKCS11 PAL 포트는 클레임 인증서와 프라이빗 키를 저장할 위치를 제공해야 합니다. 이 기능을 사용하면 디바이스의 프라이빗 키와 인증서를 안전하게 저장할 수 있습니다. 프라이빗 키와 인증서를 하드웨어 보안 모듈(HSM) 또는 신뢰할 수 있는 플랫폼 모듈(TPM)에 저장할 수 있습니다.

관리형 사물 생성

CreateManagedThing API를 사용하여 관리형 통합 클라우드에 디바이스를 등록합니다. 디바이스의 일련 번호(SN)와 범용 제품 코드(UPC)를 포함합니다.

```
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;"
```

다음은 샘플 API 응답을 표시합니다.

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

API는 프로비저닝 검증에 사용할 수 있는 관리형 사물 ID를 반환합니다. 프로비저닝 트랜잭션 중에 승인된 관리형 사물과 일치하는 디바이스 일련 번호(SN) 및 범용 제품 코드(UPC)를 제공해야 합니다. 트랜잭션은 다음과 유사한 결과를 반환합니다.

```
/**
 * @brief Device info structure.
 */
typedef struct iotmiDev_DeviceInfo
{
    char serialNumber[ IOTMI_DEVICE_MAX_SERIAL_NUMBER_LENGTH + 1U ];
    char universalProductCode[ IOTMI_DEVICE_MAX_UPC_LENGTH + 1U ];
    char internationalArticleNumber[ IOTMI_DEVICE_MAX_EAN_LENGTH + 1U ];
} iotmiDev_DeviceInfo_t;
```

SDK 사용자 Wi-Fi 프로비저닝

디바이스 제조업체와 솔루션 공급자는 Wi-Fi 자격 증명을 수신하고 구성하기 위한 자체 독점 Wi-Fi 프로비저닝 서비스를 보유하고 있습니다. Wi-Fi 프로비저닝 서비스에는 전용 모바일 앱, Bluetooth Low Energy(BLE) 연결 및 기타 독점 프로토콜을 사용하여 초기 설정 프로세스를 위해 Wi-Fi 자격 증명을 안전하게 전송하는 작업이 포함됩니다.

End 디바이스 SDK의 소비자는 Wi-Fi 프로비저닝 서비스를 구현해야 하며 디바이스는 Wi-Fi 네트워크에 연결할 수 있습니다.

클레임별 플릿 프로비저닝

최종 사용자는 권한 부여자를 사용하여 고유한 인증서를 프로비저닝하고 클레임별 프로비저닝을 사용하여 관리형 통합에 등록할 수 있습니다.

클라이언트 ID는 프로비저닝 템플릿 응답 또는 디바이스 인증서에서 가져올 수 있습니다. <common name>“_”<serial number>

관리형 사물 기능

프로비저닝 담당자는 관리형 사물 기능을 검색한 다음 관리형 통합에 해당 기능을 업로드합니다. 앱 및 기타 서비스에서 액세스할 수 있는 기능을 제공합니다. 디바이스, 기타 웹 클라이언트 및 서비스는 MQTT 및 예약된 MQTT 주제를 사용하거나 REST API를 사용하여 HTTP를 사용하여 기능을 업데이트할 수 있습니다.

Over-the-Air 업데이트

OTA 아키텍처 개요

Over-the-Air(OTA) 업데이트 프로세스에는 디바이스에 펌웨어 업데이트를 제공하기 위해 함께 작동하는 여러 구성 요소가 포함됩니다. 다음 다이어그램은 End device SDK, Hub SDK 및 기능 간의 상호 작용을 통해 OTA 업데이트 요청을 처리하는 방법을 보여줍니다.

OTA 업데이트 아키텍처는 다음 구성 요소로 구성됩니다.

- 고객: S3 버킷에 작업 문서를 업로드하고 API를 통해 업데이트를 시작합니다.
- OTA 서비스: 작업 생성, 검증 및 관리 처리
- AWS IoT 작업: 작업 실행 및 디바이스로의 전송을 관리합니다.

- 디바이스: Harmony SDK를 사용하여 업데이트 수신 및 적용

사전 조건

OTA 작업을 생성하기 전에 다음 사전 조건을 구성해야 합니다.

Amazon S3 액세스 구성

OTA 업데이트를 활성화하려면 작업 문서를 Amazon S3 버킷에 업로드하고 적절한 액세스 권한을 구성해야 합니다.

1. OTA 작업 문서를 S3 버킷에 업로드
2. 작업 문서에 대한 관리형 통합 액세스 권한을 부여하는 Amazon S3 버킷 정책을 추가합니다.

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"
      ]
    }
  ]
}
```

Over-the-Air(OTA) 작업 구현

업데이트 요구 사항과 디바이스 대상 지정 전략에 따라 두 가지 방법으로 OTA 작업을 생성할 수 있습니다.

일회성 OTA 작업 업데이트

일회성 OTA 작업에는 OTA 업데이트를 수행하기 위한 정적 대상 목록(ManagedThings)이 포함되어 있습니다. 한 번에 최대 100개의 대상을 추가할 수 있습니다. 워크플로는 관리형 통합 추상화 계층을 유지하면서 플릿 인덱싱과 함께 AWS IoT 작업을 사용합니다.

다음 예제를 사용하여 일회성 OTA 작업을 생성합니다.

```
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"}'
```

연속 OTA 작업 업데이트

OTA(Over-the-Air) 그룹화 워크플로를 사용하면 관리형 통합 추상화 계층을 유지하면서 플릿 인덱싱이 포함된 AWS IoT 작업을 사용하여 특정 속성을 기반으로 디바이스 그룹에 펌웨어 업데이트를 배포할 수 있습니다. 연속 OTA 태스크는 특정 대상 대신 쿼리 문자열을 사용합니다. 쿼리 기준과 일치하는 모든 디바이스는 OTA 업데이트를 거치며 쿼리 기준은 지속적으로 재평가됩니다. 일치하는 대상에는 작업 배포가 있습니다.

사전 조건 구성

연속 OTA 작업을 생성하기 전에 다음 사전 조건을 완료합니다.

1. CreateManagedThing API를 호출하여 관리형 사물을 생성하고 플릿 프로비저닝을 수행합니다.
2. 쿼리 대상 지정을 위해 관리형 사물에 메타데이터 속성을 추가합니다.

UpdateManagedThing API를 ManagedThing 사용하여에 속성 및 메타데이터를 추가합니다.

```
aws iotmanagedintegrations update-managed-thing \
  --managed-thing-id "YOUR_MANAGED_THING_ID" \
  --meta-data '{"owner":"managedintegrations","version":"1.0"}'
```

다음 예제를 사용하여 연속 OTA 작업을 생성합니다.

```
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"}'
```

지속적 OTA 워크플로 이해

연속 OTA 업데이트 워크플로는 다음 단계를 따릅니다.

1. UpdateManagedThing API를 사용하여 속성을 사용하여 관리형 사물을 업데이트합니다.
2. 특정 디바이스 속성을 대상으로 하는 쿼리 문자열을 사용하여 OTA 작업을 생성합니다.
3. OTA 서비스는 쿼리 속성을 AWS IoT Core 기반으로에 동적 사물 그룹을 생성합니다.
4. 일치하는 디바이스에서 업데이트를 실행하는 IoT 작업
5. ListOtaTaskExecutions API를 통해 진행 상황을 모니터링하거나 Kinesis 스트림을 통해 OTA 알림을 모니터링합니다(활성화된 경우).

관리형 통합 OTA와 IoT 작업의 차이점

관리형 통합 OTA와 IoT 작업의 근본적인 차이점은 서비스 오케스트레이션과 자동화에 있습니다. 관리형 통합 OTA는 다중 서비스 조정의 복잡성을 추상화하는 단일 서비스 솔루션을 제공합니다.

관리형 통합 OTA가 자동으로 수행하는 작업:

- 동적 사물 그룹 생성: 쿼리 기준에 따라 사물 그룹을 자동으로 생성합니다 AWS IoT Core .
- 대상 확인: 쿼리 문자열(예: `attributes.owner=managedintegrations`)을 실제 디바이스 대상으로 변환합니다.
- 서비스 통합: AWS IoT Core, IoT 작업 및 플릿 인덱싱 서비스 간에 원활하게 조정됩니다.
- 수명 주기 관리: 생성부터 실행 모니터링까지 전체 OTA 워크플로를 처리합니다.

MI OTA가 제거하는 기능:

- 에서 사물 그룹 생성 AWS IoT Core.

- 그룹에 사물 추가.
- IoT 작업 생성.

관리형 통합 OTA는 쿼리 문자열을 기반으로 내부적으로 세 가지 작업을 모두 처리하여 기준과 일치하는 디바이스를 자동으로 검색하고, 후드 아래에 IoT 작업을 생성하고, 여러 AWS 서비스와 직접 상호 작용할 필요 없이 전체 OTA 워크플로를 오케스트레이션합니다.

OTA 작업 구성 설정

OTA 업데이트에 대한 구성을 생성하여 디바이스에 업데이트가 롤아웃되는 방식을 제어하고 중단 조건을 설정하며 제한 시간을 구성할 수 있습니다.

예: CreateOtaTaskConfiguration

다음 예제를 사용하여 OTA 작업 구성을 생성합니다.

```
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"
```

OTA 작업에 구성 설정 적용

구성이 생성되면 추가 구성과 함께 `CreateOtaTask` 요청에 `taskConfigurationId` 추가된를 받게 됩니다.

```
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"}'
```

OTA 알림 모니터링

두 가지 방법을 사용하여 OTA 업데이트를 모니터링할 수 있습니다.

Kinesis Data Streams를 통한 푸시 알림

OTA 알림이 활성화되면 업데이트 상태 이벤트가 Kinesis 스트림으로 자동으로 푸시됩니다. 이를 통해 디바이스 간 펌웨어 업데이트 진행 상황을 실시간으로 파악할 수 있습니다.

ListOtaTaskExecutions API로 모니터링

ListOtaTaskExecutions API를 사용하여 관리형 사물의 OTA 업데이트 상태를 수동으로 확인할 수 있습니다.

```
aws iotmanagedintegrations list-ota-task-executions \
  --task-id "task-123456789" \
  --max-results 25
```

응답은 각 관리형 사물에 대한 세부 실행 상태를 제공합니다.

```
{
  "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"
}
```

이 API를 사용하면 타임스탬프 및 현재 상태를 포함하여 특정 OTA 작업의 대상이 되는 각 관리형 사물에 대한 세부 실행 상태를 검색할 수 있습니다.

작업 문서 처리

OTA 작업을 생성할 때 작업 핸들러는 디바이스에서 다음 단계를 실행합니다. 업데이트를 사용할 수 있게 되면 MQTT를 통해 작업 문서를 요청합니다.

1. MQTT 알림 주제를 구독합니다.
2. 대기 중인 작업에 대해 StartNextPendingJobExecution API를 호출합니다.
3. 사용 가능한 작업 문서를 수신합니다.
4. 지정된 제한 시간을 기준으로 업데이트를 처리합니다.

작업 핸들러를 사용하여 애플리케이션은 즉시 조치를 취할지 아니면 지정된 제한 시간까지 기다릴지 결정할 수 있습니다.

OTA 에이전트 구현

관리형 통합에서 작업 문서를 받으면 작업 문서를 처리하고, 업데이트를 다운로드하고, 설치 작업을 수행하는 자체 OTA 에이전트를 구현해야 합니다. OTA 에이전트는 다음 단계를 수행해야 합니다.

1. 펌웨어 Amazon S3 URLs.
2. HTTP를 통해 펌웨어 업데이트를 다운로드합니다.
3. 디지털 서명을 확인합니다.
4. 검증된 업데이트를 설치합니다.
5. SUCCESS 또는 FAILED 상태로 `iotmi\_JobsHandler\_updateJobStatus`를 호출합니다.

디바이스가 OTA 작업을 성공적으로 완료하면 상태가 인 `iotmi\_JobsHandler \_updateJobStatus` API를 호출`JobSucceeded`하여 성공적인 작업을 보고해야 합니다.

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

데이터 모델 코드 생성기

데이터 모델에 코드 생성기를 사용하는 방법을 알아봅니다. 생성된 코드를 사용하여 클라우드와 디바이스 간에 교환되는 데이터 모델을 직렬화하고 역직렬화할 수 있습니다.

프로젝트 리포지토리에는 C 코드 데이터 모델 핸들러를 생성하기 위한 코드 생성 도구가 포함되어 있습니다. 다음 주제에서는 코드 생성기와 워크플로를 설명합니다.

주제

- 코드 생성 프로세스
- 환경 설정
- 디바이스에 대한 코드 생성

코드 생성 프로세스

코드 생성기는 Zigbee Cluster Library(ZCL) Advanced Platform의 Matter Data Model(.matter 파일) AWS구현, 사전 처리를 처리하는 Python 플러그인, 코드 구조를 정의하는 Jinja2 템플릿이라는 세 가지

기본 입력에서 C 소스 파일을 생성합니다. 생성 중에 Python 플러그인은 전역 유형 정의를 추가하고, 종속성을 기반으로 데이터 유형을 구성하고, 템플릿 렌더링을 위한 정보의 형식을 지정하여 .matter 파일을 처리합니다.

다음 이미지는 C 소스 파일을 생성하는 코드 생성기를 설명합니다.

End 디바이스 SDK에는 connectedhomeip 프로젝트codegen.py에서와 함께 작동하는 Python 플러그인과 Jinja2 템플릿이 포함되어 있습니다. 이 조합은 .matter 파일 입력을 기반으로 각 클러스터에 대해 여러 C 파일을 생성합니다.

다음 하위 주제에서는 이러한 파일을 설명합니다.

- Python 플러그인
- Jinja2 템플릿
- (선택 사항) 사용자 지정 스키마

Python 플러그인

코드 생성기인는 `codegen.py`.matter 파일을 구문 분석하고 정보를 Python 객체로 플러그인에 전송합니다. 플러그인 파일은이 데이터를 `iotmi_data_model.py` 사전 처리하고 제공된 템플릿으로 소스를 렌더링합니다. 사전 처리에는 다음이 포함됩니다.

1. 글로벌 유형`codegen.py`과 같이에서 사용할 수 없는 정보 추가
2. 데이터 유형에 대한 토폴로지 정렬을 수행하여 올바른 정의 순서 설정

> **Note**
>
> 토폴로지 정렬은 종속 유형이 원래 순서에 관계없이 종속성 뒤에 정의되도록 합니다.

Jinja2 템플릿

엔드 디바이스 SDK는 데이터 모델 핸들러 및 하위 수준 C-함수에 맞게 조정된 Jinja2 템플릿을 제공합니다.

Jinja2 템플릿

템플릿	생성된 소스	설명
cluster.h.jinja	iotmi_device_<cluster>.h	하위 수준 C 함수 헤더 파일을 생성합니다.
cluster.c.jinja	iotmi_device_<cluster>.c	데이터 모델 핸들러를 사용하여 콜백 함수 포인터를 구현하고 등록합니다.
cluster_type_helpers.h.jinja	iotmi_device_type_helpers_<cluster>.h	데이터 유형에 대한 함수 프로토타입을 정의합니다.
cluster_type_helpers.c.jinja	iotmi_device_type_helpers_<cluster>.c	클러스터별 열거, 비트맵, 목록 및 구조에 대한 데이터 형식 함수 프로토타입을 생성합니다.
iot_device_dm_types.h.jinja	iotmi_device_dm_types.h	글로벌 데이터 형식에 대한 C 데이터 형식을 정의합니다.
iot_device_type_helpers_global.h.jinja	iotmi_device_type_helpers_global.h	글로벌 작업에 대한 C 데이터 형식을 정의합니다.
iot_device_type_helpers_global.c.jinja	iotmi_device_type_helpers_global.c	부울, 정수, 부동 소수점, 문자열, 비트맵, 목록 및 구조를 포함한 표준 데이터 형식을 선언합니다.

(선택 사항) 사용자 지정 스키마

End Device SDK는 표준화된 코드 생성 프로세스를 사용자 지정 스키마와 결합합니다. 이를 통해 디바이스 및 디바이스 소프트웨어에 대한 Matter Data Model을 확장할 수 있습니다. 사용자 지정 스키마는 디바이스 device-to-cloud 통신을 위한 디바이스 기능을 설명하는 데 도움이 될 수 있습니다.

형식, 구조 및 요구 사항을 포함하여 관리형 통합 데이터 모델에 대한 자세한 내용은 섹션을 참조하세요관리형 통합 데이터 모델.

codegen.py 도구를 사용하여 다음과 같이 사용자 지정 스키마에 대한 C 소스 파일을 생성합니다.

> **Note**
>
> 각 사용자 지정 클러스터에는 다음 세 파일에 대해 동일한 클러스터 ID가 필요합니다.

- 클라우드에서 새 사용자 지정 클러스터를 생성하기 위한 기능 보고를 위한 클러스터 표현을 제공하는 JSON 형식으로 사용자 지정 스키마를 생성합니다. 샘플 파일은에 있습니다codegen/custom_schemas/custom.SimpleLighting@1.0.
- 사용자 지정 스키마와 동일한 정보가 포함된 XML 형식으로 ZCL(Zigbee Cluster Library) 정의 파일을 생성합니다. ZAP 도구를 사용하여 ZCL XML에서 Matter IDL 파일을 생성합니다. 샘플 파일은에 있습니다codegen/zcl/custom.SimpleLighting.xml.
- ZAP 도구의 출력은 Matter IDL File (.matter)이며 사용자 지정 스키마에 해당하는 Matter 클러스터를 정의합니다. End device SDK용 C 소스 파일을 생성하는 codegen.py 도구의 입력입니다. 샘플 파일은에 있습니다codegen/matter_files/custom-light.matter.

사용자 지정 관리형 통합 데이터 모델을 코드 생성 워크플로에 통합하는 방법에 대한 자세한 지침은 섹션을 참조하세요디바이스에 대한 코드 생성.

환경 설정

codegen.py 코드 생성기를 사용하도록 환경을 구성하는 방법을 알아봅니다.

주제

- 사전 조건
- 환경 구성

사전 조건

환경을 구성하기 전에 다음 항목을 설치합니다.

- Git
- Python 3.10 이상
- Poetry 1.2.0 이상

환경 구성

다음 절차에 따라 codegen.py 코드 생성기를 사용하도록 환경을 구성합니다.

1. 에서 최신 버전의 End Device SDK를 다운로드합니다 AWS Management Console.
2. Python 환경을 설정합니다. Codegen 프로젝트는 Python 기반이며 종속성 관리에 Poetry를 사용합니다.
   - `codegen` 디렉터리에서 poetry를 사용하여 프로젝트 종속성을 설치합니다.

     ```
     poetry run poetry install --no-root
     ```
3. 리포지토리를 설정합니다.
   a. connectedhomeip 리포지토리를 복제합니다. 코드 생성을 위해 `connectedhomeip/scripts/` 폴더에 있는 `codegen.py` 스크립트를 사용합니다. 자세한 내용은 GitHub의 connectedhomeip를 참조하세요. GitHub

      ```
      git clone -b v1.4.0.0 https://github.com/project-chip/connectedhomeip.git
      ```
   b. `IoT-managed-integrations-End-Device-SDK` 루트 폴더와 동일한 수준에서 복제합니다. 폴더 구조는 다음과 일치해야 합니다.

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

> **Note**
>
> 하위 모듈을 재귀적으로 복제할 필요가 없습니다.

디바이스에 대한 코드 생성

관리형 통합 코드 생성 도구를 사용하여 디바이스에 대한 사용자 지정 C 코드를 생성합니다. 이 섹션에서는 SDK에 포함된 샘플 파일 또는 자체 사양에서 코드를 생성하는 방법을 설명합니다. 생성 스크립트를 사용하고, 워크플로 프로세스를 이해하고, 디바이스 요구 사항에 맞는 코드를 생성하는 방법을 알아봅니다.

주제

사전 조건

1. Python 3.10 이상.

2. 코드 생성을 위해 .matter 파일로 시작합니다. End 디바이스 SDK는에서 `codgen/matter_files folder`두 개의 샘플 파일을 제공합니다.

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

> **Note**
>
> 이러한 샘플 파일은 데모 애플리케이션 클러스터에 대한 코드를 생성합니다.

코드 생성

이 명령을 실행하여 out 폴더에 코드를 생성합니다.

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

사용자 지정 .matter 파일에 대한 코드 생성

특정 `.matter` 파일에 대한 코드를 생성하거나 자체 `.matter` 파일을 제공하려면 다음 작업을 수행합니다.

사용자 지정 .matter 파일에 대한 코드를 생성하려면

1. .matter 파일 준비
2. 생성 명령을 실행합니다.

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

(선택 사항) 사용자 지정 스키마를 사용하여 코드를 생성하려면

1. JSON 형식으로 사용자 지정 스키마 준비
2. 생성 명령을 실행합니다.

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

위 명령은 여러 구성 요소를 사용하여 `.matter` 파일을 C 코드로 변환합니다.

- `codegen.py` ConnectedHomeIP 프로젝트에서
- 에 위치한 Python 플러그인 `codegen/py_scripts/iotmi_data_model.py`
- `codegen/py_scripts/templates` 폴더의 Jinja2 템플릿

플러그인은 Jinja2 템플릿에 전달할 변수를 정의한 다음 최종 C 코드 출력을 생성하는 데 사용됩니다. `--format` 플래그를 추가하면 생성된 코드에 Clang 형식이 적용됩니다.

코드 생성 워크플로

코드 생성 프로세스는 유틸리티 함수와를 통한 토폴로지 정렬을 사용하여 .matter 파일 데이터 구조를 구성합니다`topsort.py`. 이렇게 하면 데이터 유형과 해당 종속성을 적절하게 정렬할 수 있습니다.

그런 다음 스크립트는 .matter 파일 사양을 Python 플러그인 처리와 결합하여 필요한 정보를 추출하고 형식을 지정합니다. 마지막으로 Jinja2 템플릿 형식을 적용하여 최종 C 코드 출력을 생성합니다.

이 워크플로를 사용하면 .matter 파일의 디바이스별 요구 사항이 관리형 통합 시스템과 통합되는 기능적 C 코드로 정확하게 변환됩니다.

하위 수준 C-Function APIs

제공된 하위 수준 C-Function APIs. 이 섹션에서는 효율적인 디바이스-클라우드 상호 작용을 위해 AWS 데이터 모델의 각 클러스터에 사용할 수 있는 API 작업에 대해 설명합니다. 콜백 함수를 구현하고, 이벤트를 내보내고, 속성 변경 사항을 알리고, 디바이스 엔드포인트에 클러스터를 등록하는 방법을 알아봅니다.

주요 API 구성 요소는 다음과 같습니다.

1. 속성 및 명령에 대한 콜백 함수 포인터 구조

2. 이벤트 방출 함수
3. 속성 변경 알림 함수
4. 클러스터 등록 함수

이러한 APIs 구현하면 디바이스의 물리적 운영과 관리형 통합 클라우드 기능 간에 브리지를 생성하여 원활한 통신과 제어를 보장할 수 있습니다.

다음 섹션에서는 OnOff 클러스터 API를 보여줍니다.

OnOff 클러스터 API

OnOff.xml 클러스터는와 같은 속성과 명령을 지원합니다.

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

  각 명령에 대해 구현을 연결하는 데 사용할 수 있는 1:1 매핑된 함수 포인터를 제공합니다.

속성 및 명령에 대한 모든 콜백은 클러스터 이름의 C 구조체 내에 정의됩니다.

예제 C 구조

```
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);
};
```

C 구문 외에도 모든 속성에 대해 속성 변경 보고 함수가 정의됩니다.

```
/* 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);
```

이벤트 보고 함수는 모든 클러스터별 이벤트에 대해 정의됩니다. OnOff 클러스터는 이벤트를 정의하지 않으므로 다음은 CameraAvStreamManagement 클러스터의 예입니다.

```
/* 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);
```

각 클러스터에는 레지스터 함수도 있습니다.

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

등록 함수에 전달된 사용자 포인터는 콜백 함수에 전달됩니다.

관리형 통합의 기능 및 디바이스 상호 작용

이 섹션에서는 C-Function 구현의 역할과 디바이스와 관리형 통합 디바이스 기능 간의 상호 작용을 설명합니다.

주제

- 원격 명령 처리
- 원치 않는 이벤트 처리

원격 명령 처리

원격 명령은 End Device SDK와 기능 간의 상호 작용에 의해 처리됩니다. 다음 작업은이 상호 작용을 사용하여 전구를 켤 수 있는 방법의 예를 설명합니다.

MQTT 클라이언트가 페이로드를 수신하고 데이터 모델 핸들러에 전달합니다.

원격 명령을 보내면 MQTT 클라이언트는 관리형 통합에서 JSON 형식으로 메시지를 수신합니다. 그런 다음 페이로드를 데이터 모델 핸들러에 전달합니다. 예를 들어 관리형 통합을 사용하여 전구를 켜려고 한다고 가정해 보겠습니다. 전구에는 OnOff 클러스터를 지원하는 엔드포인트 #1이 있습니다. 이 경우 전구를 켜는 명령을 보내면 관리형 통합은 MQTT를 통해 디바이스에 요청을 보냅니다. 즉, 엔드포인트 #1에서 On 명령을 호출하려고 합니다.

데이터 모델 핸들러는 콜백 함수를 확인하고 호출합니다.

데이터 모델 핸들러는 JSON 요청을 구문 분석합니다. 요청에 속성 또는 작업이 포함된 경우 데이터 모델 핸들러는 엔드포인트를 찾고 해당 콜백 함수를 순차적으로 호출합니다. 예를 들어 전구의 경우 데이터 모델 핸들러가 MQTT 메시지를 수신하면 OnOff 클러스터에 정의된 On 명령에 해당하는 콜백 함수가 엔드포인트 #1에 등록되었는지 확인합니다.

핸들러 및 C-Function 구현에서 명령 실행

데이터 모델 핸들러는 찾은 적절한 콜백 함수를 호출하고 호출합니다. 그런 다음 C-Function 구현은 해당 하드웨어 함수를 호출하여 물리적 하드웨어를 제어하고 실행 결과를 반환합니다. 예를 들어 전구의 경우 데이터 모델 핸들러는 콜백 함수를 호출하고 실행 결과를 저장합니다. 그러면 콜백 함수가 전구를 켭니다.

Data Model Handler에서 실행 결과 반환

모든 콜백 함수가 호출되면 데이터 모델 핸들러는 모든 결과를 결합합니다. 그런 다음 응답을 JSON 형식으로 압축하고 MQTT 클라이언트를 사용하여 관리형 통합 클라우드에 결과를 게시합니다. 전구의 경우 응답의 MQTT 메시지에는 콜백 함수에 의해 전구가 켜져 있다는 결과가 포함됩니다.

원치 않는 이벤트 처리

요청되지 않은 이벤트는 End 디바이스 SDK와 기능 간의 상호 작용에서도 처리됩니다. 다음 작업은 방법을 설명합니다.

디바이스가 데이터 모델 핸들러로 알림 전송

디바이스에서 물리적 버튼을 눌렀을 때와 같이 속성 변경 또는 이벤트가 발생하면 C-Function 구현은 원치 않는 이벤트 알림을 생성하고 해당 알림 함수를 호출하여 알림을 데이터 모델 핸들러로 보냅니다.

데이터 모델 핸들러에서 알림 번역

데이터 모델 핸들러는 수신된 알림을 처리하고 이를 AWS 데이터 모델로 변환합니다.

Data Model Handler가 클라우드에 알림 게시

그런 다음 Data Model Handler는 MQTT 클라이언트를 사용하여 관리형 통합 클라우드에 원치 않는 이벤트를 게시합니다.

End Device SDK 시작하기

다음 단계에 따라 Linux 디바이스에서 End device SDK를 실행합니다. 이 섹션에서는 환경 설정, 네트워크 구성, 하드웨어 함수 구현 및 엔드포인트 구성을 안내합니다.

> **Important**
>
> `examples` 디렉터리의 데모 애플리케이션과의 플랫폼 추상화 계층(PAL) 구현`platform/posix`은 참조용일 뿐입니다. 프로덕션 환경에서는 사용하지 마십시오.

다음 절차의 각 단계를 주의 깊게 검토하여 관리형 통합과의 디바이스 통합이 적절한지 확인합니다.

엔드 디바이스 SDK 통합

1. Amazon EC2 인스턴스 설정

   에 로그인 AWS Management Console 하고 Amazon Linux AMI를 사용하여 Amazon EC2 인스턴스를 시작합니다. Amazon Elastic Container Registry 사용 설명서의 Amazon Amazon EC2 시작하기를 참조하세요.

2. 빌드 환경 설정

   Amazon Linux 2023/x86_64에서 개발 호스트로 코드를 빌드합니다. 필요한 빌드 종속성을 설치합니다.

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

3. (선택 사항) 네트워크 설정

   엔드 디바이스 SDK는 물리적 하드웨어와 함께 사용하는 것이 가장 좋습니다. Amazon EC2를 사용하는 경우이 단계를 따르지 마세요.

샘플 애플리케이션을 사용하기 전에 Amazon EC2를 사용하지 않는 경우 네트워크를 초기화하고 디바이스를 사용 가능한 Wi-Fi 네트워크에 연결합니다. 디바이스 프로비저닝 전에 네트워크 설정을 완료합니다.

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

4. 프로비저닝 파라미터 구성

> **Note**
>
> 계속 진행하기 전에 Provisionee를 따라 클레임 인증서와 프라이빗 키를 가져옵니다.

다음 프로비저닝 파라미터를 `example/project_name/device_config.sh` 사용하여 구성 파일을 수정합니다.

프로비저닝 파라미터

매크로 파라미터	설명	이 정보를 가져오는 방법
`IOTMI_ROOT_CA_PATH`	루트 CA 인증서 파일입니다.	AWS IoT Core 개발자 안내서의 Amazon Root CA 인증서 다운로드 섹션에서이 파일을 다운로드할 수 있습니다.
`IOTMI_CLAIM_CERTIFICATE_PATH`	클레임 인증서 파일의 경로입니다.	클레임 인증서와 프라이빗 키를 얻으려면 CreateProvisioningProfile API를 사용하여 프로비저닝 프로파일을 생성합니다. 지침은 프로비저닝 프로필 생성 섹션을 참조하세요.
`IOTMI_CLAIM_PRIVATE_KEY_PATH`	클레임 프라이빗 키 파일의 경로입니다.	
`IOTMI_MANAGEDINTEGRATIONS_ENDPOINT`	관리형 통합을 위한 엔드포인트 URL입니다.	관리형 통합 엔드포인트를 가져오려면 RegisterCustomEndpoint API를 사용합니다. 지침은 사용자 지정 엔드포인트 등록 섹션을 참조하세요.

매크로 파라미터	설명	이 정보를 가져오는 방법
IOTMI_MAN AGEDINTEG RATIONS_E NDPOINT_PORT	관리형 통합 엔드포인트의 포트 번호	기본적으로 포트 8883은 MQTT 게시 및 구독 작업에 사용됩니다. 포트 443은 디바이스가 사용하는 애플리케이션 계층 프로토콜 협상(ALPN) TLS 확장에 대해 설정됩니다.

5. 데모 애플리케이션 빌드 및 실행

   이 섹션에서는 CMake를 빌드 시스템으로 사용하는 간단한 보안 카메라와 공기청정기라는 두 가지 Linux 데모 애플리케이션을 보여줍니다.

   a. 간단한 보안 카메라 애플리케이션

      애플리케이션을 빌드하고 실행하려면 다음 명령을 실행합니다.

```
>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
```

      이 데모에서는 RTC 세션 컨트롤러 및 레코딩 클러스터가 있는 시뮬레이션된 카메라에 대해 하위 수준의 C-함수를 구현합니다. 실행하기 프로비저닝 담당자 워크플로 전에에 언급된 흐름을 완료합니다.

      데모 애플리케이션의 샘플 출력:

```
[2406832727][MAIN][INFO] ======= Device initialization and WIFI provisioning
 =======
[2406832728][MAIN][INFO] fleetProvisioningTemplateName: XXXXXXXXXXX
[2406832728][MAIN][INFO] managedintegrationsEndpoint: XXXXXXXXX.account-prefix-
ats.iot.region.amazonaws.com
[2406832728][MAIN][INFO] pDeviceSerialNumber: XXXXXXXXXXXX
[2406832728][MAIN][INFO] universalProductCode: 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 XXXXXXXXXXXXXX.account-prefix-ats.iot.region.amazonaws.com
[2025-01-02 01:43:11.405119166][iotmi_device_sdkLog][INFO]
[2025-01-02 01:43:11.844812513][iotmi_device_sdkLog][INFO] [2406833168]
[MQTT_AGENT][INFO]
[2025-01-02 01:43:11.844842576][iotmi_device_sdkLog][INFO] TLS session
 connected
[2025-01-02 01:43:11.844852105][iotmi_device_sdkLog][INFO]
[2025-01-02 01:43:12.296421687][iotmi_device_sdkLog][INFO] [2406833620]
[MQTT_AGENT][INFO]
[2025-01-02 01:43:12.296449663][iotmi_device_sdkLog][INFO] Session present: 0.
[2025-01-02 01:43:12.296458997][iotmi_device_sdkLog][INFO]
[2025-01-02 01:43:12.296467793][iotmi_device_sdkLog][INFO] [2406833620]
[MQTT_AGENT][INFO]
[2025-01-02 01:43:12.296476275][iotmi_device_sdkLog][INFO] MQTT connect with
 clean session.
[2025-01-02 01:43:12.296484350][iotmi_device_sdkLog][INFO]
[2025-01-02 01:43:13.171056119][iotmi_device_sdkLog][INFO] [2406834494]
[FLEET_PROVISIONING][INFO]
[2025-01-02 01:43:13.171082442][iotmi_device_sdkLog][INFO] Received accepted
 response from Fleet Provisioning CreateKeysAndCertificate API.
[2025-01-02 01:43:13.171092740][iotmi_device_sdkLog][INFO]
[2025-01-02 01:43:13.171122834][iotmi_device_sdkLog][INFO] [2406834494]
[FLEET_PROVISIONING][INFO]
[2025-01-02 01:43:13.171132400][iotmi_device_sdkLog][INFO] Received privatekey
 and certificate with Id: XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
[2025-01-02 01:43:13.171141107][iotmi_device_sdkLog][INFO]
[2406834494][PKCS11][INFO] Creating a 0x3 type object.
```

```
[2406834494][PKCS11][INFO] Writing certificate into label "Device Cert".
[2406834494][PKCS11][INFO] Creating a 0x1 type object.
[2025-01-02 01:43:18.584615126][iotmi_device_sdkLog][INFO] [2406839908]
[FLEET_PROVISIONING][INFO]
[2025-01-02 01:43:18.584662031][iotmi_device_sdkLog][INFO] Received accepted
 response from Fleet Provisioning RegisterThing API.
[2025-01-02 01:43:18.584671912][iotmi_device_sdkLog][INFO]
[2025-01-02 01:43:19.100030237][iotmi_device_sdkLog][INFO] [2406840423]
[FLEET_PROVISIONING][INFO]
[2025-01-02 01:43:19.100061720][iotmi_device_sdkLog][INFO] Fleet-provisioning
 iteration 1 is successful.
[2025-01-02 01:43:19.100072401][iotmi_device_sdkLog][INFO]
[2406840423][MQTT][ERROR] MQTT Connection Disconnected Successfully
[2025-01-02 01:43:19.216938181][iotmi_device_sdkLog][INFO] [2406840540]
[MQTT_AGENT][INFO]
[2025-01-02 01:43:19.216963713][iotmi_device_sdkLog][INFO] MQTT agent thread
 leaves thread loop for iotmiDev_MQTTAgentStop.
[2025-01-02 01:43:19.216973740][iotmi_device_sdkLog][INFO]
[2406840540][MAIN][INFO] iotmiDev_MQTTAgentStop is called to break thread loop
 function.
[2406840540][MAIN][INFO] Successfully provision the device.
[2406840540][MAIN][INFO] Client ID :
 XXXXXXXXXXXXXXXXXXX_XXXXXXXXXXXXXXXXXXXXXX
[2406840540][MAIN][INFO] Managed thing ID : XXXXXXXXXXXXXXXXXXXXX
[2406840540][MAIN][INFO] ======================== application loop
 =================
[2025-01-02 01:43:19.217094828][iotmi_device_sdkLog][INFO] [2406840540]
[MQTT_AGENT][INFO]
[2025-01-02 01:43:19.217124600][iotmi_device_sdkLog][INFO] Establishing a TLS
 session to XXXXXXXXX.account-prefix-ats.iot.region.amazonaws.com:8883
[2025-01-02 01:43:19.217138724][iotmi_device_sdkLog][INFO]
[2406840540][Cluster OnOff][INFO] exampleOnOffInitCluster() for endpoint#1
[2406840540][MAIN][INFO] Press Ctrl+C when you finish testing...
[2406840540][Cluster ActivatedCarbonFilterMonitoring][INFO]
 exampleActivatedCarbonFilterMonitoringInitCluster() for endpoint#1
[2406840540][Cluster AirQuality][INFO] exampleAirQualityInitCluster() for
 endpoint#1
[2406840540][Cluster CarbonDioxideConcentrationMeasurement][INFO]
 exampleCarbonDioxideConcentrationMeasurementInitCluster() for endpoint#1
[2406840540][Cluster FanControl][INFO] exampleFanControlInitCluster() for
 endpoint#1
[2406840540][Cluster HepaFilterMonitoring][INFO]
 exampleHepaFilterMonitoringInitCluster() for endpoint#1
```

```
[2406840540][Cluster Pm1ConcentrationMeasurement][INFO]
 examplePm1ConcentrationMeasurementInitCluster() for endpoint#1
[2406840540][Cluster Pm25ConcentrationMeasurement][INFO]
 examplePm25ConcentrationMeasurementInitCluster() for endpoint#1
[2406840540][Cluster TotalVolatileOrganicCompoundsConcentrationMeasurement]
[INFO]
 exampleTotalVolatileOrganicCompoundsConcentrationMeasurementInitCluster() for
 endpoint#1
[2025-01-02 01:43:19.648185488][iotmi_device_sdkLog][INFO] [2406840971]
[MQTT_AGENT][INFO]
[2025-01-02 01:43:19.648211988][iotmi_device_sdkLog][INFO] TLS session
 connected
[2025-01-02 01:43:19.648225583][iotmi_device_sdkLog][INFO]

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

b. 간단한 공기청정기 애플리케이션

애플리케이션을 빌드하고 실행하려면 다음 명령을 실행합니다.

```
>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
```

이 데모에서는 2개의 엔드포인트와 지원되는 다음 클러스터가 있는 시뮬레이션된 공기청정기에 대해 하위 수준의 C-Functions를 구현합니다.

공기청정기 엔드포인트에 지원되는 클러스터

엔드포인트	클러스터
엔드포인트 #1: Air Purifier	OnOff
	팬 제어

엔드포인트	클러스터
	HEPA 필터 모니터링
	활성화된 탄소 필터 모니터링
엔드포인트 #2: 대기 품질 센서	대기 품질
	이산화탄소 농도 측정
	Form™ 농도 측정
	Pm25 농도 측정
	Pm1 농도 측정
	총 휘발성 유기 복합 농도 측정

출력은 지원되는 클러스터가 서로 다른 카메라 데모 애플리케이션과 유사합니다.

6. 다음 단계:

관리형 통합 이제 Amazon EC2 인스턴스에서 디바이스 SDK 및 데모 애플리케이션을 종료합니다. 이를 통해 자체 물리적 하드웨어에서 애플리케이션을 개발하고 테스트할 수 있습니다. 이 설정을 사용하면 관리형 통합 서비스를 활용하여 AWS IoT 디바이스를 제어할 수 있습니다.

a. 하드웨어 콜백 함수 개발

하드웨어 콜백 함수를 구현하기 전에 API의 작동 방식을 이해합니다. 이 예제에서는 켜기/끄기 클러스터와 OnOff 속성을 사용하여 디바이스 함수를 제어합니다. API 세부 정보는 섹션을 참조하세요하위 수준 C-Function APIs.

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

/* This implementation for OnOff getter just reads
```

```
   the state from the DeviceState */
iotmiDev_DMStatus exampleGetOnOff(bool *value, void *user)
{
   struct DeviceState *state = (struct DeviceState *)(user);
  *value = state->hwState;
  return iotmiDev_DMStatusOk;
}
```

b. 엔드포인트 및 후크 하드웨어 콜백 함수 설정

함수를 구현한 후 엔드포인트를 생성하고 콜백을 등록합니다. 다음 단계를 완료합니다.

i. 디바이스 에이전트를 생성합니다.

A. 다른 SDK 함수를 호출`iotmiDev_Agent_new()`하기 전에를 사용하여 디바이스 에이전트를 생성합니다.

B. 최소한 구성에는 thingId 및 clientId 파라미터가 포함되어야 합니다.

C. `iotmiDev_Agent_initDefaultConfig()` 함수를 사용하여 대기열 크기 및 최대 엔드포인트와 같은 파라미터에 대해 적절한 기본값을 설정합니다.

D. 리소스 사용을 마치면 `iotmiDev_Agent_free()` 함수로 리소스를 해제합니다. 이렇게 하면 메모리 누수를 방지하고 애플리케이션에서 적절한 리소스 관리를 보장할 수 있습니다.

ii. 지원하려는 각 클러스터 구조에 대해 콜백 함수 포인터를 채웁니다.

iii. 엔드포인트를 설정하고 지원되는 클러스터를 등록합니다.

를 사용하여 엔드포인트를 생성합니다`iotmiDev_Agent_addEndpoint()`. 여기에는 다음이 필요합니다.

A. 고유한 엔드포인트 ID입니다.

B. 설명이 포함된 엔드포인트 이름

C. AWS 데이터 모델 정의를 준수하는 하나 이상의 디바이스 유형입니다.

D. 엔드포인트를 생성한 후 적절한 클러스터별 등록 함수를 사용하여 클러스터를 등록합니다.

E. 각 클러스터 등록에는 속성 및 명령에 대한 콜백 함수가 필요합니다. 시스템은 사용자 컨텍스트 포인터를 콜백에 전달하여 호출 간 상태를 유지합니다.

```
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. 작업 핸들러를 사용하여 작업 문서 가져오기

i. OTA 애플리케이션에 대한 호출을 시작합니다.

```
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. 를 호출`iotmi_JobsHandler_start`하여 작업 핸들러를 초기화합니다.

iii. 를 호출`iotmi_JobsHandler_getJobDocument`하여 관리형 통합에서 작업 문서를 검색합니다.

iv. 작업 문서를 성공적으로 가져오면 `processOTA` 함수에 사용자 지정 OTA 작업을 작성하고 `JobSucceeded` 상태를 반환합니다.

```
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." ) );
```

```
}
```

디바이스에 End 디바이스 SDK 이식

End Device SDK를 디바이스 플랫폼에 이식합니다. 다음 단계에 따라 디바이스를 AWS IoT Device Management에 연결합니다.

End Device SDK 다운로드 및 확인

1. 관리형 통합 콘솔에서 End 디바이스 SDK의 최신 버전을 다운로드합니다.
2. 플랫폼이의 지원되는 플랫폼 목록에 있는지 확인합니다참조: 지원되는 플랫폼.

> **Note**
>
> End Device SDK는 지정된 플랫폼에서 테스트되었습니다. 다른 플랫폼이 작동할 수 있지만 테스트되지 않았습니다.

3. SDK 파일을 워크스페이스로 추출(압축 해제)합니다.
4. 다음 설정을 사용하여 빌드 환경을 구성합니다.
   - 소스 파일 경로
   - 헤더 파일 디렉터리
   - 필요한 라이브러리
   - 컴파일러 및 링커 플래그
5. 플랫폼 추상화 계층(PAL)을 이식하기 전에 플랫폼의 기본 기능이 초기화되었는지 확인합니다. 기능은 다음과 같습니다.
   - 운영 체제 작업
   - 주변 장치
   - 네트워크 인터페이스
   - 플랫폼별 요구 사항

PAL을 디바이스로 이식

1. 기존 플랫폼 디렉터리에서 플랫폼별 구현을 위한 새 디렉터리를 생성합니다. 예를 들어 FreeRTOS를 사용하는 경우에서 디렉터리를 생성합니다`platform/freertos`.

Example SDK 디렉터리 구조

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

2. POSIX 참조 구현 파일(.c 및 .h)을 posix 폴더에서 새 플랫폼 디렉터리로 복사합니다. 이러한 파일은 구현해야 하는 함수에 대한 템플릿을 제공합니다.

   - 자격 증명 스토리지를 위한 플래시 메모리 관리
   - PKCS#11 구현
   - 네트워크 전송 인터페이스
   - 시간 동기화
   - 시스템 재부팅 및 재설정 함수
   - 로깅 메커니즘
   - 디바이스별 구성

3. MBedTLS) 인증을 설정합니다.

   - 플랫폼의 SDK 버전과 일치하는 MBedTLS 버전이 이미 있는 경우 제공된 POSIX 구현을 사용합니다.
   - 다른 TLS 버전을 사용하면 TCP/IP 스택을 사용하여 TLS 스택의 전송 후크를 구현할 수 있습니다.

4. 플랫폼의 MbedTLS 구성을 `platform/posix/mbedtls/mbedtls_config.h`의 SDK 요구 사항과 비교합니다. 필요한 모든 옵션이 활성화되어 있는지 확인합니다.

5. SDK는 coreMQTT를 사용하여 클라우드와 상호 작용합니다. 따라서 다음 구조를 사용하는 네트워크 전송 계층을 구현해야 합니다.

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

자세한 내용은 FreeRTOS 웹 사이트의 전송 인터페이스 설명서를 참조하세요.

6. (선택 사항) SDK는 PCKS#11 API를 사용하여 인증서 작업을 처리합니다. corePKCS는 프로토타이핑을 위한 하드웨어별이 아닌 PKCS#11 구현입니다. 프로덕션 환경에서 신뢰할 수 있는 플랫폼 모듈(TPM), 하드웨어 보안 모듈(HSM) 또는 보안 요소와 같은 보안 암호화 프로세서를 사용하는 것이 좋습니다.

   - 에서 자격 증명 관리를 위해 Linux 파일 시스템을 사용하는 샘플 PKCS#11 구현을 검토합니다`platform/posix/corePKCS11-mbedtls`.
   - 에서 PKCS#11 PAL 계층을 구현합니다`commonDependencies/core_pkcs11/corePKCS11/source/include/core_pkcs11.h`.
   - 에서 Linux 파일 시스템을 구현합니다`platform/posix/corePKCS11-mbedtls/source/iotmi_pal_Pkcs11Operations.c`.
   - 에서 스토리지 유형의 스토어 및 로드 함수를 구현합니다`platform/include/iotmi_pal_Nvm.h`.
   - 에서 표준 파일 액세스를 구현합니다`platform/posix/source/iotmi_pal_Nvm.c`.

   자세한 이식 지침은 FreeRTOS 사용 설명서의 corePKCS11 라이브러리 이식을 참조하세요.

7. 빌드 환경에 SDK 정적 라이브러리를 추가합니다.

   - 라이브러리 경로를 설정하여 링커 문제 또는 기호 충돌을 해결합니다.
   - 모든 종속성이 제대로 연결되었는지 확인

포트 테스트

기존 예제 애플리케이션을 사용하여 포트를 테스트할 수 있습니다. 컴파일은 오류나 경고 없이 완료되어야 합니다.

> **Note**
>
> 가능한 가장 간단한 멀티태스킹 애플리케이션으로 시작하는 것이 좋습니다. 예제 애플리케이션은 이에 상응하는 멀티태스킹을 제공합니다.

1. 에서 예제 애플리케이션을 찾습니다`examples/[device_type_sample]`.
2. `main.c` 파일을 프로젝트로 변환하고 항목을 추가하여 기존 main() 함수를 호출합니다.
3. 데모 애플리케이션을 성공적으로 컴파일할 수 있는지 확인합니다.

기술 참조

주제

- 참조: 지원되는 플랫폼
- 참조: 기술 요구 사항
- 참조: 공통 API

참조: 지원되는 플랫폼

다음 표에는 SDK에 지원되는 플랫폼이 나와 있습니다.

지원하는 플랫폼

플랫폼	아키텍처	운영 체제
Linux x86_64	x86_64	Linux
Ambarella	ARMv8(AArch64)	Linux
AmebaD	Armv8-M 32비트	FreeRTOS
ESP32S3	Xtensa LX7 32비트	FreeRTOS

참조: 기술 요구 사항

다음 표에는 RAM 공간을 포함한 SDK의 기술 요구 사항이 나와 있습니다. End 디바이스 SDK 자체는 동일한 구성을 사용할 때 약 5~10MB의 ROM 공간이 필요합니다.

RAM 공간

SDK 및 구성 요소	스페이스 요구 사항(사용된 바이트)
디바이스 SDK 자체 종료	180KB
기본 MQTT 에이전트 명령 대기열	480바이트(구성 가능)
기본 MQTT 에이전트 수신 대기열	320바이트(구성 가능)

참조: 공통 API

이 섹션은 클러스터에만 국한되지 않는 API 작업 목록입니다.

```
/* 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);
```

에 대한 관리형 통합의 보안 AWS IoT Device Management

의 클라우드 보안 AWS 이 최우선 순위입니다. AWS 고객은 보안에 가장 민감한 조직의 요구 사항을 충족하도록 구축된 데이터 센터 및 네트워크 아키텍처의 이점을 누릴 수 있습니다.

보안은 AWS 와 사용자 간의 공동 책임입니다. 공동 책임 모델은 이 사항을 클라우드의 보안 및 클라우드 내 보안으로 설명합니다.

- 클라우드 보안 - AWS 는에서 AWS 서비스를 실행하는 인프라를 보호할 책임이 있습니다 AWS 클라우드. AWS 또한는 안전하게 사용할 수 있는 서비스를 제공합니다. 타사 감사자는 AWS 규정 준수 프로그램 일환으로 보안의 효과를 정기적으로 테스트하고 확인합니다. 관리형 통합에 적용되는 규정 준수 프로그램에 대한 자세한 내용은 규정 준수 프로그램 AWS 제공 범위 내 서비스규정 준수 프로그램 .
- 클라우드의 보안 - 사용자의 책임은 사용하는 AWS 서비스에 따라 결정됩니다. 또한 귀하는 귀사의 데이터 민감도, 귀사의 요구 사항, 관련 법률 및 규정을 비롯한 기타 요소에 대해서도 책임이 있습니다.

이 설명서는 관리형 통합을 사용할 때 공동 책임 모델을 적용하는 방법을 이해하는 데 도움이 됩니다. 다음 주제에서는 보안 및 규정 준수 목표에 맞게 관리형 통합을 구성하는 방법을 보여줍니다. 또한 관리형 통합 리소스를 모니터링하고 보호하는 데 도움이 되는 다른 AWS 서비스를 사용하는 방법을 알아봅니다.

주제

- 관리형 통합의 데이터 보호
- 관리형 통합을 위한 ID 및 액세스 관리
- C2C 워크플로 AWS Secrets Manager 의 데이터 보호에 사용
- 관리형 통합에 대한 규정 준수 검증
- 인터페이스 VPC 엔드포인트와 관리형 통합 사용
- AWS IoT Device Management FIPS 엔드포인트의 관리형 통합에 연결

관리형 통합의 데이터 보호

AWS 공동 책임 모델 통합의 데이터 보호에 적용됩니다 AWS IoT Device Management. 이 모델에 설명된 대로 AWS 는 모든를 실행하는 글로벌 인프라를 보호할 책임이 있습니다 AWS 클라우드. 사용자

는 인프라에서 호스팅되는 콘텐츠를 관리해야 합니다. 사용하는 AWS 서비스 의 보안 구성과 관리 태스크에 대한 책임도 사용자에게 있습니다. 데이터 프라이버시에 대한 자세한 내용은 데이터 프라이버시 FAQ를 참조하세요. 유럽의 데이터 보호에 대한 자세한 내용은 AWS 보안 블로그의 AWS 공동 책임 모델 및 GDPR 블로그 게시물을 참조하세요.

데이터 보호를 위해 자격 증명을 보호하고 AWS 계정 AWS IAM Identity Center 또는 AWS Identity and Access Management (IAM)를 사용하여 개별 사용자를 설정하는 것이 좋습니다. 이렇게 하면 개별 사용자에게 자신의 직무를 충실히 이행하는 데 필요한 권한만 부여됩니다. 또한 다음과 같은 방법으로 데이터를 보호하는 것이 좋습니다.

- 각 계정에 다중 인증(MFA)을 사용하세요.
- SSL/TLS를 사용하여 AWS 리소스와 통신합니다. TLS 1.2는 필수이며 TLS 1.3을 권장합니다.
- 를 사용하여 API 및 사용자 활동 로깅을 설정합니다 AWS CloudTrail. CloudTrail 추적을 사용하여 AWS 활동을 캡처하는 방법에 대한 자세한 내용은 AWS CloudTrail 사용 설명서의 CloudTrail 추적 작업을 참조하세요.
- 내부의 모든 기본 보안 제어와 함께 AWS 암호화 솔루션을 사용합니다 AWS 서비스.
- Amazon S3에 저장된 민감한 데이터를 검색하고 보호하는 데 도움이 되는 Amazon Macie와 같은 고급 관리형 보안 서비스를 사용하세요.
- 명령줄 인터페이스 또는 API를 AWS 통해에 액세스할 때 FIPS 140-3 검증 암호화 모듈이 필요한 경우 FIPS 엔드포인트를 사용합니다. 사용 가능한 FIPS 엔드포인트에 대한 자세한 내용은 Federal Information Processing Standard(FIPS) 140-3을 참조하세요.

고객의 이메일 주소와 같은 기밀 정보나 중요한 정보는 태그나 이름 필드와 같은 자유 형식 텍스트 필드에 입력하지 않는 것이 좋습니다. 여기에는 콘솔 AWS CLI, API 또는 AWS SDKs를 사용하여 AWS IoT Device Management 또는 기타 AWS 서비스 에 대한 관리형 통합으로 작업하는 경우가 포함됩니다. 이름에 사용되는 태그 또는 자유 형식 텍스트 필드에 입력하는 모든 데이터는 청구 또는 진단 로그에 사용될 수 있습니다. 외부 서버에 URL을 제공할 때 해당 서버에 대한 요청을 검증하기 위해 자격 증명을 URL에 포함해서는 안 됩니다.

관리형 통합을 위한 저장 데이터 암호화

용 관리형 통합은 기본적으로 AWS IoT Device Management 암호화 키를 사용하여 저장된 민감한 고객 데이터를 암호화합니다.

관리형 통합 고객의 민감한 데이터를 보호하는 데 사용되는 두 가지 유형의 암호화 키가 있습니다.

고객 관리형 키(CMK)

관리형 통합은 생성, 소유 및 관리할 수 있는 대칭 고객 관리형 키 사용을 지원합니다. 사용자는 키 정책, IAM 정책 및 권한 부여의 설정 및 유지 관리, 활성화 및 비활성화, 암호화 구성 요소 교체, 태그 추가, KMS 키를 가리키는 별칭 생성, KMS 키 삭제 예약 등을 포함해 이러한 KMS 키에 대한 완전한 제어 권한을 가집니다.

AWS 소유 키

관리형 통합은 기본적으로 이러한 키를 사용하여 민감한 고객 데이터를 자동으로 암호화합니다. 사용을 확인, 관리 또는 감사할 수 없습니다. 데이터를 암호화하는 키를 보호하기 위해 조치를 취하거나 프로그램을 변경할 필요가 없습니다. 저장 데이터를 기본적으로 암호화하면 민감한 데이터 보호와 관련된 운영 오버헤드와 복잡성을 줄이는 데 도움이 됩니다. 동시에 엄격한 암호화 규정 준수 및 규제 요구 사항을 충족하는 안전한 애플리케이션을 구축할 수 있습니다.

사용되는 기본 암호화 키는 AWS 소유 키입니다. 또는 암호화 키를 업데이트하는 선택적 API는 입니다`PutDefaultEncryptionConfiguration`.

AWS KMS 암호화 키 유형에 대한 자세한 내용은 AWS KMS 키를 참조하세요.

AWS KMS 관리형 통합에 대한 사용

관리형 통합은 봉투 암호화를 사용하여 모든 고객 데이터를 암호화하고 해독합니다. 이러한 유형의 암호화는 일반 텍스트 데이터를 가져와서 데이터 키로 암호화합니다. 다음으로 래핑 키라는 암호화 키는 일반 텍스트 데이터를 암호화하는 데 사용되는 원본 데이터 키를 암호화합니다. 봉투 암호화에서는 추가 래핑 키를 사용하여 원본 데이터 키와 분리 정도가 더 가까운 기존 래핑 키를 암호화할 수 있습니다. 원본 데이터 키는 별도로 저장된 래핑 키로 암호화되므로 원본 데이터 키와 암호화된 일반 텍스트 데이터를 동일한 위치에 저장할 수 있습니다. 키링은 데이터 키를 암호화 및 해독하는 데 사용되는 래핑 키 외에도 데이터 키를 생성, 암호화 및 해독하는 데 사용됩니다.

> **Note**
>
> AWS Database Encryption SDK는 클라이언트 측 암호화 구현을 위한 봉투 암호화를 제공합니다. AWS Database Encryption SDK에 대한 자세한 내용은 AWS Database Encryption SDK란 무엇입니까?를 참조하세요.

봉투 암호화, 데이터 키, 래핑 키 및 키링에 대한 자세한 내용은 봉투 암호화, 데이터 키, 래핑 키 및 키링을 참조하세요.

관리형 통합을 사용하려면 서비스가 다음 내부 작업에 고객 관리형 키를 사용해야 합니다.

- AWS KMS 에 DescribeKey 요청을 보내 데이터 키를 교체할 때 대칭 고객 관리형 키 ID가 제공되었는지 확인합니다.
- AWS KMS 에 GenerateDataKeyWithoutPlaintext 요청을 보내 고객 관리형 키로 암호화된 데이터 키를 생성합니다.
- AWS KMS 에 ReEncrypt* 요청을 보내 고객 관리형 키로 데이터 키를 다시 암호화합니다.
- AWS KMS 에 Decrypt 요청을 보내 고객 관리형 키로 데이터를 복호화합니다.

암호화 키를 사용하여 암호화된 데이터 유형

관리형 통합은 암호화 키를 사용하여 저장 시 저장된 여러 유형의 데이터를 암호화합니다. 다음 목록은 암호화 키를 사용하여 저장 시 암호화된 데이터 유형을 간략하게 설명합니다.

- 디바이스 검색 및 디바이스 상태 업데이트와 같은 클라우드 간(C2C) 커넥터 이벤트입니다.
- 물리적 디바이스와 특정 디바이스 유형에 대한 기능이 포함된 디바이스 프로파일을 나타내는 관리형 사물 생성. 디바이스 및 디바이스 프로파일에 대한 자세한 내용은 장치 및 섹션을 참조하세요장치.
- 디바이스 구현의 다양한 측면에 대한 관리형 통합 알림입니다. 관리형 통합 알림에 대한 자세한 내용은 섹션을 참조하세요관리형 통합 알림 설정.
- 디바이스 인증 자료, 디바이스 일련 번호, 최종 사용자 이름, 디바이스 식별자, 디바이스 Amazon 리소스 이름(arn)과 같은 최종 사용자의 개인 식별 정보(PII).

관리형 통합이에서 키 정책을 사용하는 방법 AWS KMS

브랜치 키 교체 및 비동기 호출의 경우 관리형 통합에는 암호화 키를 사용하기 위한 키 정책이 필요합니다. 키 정책은 다음과 같은 이유로 사용됩니다.

- 다른 보안 주체에게 암호화 키 AWS 사용을 프로그래밍 방식으로 승인합니다.

관리형 통합에서 암호화 키에 대한 액세스를 관리하는 데 사용되는 키 정책의 예는 섹션을 참조하세요. 암호화 키 생성

> **Note**
>
> AWS 소유 키의 경우 AWS 소유 키는에서 소유 AWS 하며 사용자는 해당 키를 보거나 관리하거나 사용할 수 없으므로 키 정책은 필요하지 않습니다. 관리형 통합은 기본적으로 AWS 소유 키를 사용하여 민감한 고객 데이터를 자동으로 암호화합니다.

키로 암호화 구성을 AWS KMS 관리하기 위해 키 정책을 사용하는 것 외에도 관리형 통합은 IAM 정책을 사용합니다. IAM 정책에 대한 자세한 내용은의 정책 및 권한을 참조하세요 AWS Identity and Access Management.

암호화 키 생성

AWS Management Console 또는 AWS KMS APIs.

암호화 키를 생성하려면

AWS Key Management Service 개발자 안내서의 KMS 키 생성 단계를 따릅니다.

키 정책

키 정책 문은 AWS KMS 키에 대한 액세스를 제어합니다. 각 AWS KMS 키에는 하나의 키 정책만 포함됩니다. 해당 키 정책에 따라 키를 사용할 수 있는 AWS 보안 주체와 키 사용 방법이 결정됩니다. 키 정책 설명을 사용하여 AWS KMS 키의 액세스 및 사용을 관리하는 방법에 대한 자세한 내용은 정책을 사용하여 액세스 관리를 참조하세요.

다음은 관리형 통합을 위해에 저장된 키에 AWS 계정 대한 액세스 및 사용을 관리하는 데 사용할 수 있는 AWS KMS 키 정책 설명의 예입니다.

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

키 스토어에 대한 자세한 내용은 키 스토어를 참조하세요.

암호화 구성 업데이트

암호화 구성을 원활하게 업데이트하는 기능은 관리형 통합을 위한 데이터 암호화 구현을 관리하는 데 매우 중요합니다. 관리형 통합으로 처음 온보딩하면 암호화 구성을 선택하라는 메시지가 표시됩니다. 옵션은 기본 AWS 소유 키이거나 자체 AWS KMS 키를 생성합니다.

AWS Management Console

에서 암호화 구성을 업데이트하려면 AWS IoT 서비스 홈페이지를 AWS Management Console연 다음 통합 제어를 위한 관리형 통합>설정>암호화로 이동합니다. 암호화 설정 창에서 추가 암호화 보호를 위해 새 AWS KMS 키를 선택하여 암호화 구성을 업데이트할 수 있습니다. 암호화 설정 사용자 지정(고급)을 선택하여 기존 AWS KMS 키를 선택하거나 AWS KMS 키 생성을 선택하여 자체 고객 관리형 키를 생성할 수 있습니다.

API 명령

관리형 통합에서 AWS KMS 키의 암호화 구성을 관리하는 데 사용되는 두 APIs는 `PutDefaultEncryptionConfiuration` 및 입니다`GetDefaultEncryptionConfiguration`.

기본 암호화 구성을 업데이트하려면를 호출합니다`PutDefaultEncryptionConfiuration`. 에 대한 자세한 내용은 PutDefaultEncryptionConfiuration을 `PutDefaultEncryptionConfiuration`참조하세요.

기본 암호화 구성을 보려면를 호출합니다`GetDefaultEncryptionConfiguration`. 에 대한 자세한 내용은 GetDefaultEncryptionConfiguration을 `GetDefaultEncryptionConfiguration`참조하세요.

관리형 통합을 위한 ID 및 액세스 관리

AWS Identity and Access Management (IAM)는 관리자가 AWS 리소스에 대한 액세스를 안전하게 제어하는 데 도움이 AWS 서비스 되는 입니다. IAM 관리자는 관리형 통합 리소스를 사용할 수 있는 인증(로그인) 및 권한(권한 있음)을 받을 수 있는 사용자를 제어합니다. IAM은 추가 비용 없이 사용할 수 AWS 서비스 있는 입니다.

주제

- 대상
- ID를 통한 인증
- 정책을 사용하여 액세스 관리
- AWS 관리형 통합을 위한 관리형 정책

- 관리형 통합이 IAM과 작동하는 방식
- 관리형 통합을 위한 자격 증명 기반 정책 예제
- 관리형 통합 ID 및 액세스 문제 해결
- 관리형 통합에 서비스 연결 역할 사용

대상

AWS Identity and Access Management (IAM)를 사용하는 방법은 역할에 따라 다릅니다.

- 서비스 사용자 - 기능에 액세스할 수 없는 경우 관리자에게 권한 요청(참조관리형 통합 ID 및 액세스 문제 해결)
- 서비스 관리자 - 사용자 액세스 결정 및 권한 요청 제출(관리형 통합이 IAM과 작동하는 방식 참조)
- IAM 관리자 - 액세스를 관리하기 위한 정책 작성(관리형 통합을 위한 자격 증명 기반 정책 예제 참조)

ID를 통한 인증

인증은 자격 증명 자격 증명을 AWS 사용하여에 로그인하는 방법입니다. AWS 계정 루트 사용자, IAM 사용자 또는 IAM 역할을 수임하여 인증되어야 합니다.

AWS IAM Identity Center (IAM Identity Center), Single Sign-On 인증 또는 Google/Facebook 자격 증명과 같은 자격 증명 소스의 자격 증명을 사용하여 페더레이션 자격 증명으로 로그인할 수 있습니다. 로그인하는 방법에 대한 자세한 내용은 AWS 로그인 사용 설명서의 AWS 계정에 로그인하는 방법을 참조하세요.

프로그래밍 방식 액세스를 위해는 요청에 암호화 방식으로 서명할 수 있는 SDK 및 CLI를 AWS 제공합니다. 자세한 내용은 IAM 사용 설명서의 API 요청용AWS Signature Version 4 섹션을 참조하세요.

AWS 계정 루트 사용자

를 생성할 때 모든 AWS 서비스 및 리소스에 대한 완전한 액세스 권한이 있는 AWS 계정 theroot 사용자라는 하나의 로그인 자격 증명으로 AWS 계정시작합니다. 일상적인 작업에 루트 사용자를 사용하지 않을 것을 강력히 권장합니다. 루트 사용자 자격 증명이 필요한 태스크는 IAM 사용 설명서의 루트 사용자 자격 증명이 필요한 태스크 섹션을 참조하세요.

페더레이션 자격 증명

가장 좋은 방법은 인간 사용자에게 자격 증명 공급자와의 페더레이션을 사용하여 임시 자격 증명을 AWS 서비스 사용하여에 액세스하도록 요구하는 것입니다.

페더레이션 자격 증명은 엔터프라이즈 디렉터리, 웹 자격 증명 공급자 또는 자격 증명 소스의 자격 증명을 AWS 서비스 사용하여 Directory Service 에 액세스하는 사용자입니다. 페더레이션 ID는 임시 ID를 제공하는 역할을 수임합니다.

중앙 집중식 액세스 관리를 위해 AWS IAM Identity Center를 추천합니다. 자세한 정보는 AWS IAM Identity Center 사용 설명서의 IAM Identity Center란 무엇입니까? 섹션을 참조하세요.

IAM 사용자 및 그룹

IAM 사용자는 단일 개인 또는 애플리케이션에 대한 특정 권한을 가진 ID입니다. 장기 자격 증명이 있는 IAM 사용자 대신 임시 자격 증명을 사용하는 것이 좋습니다. 자세한 내용은 IAM 사용 설명서의 자격 증명 공급자와의 페더레이션을 사용하여 IAM 사용 설명서의 임시 자격 증명을 AWS 사용하여에 액세스하도록 인간 사용자에게 요구를 참조하세요.

IAM 그룹은 IAM 사용자 모음을 지정하고 대규모 사용자 집합에 대한 관리 권한을 더 쉽게 만듭니다. 자세한 내용은 IAM 사용 설명서의 IAM 사용자 사용 사례 섹션을 참조하세요.

IAM 역할

IAM 역할은 임시 ID를 제공하는 특정 권한이 있는 ID입니다. 사용자에서 IAM 역할(콘솔)로 전환하거나 또는 API 작업을 호출하여 역할을 수임할 수 있습니다. AWS CLI AWS 자세한 내용은 IAM 사용자 설명서의 역할 수임 방법을 참조하세요.

IAM 역할은 페더레이션 사용자 액세스, 임시 IAM 사용자 권한, 교차 계정 액세스, 교차 서비스 액세스 및 Amazon EC2에서 실행되는 애플리케이션에 유용합니다. 자세한 내용은 IAM 사용 설명서의 IAM에서 교차 계정 리소스 액세스를 참조하세요.

정책을 사용하여 액세스 관리

정책을 AWS 생성하고 자격 증명 또는 리소스에 연결하여 AWS 에서 액세스를 제어합니다. 정책은 자격 증명 또는 리소스와 연결될 때 권한을 정의합니다.는 보안 주체가 요청할 때 이러한 정책을 AWS 평가합니다. 대부분의 정책은에 JSON 문서 AWS 로 저장됩니다. JSON 정책 문서에 대한 자세한 내용은 IAM 사용자 설명서의 JSON 정책 개요를 참조하세요.

정책을 사용하여 관리자는 어떤 위탁자가 어떤 리소스에 대해 어떤 조건에서 작업을 수행할 수 있는지 정의하여 누가 무엇에 액세스할 수 있는지 지정합니다.

기본적으로 사용자 및 역할에는 어떠한 권한도 없습니다. IAM 관리자는 IAM 정책을 생성하고 사용자가 수임할 수 있는 역할에 추가합니다. IAM 정책은 작업을 수행하기 위해 사용하는 방법과 관계없이 작업에 대한 권한을 정의합니다.

자격 증명 기반 정책

ID 기반 정책은 ID(사용자, 사용자 그룹 또는 역할)에 연결하는 JSON 권한 정책 문서입니다. 이러한 정책은 ID가 수행할 수 있는 작업, 대상 리소스 및 이에 관한 조건을 제어합니다. ID 기반 정책을 생성하는 방법을 알아보려면 IAM 사용 설명서에서 고객 관리형 정책으로 사용자 지정 IAM 권한 정의를 참조하세요.

ID 기반 정책은 인라인 정책(단일 ID에 직접 포함) 또는 관리형 정책(여러 ID에 연결된 독립 실행형 정책)일 수 있습니다. 관리형 정책과 인라인 정책 중에 선택하는 방법을 알아보려면 IAM 사용 설명서의 관리형 정책 및 인라인 정책 중에서 선택 섹션을 참조하세요.

리소스 기반 정책

리소스 기반 정책은 리소스에 연결하는 JSON 정책 설명서입니다. 예를 들어 IAM 역할 신뢰 정책 및 Amazon S3 버킷 정책이 있습니다. 리소스 기반 정책을 지원하는 서비스에서 서비스 관리자는 이러한 정책을 사용하여 특정 리소스에 대한 액세스를 통제할 수 있습니다. 리소스 기반 정책에서 엔터티를 지정해야 합니다.

리소스 기반 정책은 해당 서비스에 있는 인라인 정책입니다. 리소스 기반 정책에서는 IAM의 AWS 관리형 정책을 사용할 수 없습니다.

기타 정책 유형

AWS 는 보다 일반적인 정책 유형에서 부여한 최대 권한을 설정할 수 있는 추가 정책 유형을 지원합니다.

- 권한 경계 - ID 기반 정책에서 IAM 엔터티에 부여할 수 있는 최대 권한을 설정합니다. 자세한 정보는 IAM 사용자 설명서의 IAM 엔터티의 권한 경계를 참조하세요.
- 서비스 제어 정책(SCP) - AWS Organizations내 조직 또는 조직 단위에 대한 최대 권한을 지정합니다. 자세한 내용은 AWS Organizations 사용 설명서의 서비스 제어 정책을 참조하세요.
- 리소스 제어 정책(RCP) - 계정 내 리소스에 대해 사용할 수 있는 최대 권한을 설정합니다. 자세한 내용은 AWS Organizations 사용 설명서의 리소스 제어 정책(RCP)을 참조하세요.
- 세션 정책 – 역할 또는 페더레이션 사용자에 대해 임시 세션을 프로그래밍 방식으로 생성할 때 파라미터로 전달하는 고급 정책입니다. 자세한 내용은 IAM 사용자 설명서의 세션 정책을 참조하세요.

여러 정책 유형

여러 정책 유형이 요청에 적용되는 경우, 결과 권한은 이해하기가 더 복잡합니다. 에서 여러 정책 유형이 관련될 때 요청을 허용할지 여부를 AWS 결정하는 방법을 알아보려면 IAM 사용 설명서의 정책 평가 로직을 참조하세요.

AWS 관리형 통합을 위한 관리형 정책

사용자, 그룹 및 역할에 권한을 추가하려면 직접 정책을 작성하는 것보다 AWS 관리형 정책을 사용하는 것이 더 쉽습니다. 팀에 필요한 권한만 제공하는 IAM 고객 관리형 정책을 생성하기 위해서는 시간과 전문 지식이 필요합니다. 빠르게 시작하려면 AWS 관리형 정책을 사용할 수 있습니다. 이 정책은 일반적인 사용 사례를 다루며 사용자의 AWS 계정에서 사용할 수 있습니다. AWS 관리형 정책에 대한 자세한 내용은 IAM 사용 설명서의 AWS 관리형 정책을 참조하세요.

AWS 서비스는 AWS 관리형 정책을 유지 관리하고 업데이트합니다. AWS 관리형 정책에서는 권한을 변경할 수 없습니다. 서비스에서 때때로 추가 권한을 AWS 관리형 정책에 추가하여 새로운 기능을 지원합니다. 이 유형의 업데이트는 정책이 연결된 모든 ID(사용자, 그룹 및 역할)에 적용됩니다. 서비스는 새로운 기능이 시작되거나 새 작업을 사용할 수 있을 때 AWS 관리형 정책에 업데이트됩니다. 서비스는 AWS 관리형 정책에서 권한을 제거하지 않으므로 정책 업데이트로 인해 기존 권한이 손상되지 않습니다.

또한는 여러 서비스에 걸쳐 있는 직무에 대한 관리형 정책을 AWS 지원합니다. 예를 들어 ReadOnlyAccess AWS 관리형 정책은 모든 AWS 서비스 및 리소스에 대한 읽기 전용 액세스를 제공합니다. 서비스가 새 기능을 시작하면는 새 작업 및 리소스에 대한 읽기 전용 권한을 AWS 추가합니다. 직무 정책의 목록과 설명은 IAM 사용 설명서의 직무에 관한AWS 관리형 정책을 참조하세요.

AWS 관리형 정책: AWSIoTManagedIntegrationsFullAccess

`AWSIoTManagedIntegrationsFullAccess` 정책을 IAM ID에 연결할 수 있습니다.

이 정책은 관리형 통합 및 관련 서비스에 대한 전체 액세스 권한을 부여합니다. 에서이 정책을 보려면 AWSIoTManagedIntegrationsFullAccess를 AWS Management Console참조하세요.

권한 세부 정보

이 정책에는 다음 권한이 포함되어 있습니다.

- `iotmanagedintegrations` -이 정책을 추가하는 IAM 사용자, 그룹 및 역할에 대한 관리형 통합 및 관련 서비스에 대한 전체 액세스 권한을 제공합니다.
- `iam` - 할당된 IAM 사용자, 그룹 및 역할이에서 서비스 연결 역할을 생성할 수 있도록 허용합니다 AWS 계정.

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 관리형 정책: AWS IoTManagedIntegrationsRolePolicy

`AWS IoTManagedIntegrationsRolePolicy` 정책을 IAM ID에 연결할 수 있습니다.

이 정책은 사용자를 대신하여 Amazon CloudWatch logs 및 지표를 게시할 수 있는 관리형 통합 권한을 부여합니다.

에서이 정책을 보려면 AWSIoTManagedIntegrationsRolePolicy를 AWS Management Console참조하세요.

권한 세부 정보

이 정책에는 다음 권한이 포함되어 있습니다.

- `logs` - Amazon CloudWatch 로그 그룹을 생성하고 로그를 그룹으로 스트리밍하는 기능을 제공합니다.
- `cloudwatch` - Amazon CloudWatch 지표를 게시하는 기능을 제공합니다. Amazon CloudWatch 지표에 대한 자세한 내용은 Amazon CloudWatch의 지표를 참조하세요.

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"
          ]
        }
      }
    }
  ]
}
```

관리형 정책에 대한 AWS 관리형 통합 업데이트

이 서비스가 이러한 변경 사항을 추적하기 시작한 이후 AWS 부터 관리형 통합을 위한 관리형 정책 업데이트에 대한 세부 정보를 봅니다. 이 페이지의 변경 사항에 대한 자동 알림을 받으려면 관리형 통합 문서 기록 페이지에서 RSS 피드를 구독하세요.

변경	설명	Date
관리형 통합에서 변경 사항 추적 시작	관리형 통합은 AWS 관리형 정책에 대한 변경 사항 추적을 시작했습니다.	2025년 3월 3일

관리형 통합이 IAM과 작동하는 방식

IAM을 사용하여 관리형 통합에 대한 액세스를 관리하기 전에 관리형 통합에 사용할 수 있는 IAM 기능을 알아봅니다.

관리형 통합에 사용할 수 있는 IAM 기능

IAM 특성	관리형 통합 지원
자격 증명 기반 정책	예
리소스 기반 정책	아니요
정책 작업	예
정책 리소스	예
정책 조건 키	예
ACL	아니요
ABAC(정책 내 태그)	아니요
임시 보안 인증	예
위탁자 권한	예
서비스 역할	예
서비스 연결 역할	예

관리형 통합 및 기타 AWS 서비스가 대부분의 IAM 기능과 작동하는 방식을 개괄적으로 알아보려면 IAM 사용 설명서의 AWS IAM으로 작업하는 서비스를 참조하세요.

관리형 통합을 위한 자격 증명 기반 정책

자격 증명 기반 정책 지원: 예

자격 증명 기반 정책은 IAM 사용자, 사용자 그룹 또는 역할과 같은 ID에 연결할 수 있는 JSON 권한 정책 문서입니다. 이러한 정책은 사용자 및 역할이 어떤 리소스와 어떤 조건에서 어떤 작업을 수행할 수 있는지를 제어합니다. 자격 증명 기반 정책을 생성하는 방법을 알아보려면 IAM 사용 설명서에서 고객 관리형 정책으로 사용자 지정 IAM 권한 정의를 참조하세요.

IAM 자격 증명 기반 정책을 사용하면 허용되거나 거부되는 작업과 리소스뿐 아니라 작업이 허용되거나 거부되는 조건을 지정할 수 있습니다. JSON 정책에서 사용할 수 있는 모든 요소에 대해 알아보려면 IAM 사용 설명서의 IAM JSON 정책 요소 참조를 참조하세요.

관리형 통합을 위한 자격 증명 기반 정책 예제

관리형 통합 자격 증명 기반 정책의 예를 보려면 섹션을 참조하세요관리형 통합을 위한 자격 증명 기반 정책 예제.

관리형 통합 내의 리소스 기반 정책

리소스 기반 정책 지원: 아니요

리소스 기반 정책은 리소스에 연결하는 JSON 정책 설명서입니다. 리소스 기반 정책의 예제는 IAM 역할 신뢰 정책과 Amazon S3 버킷 정책입니다. 리소스 기반 정책을 지원하는 서비스에서 서비스 관리자는 이러한 정책을 사용하여 특정 리소스에 대한 액세스를 통제할 수 있습니다. 정책이 연결된 리소스의 경우 정책은 지정된 위탁자가 해당 리소스와 어떤 조건에서 어떤 작업을 수행할 수 있는지를 정의합니다. 리소스 기반 정책에서 위탁자를 지정해야 합니다. 보안 주체에는 계정, 사용자, 역할, 페더레이션 사용자 또는이 포함될 수 있습니다 AWS 서비스.

교차 계정 액세스를 활성화하려는 경우 전체 계정이나 다른 계정의 IAM 개체를 리소스 기반 정책의 보안 주체로 지정할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 IAM에서 교차 계정 리소스 액세스를 참조하세요.

관리형 통합에 대한 정책 작업

정책 작업 지원: 예

관리자는 AWS JSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지 지정할 수 있습니다. 즉, 어떤 엔터티가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

JSON 정책의 `Action`요소는 정책에서 액세스를 허용하거나 거부하는 데 사용할 수 있는 작업을 설명합니다. 연결된 작업을 수행할 수 있는 권한을 부여하기 위한 정책에 작업을 포함하세요.

관리형 통합 작업 목록을 보려면 서비스 승인 참조의 관리형 통합에서 정의한 작업을 참조하세요.

관리형 통합의 정책 작업은 작업 앞에 다음 접두사를 사용합니다.

```
iot-mi
```

단일 문에서 여러 작업을 지정하려면 쉼표로 구분합니다.

```
"Action": [
```

```
      "iot-mi:action1",
      "iot-mi:action2"
         ]
```

관리형 통합 자격 증명 기반 정책의 예를 보려면 섹션을 참조하세요관리형 통합을 위한 자격 증명 기반 정책 예제.

관리형 통합을 위한 정책 리소스

정책 리소스 지원: 예

관리자는 AWS JSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지 지정할 수 있습니다. 즉, 어떤 위탁자가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

`Resource` JSON 정책 요소는 작업이 적용되는 하나 이상의 객체를 지정합니다. 모범 사례에 따라 Amazon 리소스 이름(ARN)을 사용하여 리소스를 지정합니다. 리소스 수준 권한을 지원하지 않는 작업의 경우, 와일드카드(*)를 사용하여 해당 문이 모든 리소스에 적용됨을 나타냅니다.

```
"Resource": "*"
```

관리형 통합 리소스 유형 및 해당 ARNs 목록을 보려면 서비스 승인 참조의 관리형 통합에서 정의한 리소스를 참조하세요. 각 리소스의 ARN을 지정할 수 있는 작업을 알아보려면 관리형 통합에서 정의한 작업을 참조하세요.

관리형 통합 자격 증명 기반 정책의 예를 보려면 섹션을 참조하세요관리형 통합을 위한 자격 증명 기반 정책 예제.

관리형 통합을 위한 정책 조건 키

서비스별 정책 조건 키 지원: 예

관리자는 AWS JSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지 지정할 수 있습니다. 즉, 어떤 엔터티가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

`Condition` 요소는 정의된 기준에 따라 문이 실행되는 시기를 지정합니다. 같음(equals) 또는 미만(less than)과 같은 조건 연산자를 사용하여 정책의 조건을 요청의 값과 일치시키는 조건식을 생성할 수 있습니다. 모든 AWS 전역 조건 키를 보려면 IAM 사용 설명서의 AWS 전역 조건 컨텍스트 키를 참조하세요.

관리형 통합 조건 키 목록을 보려면 서비스 승인 참조의 관리형 통합에 대한 조건 키를 참조하세요. 조건 키를 사용할 수 있는 작업과 리소스를 알아보려면 관리형 통합에서 정의한 작업을 참조하세요.

관리형 통합 자격 증명 기반 정책의 예를 보려면 섹션을 참조하세요관리형 통합을 위한 자격 증명 기반 정책 예제.

관리형 통합의 ACLs

ACL 지원: 아니요

액세스 제어 목록(ACL)은 어떤 위탁자(계정 멤버, 사용자 또는 역할)가 리소스에 액세스할 수 있는 권한을 가지고 있는지를 제어합니다. ACL은 JSON 정책 문서 형식을 사용하지 않지만 리소스 기반 정책과 유사합니다.

관리형 통합이 포함된 ABAC

ABAC 지원(정책의 태그): 부분적

속성 기반 액세스 제어(ABAC)는 태그라고 불리는 속성을 기반으로 권한을 정의하는 권한 부여 전략입니다. IAM 엔터티 및 AWS 리소스에 태그를 연결한 다음 보안 주체의 태그가 리소스의 태그와 일치할 때 작업을 허용하는 ABAC 정책을 설계할 수 있습니다.

태그에 근거하여 액세스를 제어하려면 `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`또는 `aws:TagKeys`조건 키를 사용하여 정책의 조건 요소에 태그 정보를 제공합니다.

서비스가 모든 리소스 유형에 대해 세 가지 조건 키를 모두 지원하는 경우, 값은 서비스에 대해 예입니다. 서비스가 일부 리소스 유형에 대해서만 세 가지 조건 키를 모두 지원하는 경우, 값은 부분적입니다.

ABAC에 관한 자세한 내용은 IAM 사용 설명서의 ABAC 권한 부여를 통한 권한 정의를 참조하세요. ABAC 설정 단계가 포함된 자습서를 보려면 IAM 사용자 설명서의 속성 기반 액세스 제어(ABAC) 사용을 참조하세요.

관리형 통합에서 임시 자격 증명 사용

임시 자격 증명 지원: 예

임시 자격 증명은 AWS 리소스에 대한 단기 액세스를 제공하며 페더레이션을 사용하거나 역할을 전환할 때 자동으로 생성됩니다. 장기 액세스 키를 사용하는 대신 임시 자격 증명을 동적으로 생성하는 것이 AWS 좋습니다. 자세한 내용은 IAM 사용 설명서의 IAM의 임시 보안 자격 증명 및 IAM으로 작업하는AWS 서비스 섹션을 참조하세요.

관리형 통합에 대한 교차 서비스 보안 주체 권한

전달 액세스 세션(FAS) 지원: 예

전달 액세스 세션(FAS)은를 호출하는 보안 주체의 권한을 다운스트림 서비스에 AWS 서비스 대한 요청과 AWS 서비스함께 사용합니다. FAS 요청 시 정책 세부 정보는 전달 액세스 세션을 참조하세요.

관리형 통합을 위한 서비스 역할

서비스 역할 지원: 예

서비스 역할은 서비스가 사용자를 대신하여 작업을 수행하는 것으로 가정하는 IAM 역할입니다. IAM 관리자는 IAM 내에서 서비스 역할을 생성, 수정 및 삭제할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 AWS 서비스 AWS에 권한을 위임할 역할 생성을 참조하세요.

> **Warning**
>
> 서비스 역할에 대한 권한을 변경하면 관리형 통합 기능이 중단될 수 있습니다. 관리형 통합에서 관련 지침을 제공하는 경우에만 서비스 역할을 편집합니다.

관리형 통합을 위한 서비스 연결 역할

서비스 연결 역할 지원: 예

서비스 연결 역할은에 연결된 서비스 역할의 한 유형입니다 AWS 서비스. 서비스는 사용자를 대신하여 작업을 수행하기 위해 역할을 수임할 수 있습니다. 서비스 연결 역할은에 나타나 AWS 계정 며 서비스가 소유합니다. IAM 관리자는 서비스 연결 역할의 권한을 볼 수 있지만 편집은 할 수 없습니다.

서비스 연결 역할 생성 또는 관리에 관한 자세한 내용은 IAM으로 작업하는AWS 서비스를 참조하세요. 서비스 연결 역할 열에서 Yes가 포함된 서비스를 테이블에서 찾습니다. 해당 서비스에 대한 서비스 연결 역할 설명서를 보려면 예(Yes) 링크를 선택합니다.

관리형 통합을 위한 자격 증명 기반 정책 예제

기본적으로 사용자 및 역할에는 관리형 통합 리소스를 생성하거나 수정할 수 있는 권한이 없습니다. 사용자에게 사용자가 필요한 리소스에서 작업을 수행할 권한을 부여하려면 IAM 관리자가 IAM 정책을 생성하면 됩니다.

이러한 예제 JSON 정책 문서를 사용하여 IAM 자격 증명 기반 정책을 생성하는 방법을 알아보려면 IAM 사용 설명서의 IAM 정책 생성(콘솔)을 참조하세요.

각 리소스 유형에 대한 ARNs 형식을 포함하여 관리형 통합에서 정의한 작업 및 리소스 유형에 대한 자세한 내용은 서비스 승인 참조의 관리형 통합을 위한 작업, 리소스 및 조건 키를 참조하세요.

주제

- 정책 모범 사례
- 관리형 통합 콘솔 사용
- 사용자가 자신의 고유한 권한을 볼 수 있도록 허용

정책 모범 사례

자격 증명 기반 정책에 따라 계정에서 사용자가 관리형 통합 리소스를 생성, 액세스 또는 삭제할 수 있는지 여부가 결정됩니다. 이 작업으로 인해 AWS 계정에 비용이 발생할 수 있습니다. 자격 증명 기반 정책을 생성하거나 편집할 때는 다음 지침과 권장 사항을 따릅니다.

- AWS 관리형 정책을 시작하고 최소 권한으로 전환 - 사용자 및 워크로드에 권한 부여를 시작하려면 많은 일반적인 사용 사례에 대한 권한을 부여하는 AWS 관리형 정책을 사용합니다. 에서 사용할 수 있습니다 AWS 계정. 사용 사례에 맞는 AWS 고객 관리형 정책을 정의하여 권한을 추가로 줄이는 것이 좋습니다. 자세한 내용은 IAM 사용자 설명서의 AWS 관리형 정책 또는 AWS 직무에 대한 관리형 정책을 참조하세요.
- 최소 권한 적용 – IAM 정책을 사용하여 권한을 설정하는 경우, 작업을 수행하는 데 필요한 권한만 부여합니다. 이렇게 하려면 최소 권한으로 알려진 특정 조건에서 특정 리소스에 대해 수행할 수 있는 작업을 정의합니다. IAM을 사용하여 권한을 적용하는 방법에 대한 자세한 정보는 IAM 사용자 설명서에 있는 IAM의 정책 및 권한을 참조하세요.
- IAM 정책의 조건을 사용하여 액세스 추가 제한 – 정책에 조건을 추가하여 작업 및 리소스에 대한 액세스를 제한할 수 있습니다. 예를 들어, SSL을 사용하여 모든 요청을 전송해야 한다고 지정하는 정책 조건을 작성할 수 있습니다. AWS 서비스와 같은 특정를 통해 사용되는 경우 조건을 사용하여 서비스 작업에 대한 액세스 권한을 부여할 수도 있습니다 CloudFormation. 자세한 내용은 IAM 사용자 설명서의 IAM JSON 정책 요소: 조건을 참조하세요.
- IAM Access Analyzer를 통해 IAM 정책을 확인하여 안전하고 기능적인 권한 보장 - IAM Access Analyzer에서는 IAM 정책 언어(JSON)와 모범 사례가 정책에서 준수되도록 새로운 및 기존 정책을 확인합니다. IAM Access Analyzer는 100개 이상의 정책 확인 항목과 실행 가능한 추천을 제공하여 안전하고 기능적인 정책을 작성하도록 돕습니다. 자세한 내용은 IAM 사용자 설명서의 IAM Access Analyzer에서 정책 검증을 참조하세요.
- 다중 인증(MFA) 필요 -에서 IAM 사용자 또는 루트 사용자가 필요한 시나리오가 있는 경우 추가 보안을 위해 MFA를 AWS 계정켭니다. API 작업을 직접적으로 호출할 때 MFA가 필요하면 정책에 MFA

조건을 추가합니다. 자세한 내용은 IAM 사용자 설명서의 MFA를 통한 보안 API 액세스를 참조하세요.

IAM의 모범 사례에 대한 자세한 내용은 IAM 사용자 설명서의 IAM의 보안 모범 사례를 참조하세요.

관리형 통합 콘솔 사용

관리형 통합 콘솔에 액세스하려면 최소 권한 집합이 있어야 합니다. 이러한 권한은에서 관리형 통합 리소스에 대한 세부 정보를 나열하고 볼 수 있도록 허용해야 합니다 AWS 계정. 최소 필수 권한보다 더 제한적인 자격 증명 기반 정책을 생성하는 경우, 콘솔이 해당 정책에 연결된 엔티티(사용자 또는 역할)에 대해 의도대로 작동하지 않습니다.

AWS CLI 또는 AWS API만 호출하는 사용자에게는 최소 콘솔 권한을 허용할 필요가 없습니다. 대신, 수행하려는 API 작업과 일치하는 작업에만 액세스할 수 있도록 합니다.

사용자와 역할이 관리형 통합 콘솔을 계속 사용할 수 있도록 하려면 관리형 통합 *ConsoleAccess* 또는 *ReadOnly* AWS 관리형 정책도 엔터티에 연결합니다. 자세한 내용은 IAM 사용 설명서의 사용자에게 권한 추가를 참조하세요.

사용자가 자신의 고유한 권한을 볼 수 있도록 허용

이 예제는 IAM 사용자가 자신의 사용자 ID에 연결된 인라인 및 관리형 정책을 볼 수 있도록 허용하는 정책을 생성하는 방법을 보여줍니다. 이 정책에는 콘솔에서 또는 AWS CLI 또는 AWS API를 사용하여 프로그래밍 방식으로이 작업을 완료할 수 있는 권한이 포함됩니다.

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

관리형 통합 ID 및 액세스 문제 해결

다음 정보를 사용하여 관리형 통합 및 IAM 작업 시 발생할 수 있는 일반적인 문제를 진단하고 수정할 수 있습니다.

주제

- 관리형 통합에서 작업을 수행할 권한이 없음
- iam:PassRole을 수행하도록 인증되지 않음
- 내 외부의 사람이 내 관리형 통합 리소스에 액세스 AWS 계정 하도록 허용하려고 함

관리형 통합에서 작업을 수행할 권한이 없음

작업을 수행할 권한이 없다는 오류가 표시되면 작업을 수행할 수 있도록 정책을 업데이트해야 합니다.

다음의 예제 오류는 mateojackson IAM 사용자가 콘솔을 사용하여 가상 *my-example-widget* 리소스에 대한 세부 정보를 보려고 하지만 가상 iot-mi:*GetWidget* 권한이 없을 때 발생합니다.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform: iot-
mi:GetWidget on resource: my-example-widget
```

이 경우, iot-mi:*GetWidget* 작업을 사용하여 *my-example-widget* 리소스에 액세스할 수 있도록 mateojackson 사용자 정책을 업데이트해야 합니다.

도움이 필요한 경우 AWS 관리자에게 문의하세요. 관리자는 로그인 자격 증명을 제공한 사람입니다.

iam:PassRole을 수행하도록 인증되지 않음

`iam:PassRole` 작업을 수행할 권한이 없다는 오류가 수신되면 관리형 통합에 역할을 전달할 수 있도록 정책을 업데이트해야 합니다.

일부 AWS 서비스 에서는 새 서비스 역할 또는 서비스 연결 역할을 생성하는 대신 기존 역할을 해당 서비스에 전달할 수 있습니다. 이렇게 하려면 역할을 서비스에 전달할 권한이 있어야 합니다.

다음 예제 오류는 라는 IAM 사용자가 콘솔을 사용하여 관리형 통합에서 작업을 수행하려고 `marymajor` 할 때 발생합니다. 하지만 작업을 수행하려면 서비스 역할이 부여한 권한이 서비스에 있어야 합니다. Mary는 서비스에 역할을 전달할 권한이 없습니다.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

이 경우, Mary가 `iam:PassRole`작업을 수행할 수 있도록 Mary의 정책을 업데이트해야 합니다.

도움이 필요한 경우 AWS 관리자에게 문의하세요. 관리자는 로그인 자격 증명을 제공한 사람입니다.

내 외부의 사람이 내 관리형 통합 리소스에 액세스 AWS 계정 하도록 허용하려고 함

다른 계정의 사용자 또는 조직 외부의 사람이 리소스에 액세스할 때 사용할 수 있는 역할을 생성할 수 있습니다. 역할을 수임할 신뢰할 수 있는 사람을 지정할 수 있습니다. 리소스 기반 정책 또는 액세스 제어 목록(ACL)을 지원하는 서비스의 경우, 이러한 정책을 사용하여 다른 사람에게 리소스에 대한 액세스 권한을 부여할 수 있습니다.

자세한 내용은 다음을 참조하세요.

- 관리형 통합이 이러한 기능을 지원하는지 여부를 알아보려면 섹션을 참조하세요관리형 통합이 IAM 과 작동하는 방식.
- 소유 AWS 계정 한의 리소스에 대한 액세스 권한을 제공하는 방법을 알아보려면 IAM 사용 설명서의 소유한 다른의 IAM 사용자에게 액세스 권한 제공을 참조 AWS 계정 하세요.
- 리소스에 대한 액세스 권한을 타사에 제공하는 방법을 알아보려면 IAM 사용 설명서의 타사 AWS 계정 소유에 대한 액세스 권한 제공을 AWS 계정참조하세요.
- ID 페더레이션을 통해 액세스 권한을 제공하는 방법을 알아보려면 IAM 사용자 설명서의 외부에서 인증된 사용자에게 액세스 권한 제공(ID 페더레이션)을 참조하세요.

- 교차 계정 액세스에 대한 역할과 리소스 기반 정책 사용의 차이점을 알아보려면 IAM 사용 설명서의 IAM의 교차 계정 리소스 액세스를 참조하세요.

관리형 통합에 서비스 연결 역할 사용

AWS IoT 디바이스 관리를 위한 관리형 통합은 AWS Identity and Access Management (IAM) 서비스 연결 역할을 사용합니다. 서비스 연결 역할은 관리형 통합에 직접 연결된 고유한 유형의 IAM 역할입니다. 서비스 연결 역할은 관리형 통합에 의해 사전 정의되며 서비스가 사용자를 대신하여 다른 AWS 서비스를 호출하는 데 필요한 모든 권한을 포함합니다.

필요한 권한을 수동으로 추가할 필요가 없으므로 서비스 연결 역할을 사용하면 관리형 통합을 더 쉽게 설정할 수 있습니다. AWS IoT 디바이스 관리용 관리형 통합은 서비스 연결 역할의 권한을 정의하며, 달리 정의되지 않은 한 관리형 통합만 해당 역할을 수임할 수 있습니다. 정의된 권한에는 신뢰 정책과 권한 정책이 포함되며 이 권한 정책은 다른 IAM 엔터티에 연결할 수 없습니다.

먼저 관련 리소스를 삭제한 후에만 서비스 연결 역할을 삭제할 수 있습니다. 이렇게 하면 리소스에 대한 액세스 권한을 실수로 제거할 수 없기 때문에 관리형 통합 리소스가 보호됩니다.

서비스 연결 역할을 지원하는 다른 서비스에 대한 자세한 내용은 AWS IAM으로 작업하는 서비스를 참조하고 서비스 연결 역할 열에서 예인 서비스를 찾습니다. 해당 서비스에 대한 서비스 연결 역할 설명서를 보려면 예 링크를 선택합니다.

관리형 통합에 대한 서비스 연결 역할 권한

AWS IoT 디바이스 관리를 위한 관리형 통합은 AWSServiceRoleForIoTManagedIntegrations라는 서비스 연결 역할을 사용합니다. - 사용자를 대신하여 로그 및 지표를 게시할 수 있는 AWS IoT 디바이스 관리 권한을 위한 관리형 통합을 제공합니다.

AWSServiceRoleForIoTManagedIntegrations 서비스 연결 역할은 역할을 수임하기 위해 다음 서비스를 신뢰합니다.

- `iotmanagedintegrations.amazonaws.com`

AWSIoTManagedIntegrationsServiceRolePolicy라는 역할 권한 정책을 사용하면 관리형 통합이 지정된 리소스에서 다음 작업을 완료할 수 있습니다.

- 작업: `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"
          ]
        }
      }
```

```
    }
  ]
}
```

사용자, 그룹 또는 역할이 서비스 연결 역할을 생성, 편집 또는 삭제할 수 있도록 사용 권한을 구성해야 합니다. 자세한 내용은 IAM 사용 설명서의 서비스 연결 역할 권한을 참조하세요.

관리형 통합을 위한 서비스 연결 역할 생성

서비스 연결 역할은 수동으로 생성할 필요가 없습니다. `PutRuntimeLogConfiguration`, 또는 `RegisterCustomEndpoint` API에서 `CreateEventLogConfiguration`, 또는 API 명령을 호출하는 등의 이벤트 유형이 발생하면 AWS CLI AWS 관리형 통합 AWS Management Console이 서비스 연결 역할을 생성합니다. `PutRuntimeLogConfiguration`, `CreateEventLogConfiguration`또는에 대한 자세한 내용은 PutRuntimeLogConfiguration, CreateEventLogConfiguration 또는 RegisterCustomEndpoint를 `RegisterCustomEndpoint`참조하세요.

이 서비스 연결 역할을 삭제했다가 다시 생성해야 하는 경우 동일한 프로세스를 사용하여 계정에서 역할을 다시 생성할 수 있습니다. `PutRuntimeLogConfiguration`, `CreateEventLogConfiguration`또는 `RegisterCustomEndpoint` API 명령 호출과 같은 이벤트 유형을 유발하면 관리형 통합이 서비스 연결 역할을 다시 생성합니다. 또는를 통해 AWS 고객 지원에 문의할 수 있습니다 AWS Support Center Console. AWS 지원 플랜에 대한 자세한 내용은 AWS 지원 플랜 비교를 참조하세요.

IAM 콘솔을 사용하여 IoT ManagedIntegrations - 관리형 역할 사용 사례로 서비스 연결 역할을 생성할 수도 있습니다. AWS CLI 또는 AWS API에서 서비스 이름을 사용하여 `iotmanagedintegrations.amazonaws.com` 서비스 연결 역할을 생성합니다. 자세한 내용은 IAM 사용자 설명서의 서비스 연결 역할 생성 섹션을 참조하세요. 이 서비스 연결 역할을 삭제하면 동일한 프로세스를 사용하여 역할을 다시 생성할 수 있습니다.

관리형 통합을 위한 서비스 연결 역할 편집

관리형 통합에서는 AWSServiceRoleForIoTManagedIntegrations 서비스 연결 역할을 편집할 수 없습니다. 서비스 연결 역할을 생성한 후에는 다양한 개체가 역할을 참조할 수 있기 때문에 역할 이름을 변경할 수 없습니다. 하지만 IAM을 사용하여 역할의 설명을 편집할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 서비스 연결 역할 편집을 참조하세요.

관리형 통합을 위한 서비스 연결 역할 삭제

서비스 연결 역할이 필요한 기능 또는 서비스가 더 이상 필요 없는 경우에는 해당 역할을 삭제하는 것이 좋습니다. 따라서 적극적으로 모니터링하거나 유지하지 않는 미사용 엔터티가 없도록 합니다. 단, 서비스 연결 역할에 대한 리소스를 먼저 정리해야 수동으로 삭제할 수 있습니다.

> **Note**
>
> 리소스를 삭제하려고 할 때 관리형 통합에서 역할을 사용하는 경우 삭제에 실패할 수 있습니다. 이 문제가 발생하면 몇 분 기다렸다가 작업을 다시 시도하세요.

IAM을 사용하여 수동으로 서비스 연결 역할을 삭제하려면 다음을 수행하세요.

IAM 콘솔 AWS CLI, 또는 AWS API를 사용하여 AWSServiceRoleForIoTManagedIntegrations 서비스 연결 역할을 삭제합니다. 자세한 내용은 IAM 사용 설명서의 서비스 연결 역할 삭제를 참조하십시오.

관리형 통합 서비스 연결 역할에 지원되는 리전

AWS IoT 디바이스 관리를 위한 관리형 통합은 서비스를 사용할 수 있는 모든 리전에서 서비스 연결 역할 사용을 지원합니다. 자세한 내용은 AWS 리전 및 엔드포인트 단원을 참조하세요.

C2C 워크플로 AWS Secrets Manager 의 데이터 보호에 사용

AWS Secrets Manager 는 데이터베이스 자격 증명, API 키 및 기타 보안 암호 정보를 보호하는 데 사용할 수 있는 보안 암호 스토리지 서비스입니다. 그런 다음, 코드에서 하드코딩된 자격 증명를 Secrets Manager에 대한 API 직접 호출로 교체할 수 있습니다. 그러면 보안 암호가 해당 위치에 있지 않기 때문에 여러분의 코드를 검사하는 누군가에 의해 보안 암호가 손상되지 않도록 방지할 수 있습니다. 개요는 AWS Secrets Manager 사용 설명서를 참조하십시오.

Secrets Manager는 AWS Key Management Service 키를 사용하여 보안 암호를 암호화합니다. 자세한 내용은 AWS Key Management Service에서 시크릿 암호화 및 복호화를 참조하세요.

용 관리형 통합은와 AWS IoT Device Management 통합되어 Secrets Manager에 데이터를 저장하고 구성에서 보안 암호 ID를 사용할 AWS Secrets Manager 수 있습니다.

관리형 통합에서 보안 암호를 사용하는 방법

공개 권한 부여(OAuth)는 위임된 액세스 권한 부여의 공개 표준으로, 사용자가 암호를 공유하지 않고도 웹 사이트 또는 애플리케이션에 다른 웹 사이트의 정보에 대한 액세스 권한을 부여할 수 있습니다.

이는 타사 애플리케이션이 사용자를 대신하여 사용자 데이터에 액세스할 수 있는 안전한 방법으로 암호 공유에 대한 보다 안전한 대안을 제공합니다.

OAuth에서 클라이언트 ID와 클라이언트 보안 암호는 액세스 토큰을 요청할 때 클라이언트 애플리케이션을 식별하고 인증하는 자격 증명입니다.

용 관리형 통합 AWS IoT Device Management 은 OAuth를 사용하여 C2C 워크플로를 사용하는 고객과 통신합니다. 고객은 통신을 위해 클라이언트 ID와 클라이언트 암호를 제공해야 합니다. 관리형 통합 고객은 AWS 계정에 클라이언트 ID와 클라이언트 보안 암호를 저장하고 관리형 통합은 고객 계정에서 클라이언트 ID와 클라이언트 보안 암호를 읽습니다.

보안 암호 생성 방법

보안 암호를 생성하려면 AWS Secrets Manager 사용 설명서의 AWS Secrets Manager 보안 암호 생성 단계를 따르세요.

관리형 통합이 보안 암호 값을 읽으려면 고객 관리형 AWS KMS 키로 보안 암호를 생성해야 합니다. 자세한 내용은 AWS Secrets Manager 사용 설명서의 AWS KMS 키에 대한 권한을 참조하세요.

다음 섹션의 IAM 정책도 사용해야 합니다.

가 보안 암호를 검색할 AWS IoT Device Management 수 있도록 관리형 통합에 대한 액세스 권한 부여

관리형 통합이 Secrets Manager에서 보안 암호 값을 검색할 수 있도록 하려면 보안 암호를 생성할 때 해당 보안 암호에 대한 리소스 정책에 다음 권한을 포함합니다.

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"

        }
      }
  } ]
}
```

고객 관리형 AWS KMS 키의 정책에 다음 문을 추가합니다.

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/*"
            ]
        }

    ]
}
```

관리형 통합에 대한 규정 준수 검증

AWS 서비스 가 특정 규정 준수 프로그램의 범위 내에 있는지 알아보려면 AWS 서비스 규정 준수 프로그램 범위 내를 참조하고 관심 있는 규정 준수 프로그램을 선택합니다. 일반 정보는 AWS 규정 준수 프로그램.

를 사용하여 타사 감사 보고서를 다운로드할 수 있습니다 AWS Artifact. 자세한 내용은 Downloading Reports inDownloading AWS Artifact을 참조하세요.

사용 시 규정 준수 책임은 데이터의 민감도, 회사의 규정 준수 목표, 관련 법률 및 규정에 따라 AWS 서비스 결정됩니다. 사용 시 규정 준수 책임에 대한 자세한 내용은 AWS 보안 설명서를 AWS 서비스참조하세요.

인터페이스 VPC 엔드포인트와 관리형 통합 사용

인터페이스 Amazon VPC 엔드포인트를 생성하여 Amazon VPC와 AWS IoT 관리형 통합 간에 프라이빗 연결을 설정할 수 있습니다. 인터페이스 엔드포인트는 프라이빗 IP 주소를 사용하여 서비스에 비공개로 액세스할 수 있는 AWS PrivateLink기술로 구동됩니다. AWS PrivateLink는 VPC와 IoT 관리형 통합 간의 모든 네트워크 트래픽을 Amazon 네트워크로 제한합니다. 인터넷 게이트웨이, NAT 디바이스 또는 VPN 연결이 필요하지 않습니다.

반드시를 사용할 필요는 AWS PrivateLink없지만 권장됩니다. AWS PrivateLink 및 VPC 엔드포인트에 대한 자세한 내용은 AWS PrivateLink 가이드의 를 통한 AWS 서비스 액세스를 AWS PrivateLink 참조하세요.

주제

- AWS IoT 관리형 통합 VPC 엔드포인트에 대한 고려 사항
- AWS IoT 관리형 통합을 위한 인터페이스 VPC 엔드포인트 생성
- VPC 엔드포인트 테스트
- VPC 엔드포인트를 통해 서비스에 대한 액세스 제어
- 요금
- 제한 사항

AWS IoT 관리형 통합 VPC 엔드포인트에 대한 고려 사항

AWS IoT 관리형 통합을 위한 인터페이스 VPC 엔드포인트를 설정하기 전에 AWS PrivateLink 가이드의 인터페이스 엔드포인트 속성 및 제한 사항을 검토하세요.

AWS IoT 관리형 통합은 인터페이스 VPC 엔드포인트를 통해 VPC에서 모든 API 작업을 호출할 수 있도록 지원합니다.

지원되는 엔드포인트

AWS IoT 관리형 통합은 다음 서비스 인터페이스에 대해 VPC 엔드포인트를 지원합니다.

- 컨트롤 플레인 API: `com.amazonaws.region.iotmanagedintegrations.api`

지원되지 않는 엔드포인트

다음 AWS IoT 관리형 통합 엔드포인트는 VPC 엔드포인트를 지원하지 않습니다.

- MQTT 엔드포인트: MQTT 디바이스는 일반적으로 AWS VPCs가 아닌 최종 사용자 환경에 배포되므로 AWS PrivateLink 통합이 필요하지 않습니다.
- OAuth 콜백 엔드포인트: 많은 타사 플랫폼이 AWS 인프라 내에서 작동하지 않으므로 OAuth 흐름 AWS PrivateLink 지원의 이점이 줄어듭니다.

가용성

AWS IoT 관리형 통합 VPC 엔드포인트는 다음 AWS 리전에서 사용할 수 있습니다.

- 캐나다(중부) - ca-central-1
- 유럽(아일랜드) - eu-west-1

AWS IoT 관리형 통합의 가용성이 확장됨에 따라 추가 리전이 지원됩니다.

듀얼 스택 지원

AWS IoT 관리형 통합 VPC 엔드포인트는 IPv4 및 IPv6 트래픽을 모두 지원합니다. 다음 IP 주소 유형을 사용하여 VPC 엔드포인트를 생성할 수 있습니다.

- IPv4: 엔드포인트 네트워크 인터페이스에 IPv4 주소 할당
- IPv6: 엔드포인트 네트워크 인터페이스에 IPv6 주소 할당(IPv6-only 서브넷 필요)
- 듀얼 스택: 엔드포인트 네트워크 인터페이스에 IPv4 및 IPv6 주소를 모두 할당합니다.

AWS IoT 관리형 통합을 위한 인터페이스 VPC 엔드포인트 생성

Amazon VPC 콘솔 또는 AWS CLI (AWS CLI)를 사용하여 AWS IoT 관리형 통합 서비스에 대한 VPC 엔드포인트를 생성할 수 있습니다.

AWS IoT 관리형 통합을 위한 인터페이스 VPC 엔드포인트를 생성하려면(콘솔)

1. Amazon VPC 콘솔에서 Amazon VPC 콘솔을 엽니다.

2. 탐색 창에서 엔드포인트를 선택합니다.
3. 엔드포인트 생성을 선택합니다.
4. 서비스 범주(Service category)에서 AWS 서비스를 선택합니다.
5. 서비스 이름에서 해당 AWS 리전에 해당하는 서비스 이름을 선택합니다. 예:
   - `com.amazonaws.ca-central-1.iotmanagedintegrations.api`
   - `com.amazonaws.eu-west-1.iotmanagedintegrations.api`
6. VPC에서 AWS IoT 관리형 통합에 액세스할 VPC를 선택합니다.
7. 추가 설정의 경우 DNS 이름 활성화가 기본적으로 선택됩니다. 이 설정을 유지하는 것이 좋습니다. 이렇게 하면 AWS IoT 관리형 통합 퍼블릭 서비스 엔드포인트에 대한 요청이 Amazon VPC 엔드포인트로 확인됩니다.
8. 서브넷에서 엔드포인트 네트워크 인터페이스를 생성할 서브넷을 선택합니다. 가용 영역당 서브넷 한 개를 선택할 수 있습니다.
9. IP 주소 유형에서 다음 옵션 중에서 선택합니다.
   - IPv4: 엔드포인트 네트워크 인터페이스에 IPv4 주소 할당
   - IPv6: 엔드포인트 네트워크 인터페이스에 IPv6 주소 할당(선택한 모든 서브넷이 IPv6-only인 경우에만 지원됨)
   - 듀얼 스택: 엔드포인트 네트워크 인터페이스에 IPv4 및 IPv6 주소 모두 할당
10. 보안 그룹에서 엔드포인트 네트워크 인터페이스에 연결할 보안 그룹을 선택합니다. 보안 그룹 규칙은 엔드포인트 네트워크 인터페이스와 서비스와 통신하는 VPC의 리소스 간의 통신을 허용해야 합니다.
11. 정책에서 인터페이스 엔드포인트를 통해 모든 리소스에 대한 모든 보안 주체의 모든 작업을 허용하려면 전체 액세스를 선택합니다. 액세스를 제한하려면 사용자 지정을 선택하고 정책을 지정합니다.
12. (선택 사항) 태그를 추가하려면 Add new tag(새 태그 추가)를 선택하고 태그 키와 태그 값을 입력합니다.
13. 엔드포인트 생성을 선택합니다.

IoT 관리형 통합을 위한 인터페이스 VPC 엔드포인트를 생성하려면(AWS CLI)

create-vpc-endpoint 명령을 사용하여 엔드포인트를 사용할 VPC ID, VPC 엔드포인트 유형(인터페이스), 서비스 이름, 서브넷 및 엔드포인트 네트워크 인터페이스와 연결할 보안 그룹을 지정합니다.

```
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
```

VPC 엔드포인트 테스트

VPC 엔드포인트를 생성한 후 VPC의 EC2 인스턴스에서 AWS IoT 관리형 통합에 대한 API 호출을 수행하여 연결을 테스트할 수 있습니다.

사전 조건

- VPC 내 프라이빗 서브넷의 EC2 인스턴스
- AWS IoT 관리형 통합 작업에 대한 적절한 IAM 권한
- VPC 엔드포인트에 대한 HTTPS 트래픽(포트 443)을 허용하는 보안 그룹 규칙

연결 테스트

1. 프라이빗 서브넷의 Amazon EC2 인스턴스에 연결합니다.
2. 프라이빗 DNS 이름에 대한 DNS 확인 확인:

```
dig api.iotmanagedintegrations.region.api.aws
```

3. HTTPS 연결 테스트:

```
curl -v https://api.iotmanagedintegrations.region.api.aws
```

4. AWS IoT 관리형 통합 API 직접 호출을 수행합니다.

```
aws iot-managed-integrations list-destinations \
  --region region \
  --endpoint-url https://api.iotmanagedintegrations.region.api.aws
```

를 AWS 리전(예: `ca-central-1`)`region`으로 바꿉니다.

VPC 엔드포인트를 통해 서비스에 대한 액세스 제어

VPC 엔드포인트 정책은 엔드포인트를 생성하거나 수정할 때 인터페이스 VPC 엔드포인트에 연결하는 IAM 리소스 정책입니다. 엔드포인트를 만들 때 정책을 추가하지 않으면 서비스에 대한 모든 액세스를 허용하는 기본 정책이 추가됩니다. 엔드포인트 정책은 IAM 사용자 정책 또는 서비스별 정책을 재정의하거나 대체하지 않습니다. 이는 엔드포인트에서 지정된 서비스로의 액세스를 제어하기 위한 별도의 정책입니다.

엔드포인트 정책은 JSON 형식으로 작성해야 합니다. 자세한 내용은 Amazon VPC 사용 설명서의 VPC 종단점을 통해 서비스에 대한 액세스 제어를 참조하세요.

예: AWS IoT 관리형 통합 작업에 대한 VPC 엔드포인트 정책

다음은 AWS IoT 관리형 통합에 대한 엔드포인트 정책의 예입니다. 이 정책은 VPC 엔드포인트를 통해 AWS IoT 관리형 통합에 연결하는 사용자가 대상에 액세스할 수 있도록 허용하지만 자격 증명 로커에 대한 액세스는 거부합니다.

```
{
    "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": "*"
```

```
        }
    ]
}
```

예: 특정 IAM 역할에 대한 액세스를 제한하는 VPC 엔드포인트 정책

다음 VPC 엔드포인트 정책은 신뢰 체인에 지정된 IAM 역할이 있는 IAM 보안 주체에 대해서만 AWS IoT 관리형 통합에 대한 액세스를 허용합니다. 다른 모든 IAM 보안 주체는 액세스가 거부됩니다.

```
{
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "*",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:PrincipalArn": "arn:aws:iam::123456789012:role/
IoTManagedIntegrationsVPCRole"
                }
            }
        }
    ]
}
```

요금

AWS IoT 관리형 통합을 통해 인터페이스 VPC 엔드포인트를 생성하고 사용하는 경우 표준 요금이 부과됩니다. 자세한 내용은 AWS PrivateLink 요금을 참조하십시오.

제한 사항

- CreateAccountAssociation API는 타사 클라우드 서비스에서 OAuth를 수행하도록 설계되었으며, 이를 위해서는 Amazon 네트워크에서 나가기 위한 요청이 필요합니다. 이는 AWS PrivateLink 를 사용하여 VPC 내에서 트래픽을 포함하는 고객에게 중요합니다.는이 API 호출에 대한 완전한 end-to-end 제한을 제공할 AWS PrivateLink 수 없기 때문입니다.
- AWS IoT 관리형 통합을 위한 VPC 엔드포인트는에서 사용할 수 없습니다 AWS GovCloud (US) Regions.

일반적인 VPC 엔드포인트 제한 사항은 Amazon VPC 사용 설명서의 인터페이스 엔드포인트 속성 및 제한을 참조하세요.

AWS IoT Device Management FIPS 엔드포인트의 관리형 통합에 연결

AWS IoT 는 FIPS(Federal Information Processing Standard) 140-2를 지원하는 컨트롤 플레인 엔드포인트를 제공합니다. FIPS 준수 엔드포인트는 표준 AWS 엔드포인트와 다릅니다. FIPS 준수 방식으로에 대한 AWS IoT Device Management 관리형 통합과 상호 작용하려면 FIPS 준수 클라이언트와 함께 아래 설명된 엔드포인트를 사용해야 합니다. AWS IoT 콘솔은 FIPS를 준수하지 않습니다.

다음 섹션에서는 REST API, SDK 또는를 사용하여 FIPS 준수 AWS IoT 엔드포인트에 액세스하는 방법을 설명합니다 AWS CLI.

컨트롤 플레인 엔드포인트

관리형 통합 작업 및 관련 AWS CLI 명령을 지원하는 FIPS 준수 컨트롤 플레인 엔드포인트는 서비스별 FIPS 엔드포인트에 나열됩니다. 서비스별 FIPS 엔드포인트에서 AWS IoT Device Management - 관리형 통합 서비스를 찾고의 엔드포인트를 조회합니다 AWS 리전.

관리형 통합 작업에 액세스할 때 FIPS 준수 엔드포인트를 사용하려면 AWS SDK 또는 REST API를에 적합한 엔드포인트와 함께 사용합니다 AWS 리전.

관리형 통합 CLI 명령을 실행할 때 FIPS 준수 엔드포인트를 사용하려면에 적합한 엔드포인트가 있는 --endpoint 파라미터를 AWS 리전 명령에 추가합니다.

관리형 통합 모니터링

모니터링은 관리형 통합 및 기타 AWS 솔루션의 안정성, 가용성 및 성능을 유지하는 데 중요한 부분입니다. AWS는 관리형 통합을 모니터링하고, 이상이 있을 때 보고하고, 적절한 경우 자동 조치를 취할 수 있는 다음과 같은 모니터링 도구를 제공합니다.

- AWS CloudTrail는 AWS 계정에 의해 또는 계정을 대신하여 수행된 API 호출 및 관련 이벤트를 캡처하고 사용자가 지정한 Amazon S3 버킷에 로그 파일을 전송합니다. 호출한 사용자 및 계정 AWS, 호출이 수행된 소스 IP 주소, 호출이 발생한 시기를 식별할 수 있습니다. 자세한 내용은 AWS CloudTrail 사용 설명서를 참조하십시오.

를 사용하여 관리형 통합 API 호출 로깅 AWS CloudTrail

관리형 통합은 사용자AWS CloudTrail, 역할 또는가 수행한 작업에 대한 레코드를 제공하는 서비스인와 통합됩니다 AWS 서비스. CloudTrail은 관리형 통합에 대한 모든 API 호출을 이벤트로 캡처합니다. 캡처되는 호출에는 관리형 통합 콘솔의 호출과 관리형 통합 API 작업에 대한 코드 호출이 포함됩니다. CloudTrail에서 수집한 정보를 사용하여 관리형 통합에 수행된 요청, 요청이 수행된 IP 주소, 요청이 수행된 시간 및 추가 세부 정보를 확인할 수 있습니다.

모든 이벤트 또는 로그 항목에는 요청을 생성했던 사용자에 대한 정보가 포함됩니다. 자격 증명을 이용하면 다음을 쉽게 판단할 수 있습니다.

- 요청을 루트 사용자로 했는지 사용자 보안 인증으로 했는지 여부.
- IAM Identity Center 사용자를 대신하여 요청이 이루어졌는지 여부입니다.
- 역할 또는 페더레이션 사용자에 대한 임시 자격 증명을 사용하여 요청이 생성되었는지 여부.
- 다른 AWS 서비스에서 요청했는지 여부.

CloudTrail은 계정을 생성할 AWS 계정 때에서 활성화되며 CloudTrail 이벤트 기록에 자동으로 액세스할 수 있습니다. CloudTrail 이벤트 기록은 지난 90일 간 AWS 리전의 관리 이벤트에 대해 보기, 검색 및 다운로드가 가능하고, 수정이 불가능한 레코드를 제공합니다. 자세한 설명은 AWS CloudTrail 사용 설명서의 CloudTrail 이벤트 기록 작업을 참조하세요. Event history(이벤트 기록) 보기는 CloudTrail 요금이 부과되지 않습니다.

AWS 계정 지난 90일 동안의 이벤트를 지속적으로 기록하려면 추적 또는 CloudTrail Lake 이벤트 데이터 스토어를 생성합니다.

CloudTrail 추적

CloudTrail은 추적을 사용하여 Amazon S3 버킷으로 로그 파일을 전송할 수 있습니다. 를 사용하여 생성된 모든 추적 AWS Management Console 은 다중 리전입니다. AWS CLI를 사용하여 단일 리전 또는 다중 리전 추적을 생성할 수 있습니다. 계정 AWS 리전 의 모든에서 활동을 캡처하므로 다중 리전 추적을 생성하는 것이 좋습니다. 단일 리전 추적을 생성하는 경우 추적의 AWS 리전에 로깅된 이벤트만 볼 수 있습니다. 추적에 대한 자세한 내용은 AWS CloudTrail 사용 설명서의 Creating a trail for your AWS 계정 및 Creating a trail for an organization을 참조하세요.

CloudTrail에서 추적을 생성하여 진행 중인 관리 이벤트의 사본 하나를 Amazon S3 버킷으로 무료로 전송할 수는 있지만, Amazon S3 스토리지 요금이 부과됩니다. CloudTrail 요금에 대한 자세한 내용은 AWS CloudTrail 요금을 참조하세요. Amazon S3 요금에 대한 자세한 내용은 Amazon S3 요금을 참조하세요.

CloudTrail Lake 이벤트 데이터 스토어

CloudTrail Lake를 사용하면 이벤트에 대해 SQL 기반 쿼리를 실행할 수 있습니다. CloudTrail Lake는 행 기반 JSON 형식의 기존 이벤트를 Apache ORC 형식으로 변환합니다. ORC는 빠른 데이터 검색에 최적화된 열 기반 스토리지 형식입니다. 이벤트는 이벤트 데이터 스토어로 집계되며, 이벤트 데이터 스토어는 고급 이벤트 선택기를 적용하여 선택한 기준을 기반으로 하는 변경 불가능한 이벤트 컬렉션입니다. 이벤트 데이터 스토어에 적용하는 선택기는 어떤 이벤트가 지속되고 쿼리할 수 있는지 제어합니다. CloudTrail Lake에 대한 자세한 내용은 AWS CloudTrail 사용 설명서의 AWS CloudTrail Lake 작업을 참조하세요.

CloudTrail Lake 이벤트 데이터 스토어 및 쿼리에는 비용이 발생합니다. 이벤트 데이터 스토어를 생성할 때 이벤트 데이터 스토어에 사용할 요금 옵션을 선택합니다. 요금 옵션에 따라 이벤트 모으기 및 저장 비용과 이벤트 데이터 스토어의 기본 및 최대 보존 기간이 결정됩니다. CloudTrail 요금에 대한 자세한 내용은 AWS CloudTrail 요금을 참조하세요.

CloudTrail의 관리 이벤트

관리 이벤트는의 리소스에서 수행되는 관리 작업에 대한 정보를 제공합니다 AWS 계정. 이를 컨트롤 플레인 작업이라고도 합니다. 기본적으로 CloudTrail은 관리 이벤트를 로깅합니다.

관리형 통합은 다음과 같은 관리형 통합 컨트롤 플레인 작업을 관리 이벤트로 CloudTrail에 로깅합니다.

- `CreateCloudConnector`
- `UpdateCloudConnector`

- GetCloudConnector
- DeleteCloudConnector
- ListCloudConnectors
- CreateConnectorDestination
- UpdateConnectorDestination
- GetConnectorDestination
- DeleteConnectorDestination
- ListConnectorDestinations
- CreateAccountAssociation
- UpdateAccountAssociation
- GetAccountAssociation
- DeleteAccountAssociation
- ListAccountAssociations
- StartAccountAssociationRefresh
- ListManagedThingAccountAssociations
- RegisterAccountAssociation
- DeregisterAccountAssociation
- SendConnectorEvent
- ListDeviceDiscoveries
- ListDiscoveredDevices

이벤트 예

이벤트는 모든 소스로부터의 단일 요청을 나타내며 요청된 API 작업, 작업 날짜와 시간, 요청 파라미터 등에 대한 정보가 들어 있습니다. CloudTrail 로그 파일은 퍼블릭 API 직접 호출의 주문 스택 추적이 아니므로 이벤트가 특정 순서로 표시되지 않습니다.

다음 예제에서는 성공적인 CreateCloudConnector API 작업을 보여주는 CloudTrail 이벤트를 보여줍니다.

CreateCloudConnector API 작업을 통한 CloudTrail 이벤트 성공.

```
{
    "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"
}
```

다음 예제에서는 성공적인 `ListDiscoveredDevices` API 작업을 보여주는 CloudTrail 이벤트를 보여줍니다.

`ListDiscoveredDevices` API 작업을 통한 CloudTrail 이벤트 성공.

```
{
    "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"
}
```


CloudTrail 레코드 콘텐츠에 대한 자세한 내용은 AWS CloudTrail 사용 설명서의 CloudTrail record contents를 참조하세요.

관리형 통합 개발자 안내서의 문서 기록

다음 표에서는 관리형 통합에 대한 설명서 릴리스를 설명합니다.

변경 사항	설명	날짜
정식 출시 릴리스	관리형 통합 개발자 안내서의 정식 릴리스	2025년 6월 25일
초기 미리 보기 릴리스	관리형 통합 개발자 안내서의 최초 미리 보기 릴리스	2025년 3월 3일

기계 번역으로 제공되는 번역입니다. 제공된 번역과 원본 영어의 내용이 상충하는 경우에는 영어 버전이 우선합니다.